Alois Lechicki (Fürth, Germany; E-mail: mail@lechicki.de)
Posted date: February 23, 2026
https://www.doi.org/10.13140/RG.2.2.12347.68645
[Version 1.0]

Artificial Intelligence: What Is It and What Can It Do?

Artificial Intelligence as understood by a human

Summary

Artificial Intelligence (AI) is not just an abstract field for experts. It is already having a pervasive impact on daily life as a practical technology that influences our economic opportunities and societal norms. AI is important for everyone because it is reshaping almost every aspect of our lives, from the way we work and make decisions to how we access services and participate in civic society. AI drives everything from personalised recommendations on streaming services to critical decision-making in sectors like finance and healthcare. As AI systems become increasingly integrated into our everyday tools and services, it is important for individuals to understand the principles on which they are based. This allows them to navigate the digital world, appreciate the convenience that AI offers, and be sceptical of any potential biases or hidden agendas that could affect their personal data or the outcomes of decisions.

This essay examines the concept of artificial intelligence, its origins, operational mechanisms, and capabilities. It attempts to explain the key principles and algorithms, and their real-world applications. The paper is primarily intended for individuals interested in the technical foundations of AI. However, the author believes that a basic understanding of these principles is necessary for anyone who wants to grasp the technology's potential, limitations and associated risks. Consequently, this text may also be of interest to readers with less of a technical background.

Contents

Before we work on artificial intelligence why don't we do something about natural stupidity?
– Stephen Polyak ([1])

1. Introduction

Artificial intelligence has recently become an increasingly prevalent part of modern life, influencing domains ranging from medicine, science, education, finance, and engineering to entertainment, art, and military applications.

The notion of *Artificial Intelligence* is used to denote the simulation of human intelligence in machines that have been programmed to think, learn, and problem-solve in a manner analogous to that of humans. These systems have the capacity to perform tasks that typically require human intelligence, such as speech recognition, decision-making, and language translation. Many of the algorithms, design principles, and concepts used in AI today, such as neural networks or reinforcement learning, have their origins in biology and psychology.

The term 'Artificial Intelligence', or AI, was first coined seventy years ago by the American computer scientist John McCarthy during the 2nd Dartmouth Conference in 1956, with the definition being originally described as a means for manufactured devices to emulate or even exceed the capabilities of humans to perform mental tasks. AI today upholds a similar definition, anchored on enabling machines to think and operate in a similar manner to the human brain. The fundamental principle underlying AI is the analysis of patterns in human and machine behaviour to facilitate intelligent problem-solving across a variety of contexts. ([T1])

Artificial intelligence is transforming the way we live, work and interact with the world around us. One of the most transformative technologies of the 21st century, AI encompasses a wide variety of capabilities, including machine learning, natural language processing, robotics, and computer vision. Significant advancements in computational power, data availability and algorithmic innovation over the decades have elevated AI to new levels. Nowadays, AI systems can analyse vast amounts of data, recognise patterns, and make decisions with remarkable accuracy.

AI is being integrated into various industries, from healthcare and finance to transportation and entertainment, driving efficiency and innovation. In healthcare, for example, AI can help with disease diagnosis and personalising treatment plans. In finance, AI-powered algorithms optimise investment strategies and detect fraudulent activities. Autonomous vehicles, powered by AI, promise to make transportation safer and more efficient.

AI is sometimes referred to as an industry, reflecting its shift from research to large-scale production and denoting the rapidly expanding economic sector devoted to developing and commercialising artificial intelligence technologies. Conversely, AI is often described as an enabler, emphasising its function as a fundamental technology that enhances existing systems to drive innovation and efficiency across various fields, rather than operating as a final product itself.

However, as AI continues to become more advanced, important ethical and societal questions arise. Its potential to outperform humans in certain tasks calls for a revision of our roles and responsibilities. It is vital that AI is developed and used in a responsible way to maximise its benefits and minimise risks. One of the less obvious risks is the potential atrophy of human cognitive abilities through overreliance on AI systems. Psychological research has shown ([R15]) that when individuals consistently delegate mental tasks to external devices or algorithms – a process known as 'cognitive offloading' – their capacity for independent reasoning, memory retention, and problem-solving can diminish over time. Just as physical

[1] Stephen Polyak (1889 – 1955) was an American neuroanatomist and neurologist considered to be one of the most prominent neuroanatomists of the 20th century.

muscles weaken without regular use, our mental flexibility may deteriorate if we routinely outsource complex thinking to AI systems. For example, an MIT research team reported that students who used ChatGPT to write essays incurred cognitive debt, with users performing consistently worse in terms of their neural, linguistic, and behavioural abilities. These worrying results show that more research is needed into the effect of AI on learning ([K22], [S27]). Another example is a recent study published in The Lancet Gastroenterology & Hepatology, which found that gastroenterologists who had become accustomed to an AI-assisted colonoscopy system performed around 20% worse at identifying polyps and other abnormalities when working without AI assistance. Over a period of just six months, the authors observed that clinicians became "*less motivated, less focused and less responsible when making cognitive decisions without AI assistance*" (([B27], [S27]).

Moreover, placing too much trust in AI systems can be risky. Despite their impressive capabilities, they are not infallible. They can produce inaccurate, biased or entirely fabricated results, a phenomenon commonly referred to as 'hallucination'. These errors can be caused by limitations in training data, model design or contextual understanding. Consequently, users must adopt a critical approach, verifying AI-generated information and ensuring that automated outputs are subject to human oversight and validation.

This essay explores the question of what artificial intelligence is, where it comes from, how it works and what it can do. We attempt to shed light on its key concepts and fundamental algorithms, as well as its real-world applications. By understanding the technical capabilities and limitations of AI, we can better anticipate its potential for the future.

This text is intended as an introduction to artificial intelligence for beginners, providing a general overview of the subject. A reasonable level of mathematics is assumed. While it is possible to gain a basic understanding of how AI works without an in-depth mathematical knowledge, mathematics is essential for properly grasping the technical details. This does not mean, however, that readers need to fully comprehend all mathematical subtleties. It is possible to capture the essence of the discussion without detailed familiarity with the mathematical concepts involved.

The chapters of this article are organised to guide readers step by step into the world of artificial intelligence.

Chapter 2 begins by explaining what AI actually is and how the field developed over time, offering a historical foundation for understanding today's technologies.

Chapter 3 introduces the main areas of AI and the fundamental algorithms that support them, from early statistical methods to modern neural-network models.

Chapter 4 provides a more detailed look at machine learning, describing how machines learn from data, the different forms this learning can take, and the strengths and limitations of these approaches.

Chapter 5 focuses on natural language processing, exploring how computers process human language and how large language models and the Transformer architecture work in practice.

Finally, Chapter 6 looks ahead to the next decade, discussing how AI may evolve, where it is likely to be applied, and how these developments could influence society and everyday life.

2. What is Artificial Intelligence, and how did it all start?

Computers are like humans –
they do everything except think.
– John von Neumann ([2])

This chapter provides the conceptual framework and historical context for understanding the current AI landscape, offering both an introduction to the nature of artificial intelligence and a concise historical overview of its development. By tracing the evolution of AI from theoretical enquiries to large-scale generative architectures, it illustrates how the field has progressed through cycles of optimism, challenge, and innovation to become a transformative force in the modern world.

2.1. What is Artificial Intelligence?

After a period of 60 years, the field of Artificial Intelligence has now become a part of industry and is also a topic of discussion among the population at large. According to the Encyclopaedia Britannica ([B1]), AI can be defined as

> *AI is the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings*.

However, this definition is not without its limitations. For instance, it would permit the conclusion that a computer with substantial memory capacity capable of saving an extensive text and subsequently retrieving it on demand would be regarded as intelligent. Consequently, according to this definition, every computer could be considered an AI system. The following characterisation of artificial intelligence by Elaine Rich [R1] provides a resolution to the issue ([E1]):

> *Artificial Intelligence is the study of how to make computers do things at which, at the moment, people are better.*

A fundamental goal of AI research is to build a machine whose behaviour is comparable to that of a human being, i.e. a machine that exhibits human-like intelligence. It is not required that AI systems employ the same mechanisms (whatever they may be) that underlie human cognition, nor that they undergo the developmental or learning stages characteristic of humans. The objective is to produce human-like intelligence, not to imitate the biological processes that give rise to it. Nevertheless, biological cognition can provide valuable inspiration for developing new AI models and algorithms ([J1]).

The Turing Test, devised by the British mathematician and computer scientist Alan Turing in 1950, is one of the most important concepts in the field of artificial intelligence ([T2]). It serves as a seminal exploration of the ability of machines to exhibit intelligent behaviour indistinguishable from that of human beings. Turing proposed that if a human interrogator could not reliably tell whether a respondent was a machine or a human based solely on their answers to questions, then the machine would be considered to have passed the test. ([T8])

Here is a breakdown of the Turing Test:

1. Setup. In the Turing Test, a human judge (let's call him Person A) interacts with two entities – one is a human (Person B) and the other is a machine (entity C). The interactions are typically conducted via a text-based interface to avoid any bias based on the physical appearance or voice of the participants.

---

[2] John von Neumann (1903–1957) was a Hungarian-American mathematician, physicist, computer scientist, and polymath who made significant contributions to various fields. He is considered one of the most influential mathematicians of the 20th century.

2. Goal. Person A's task is to interact with both Person B and entity C and determine which is the machine. The machine's goal is to convince Person A that it is human.

3. Evaluation. If Person A cannot distinguish one from the other with significantly greater than 50% accuracy after many attempts, and this result remains consistent regardless of which Person B is involved in the experiment, the machine is said to have passed the Turing Test. Here, 50% accuracy means that Person A performs, on average, no better than guessing at random when identifying which participant is human and which is the machine. However, if his accuracy is significantly above 50%, the machine can still be distinguished from a human, meaning it fails the test. Conversely, if the evaluator's accuracy falls significantly below 50%, this implies that the machine appears more human than the human participant. This would mean that the machine exhibits behaviour indistinguishable from that of a human and thus demonstrates a form of artificial intelligence.

Language plays a central role in the Turing Test as it is the medium through which the interrogation takes place. It tests the machine's ability to understand questions, formulate responses, grasp nuance and respond in a way that conveys an understanding of context and emotion. Turing's focus on language emphasises the importance of communication as a measure of intelligence. ([T8])

However, critics argue that the Turing Test focuses too much on language and behaviour, ignoring the rich cognitive processes that underpin true intelligence. By privileging outward responses, we may be overlooking the potential limitations of machines in understanding and reasoning. Regardless of a computer's ability to pass the Turing Test, there is no real way to tell whether or not a machine really understands human semantics. The test simply judges machines on their ability to communicate with human-like eloquence, not human-like understanding. This limitation has led some AI researchers to argue that the Turing Test is less relevant than it once was. ([K1])

There have been several instances where AI systems have come close to passing the Turing Test, although it's a subject of ongoing debate. Here are some notable examples:

- **Cleverbot** (2011). Cleverbot is an AI chatbot that has been interacting with users since 1997. Although it hasn't officially passed the Turing Test, it has been able to engage in conversations that many users have found convincingly human-like. ([A1])

- **Eugene Goostman** (2014). A chatbot called Eugene Goostman, which simulated a 13-year-old Ukrainian boy, was claimed to have passed the Turing Test at an event organised by the University of Reading. It convinced 33% of the judges that it was human. ([T3])

- **ChatGPT** (2023). Researchers at UC San Diego published a paper claiming that ChatGPT, an AI language model, had passed the Turing Test. This was based on its ability to engage in conversations indistinguishable from those with a human. ([G1])

These examples highlight the progress made in AI development, but it's important to note that the Turing Test is only one measure of AI's capabilities. The field continues to evolve, and new benchmarks and tests are being developed to assess the intelligence and human-like behaviour of AI.

In summary, although Alan Turing devised an influential test for determining whether machines can think, the Turing Test is not a sufficient indicator of artificial intelligence. It evaluates only whether a machine's responses are indistinguishable from those of a human, without considering whether the machine actually understands the meaning of its inputs and outputs – that is, whether it can relate symbols and words to the concepts or situations they represent ([3]). Moreover, the test does not account for a machine's ability to recognise patterns, to reason about the world, or to apply common knowledge and common sense. ([K1])

---

[3] This issue is central to John Searle's 'Chinese Room' argument ([S26]), which challenges the idea that symbol manipulation alone constitutes understanding. According to Searle, a system may process information syntactically – following formal rules – without any genuine comprehension of meaning (semantics).

2.2. A brief history of Artificial Intelligence

The history of artificial intelligence is a fascinating journey that spans centuries, with significant milestones and advances along the way. Here's a brief overview (see [M1], [M2], [T4] and [W1] for more details):

- **Early Beginnings.** The concept of artificial beings with intelligence dates back to ancient myths and legends. Automatons, mechanical devices that could move independently, were created by inventors like Leonardo da Vinci.

The idea of artificial intelligence dates back thousands of years to ancient. In antiquity, inventors created things called 'automatons' that moved mechanically without human intervention. The word 'automaton' means 'acting of its own will'. One of the earliest records of an automaton dates back to 400 B.C. and refers to a mechanical pigeon built by a friend of the philosopher Plato. Many years later, mechanical devices that could move on their own were created by Leonardo da Vinci around 1495.

Jonathan Swift's "*Gulliver's Travels*" (1726) introduced the vision of a machine that could generate new ideas, sentences and books. Swift's fantasy anticipates the concept of algorithmic text generation, which is now a reality with modern AI. AI models can produce coherent text by combining words and ideas based on underlying algorithms, similar to what Swift's fictional Engine is intended to do.

- **Groundwork for AI** (1900-1950). In the early 1900s, a lot of literature was written about the idea of artificial humans. So much that scientists of all kinds began to ask the question: is it possible to create an artificial brain? Some creators even made some versions of what we now call 'robots', the word introduced by the Czech playwright Karel Čapek in his 1921 play "*Rossum's Universal Robots*". However, most of them were relatively simple. They were mostly steam-powered, and some could make facial expressions and even walk. Although Čapek's robots are organic, the word 'robots' came to be associated with mechanical, humanoid machines designed to perform monotonous, unskilled labour.

In 1929, Japanese professor Makoto Nishimura built the first Japanese robot, called *Gakutensoku*.

In 1939, John Vincent Atanasoff, a professor of physics and mathematics at Iowa State College, and his graduate student Clifford Berry built the Atanasoff-Berry (ABC) computer at Iowa State University with a grant of $650. The ABC is considered one of the earliest digital electronic computers and a milestone in American computing. Although the ABC was never fully operational or widely used, it introduced several key concepts that would become fundamental to the development of modern computing. ABC used around 300 vacuum tubes for its logic operations, making it much faster than earlier mechanical calculators. The ABC weighed over 700 pounds and could solve up to 29 linear equations at the same time.

In the 1930s, Kurt Gödel, Alonso Church, and Alan Turing established foundational principles in logic and theoretical computer science that would prove instrumental in the development of AI. Among these seminal contributions was Gödel's theorems, specifically the completeness theorem, which asserted the completeness of first-order predicate logic. This implies that every true statement that can be expressed in predicate logic is also provable using the rules of a formal calculus ([4]). This paved the way for the subsequent development of automatic theorem

[4] First-order predicate logic (FOL) is a formal logical system that has found application in mathematics, computer science and artificial intelligence. It extends propositional logic by introducing the concepts of quantifiers and predicates, thus allowing for more expressive statements about objects and their relationships. Here are some key components and concepts of first-order predicate logic:
(1) *Variables*: Symbols that represent objects in a domain (e.g., x, y, z).
(2) *Predicates*: Functions that represent properties or relationships between objects (e.g., P(x), Q(x, y)).
(3) *Quantifiers*: Symbols that express the quantity of objects that satisfy a given predicate.
(4) *Connectives*: Logical operators such as AND (∧), OR (∨), NOT (¬), IMPLIES (→), and EQUIVALENT (↔).

provers as implementations of formal calculi. Notably, Gödel's incompleteness theorems reveal the existence of true statements that are unprovable within higher-order logics ([5]). ([E1])

Alan Turing's proof of the undecidability of the halting problem also falls into this time period. Turing's proof demonstrated the impossibility of identifying whether a given arbitrary program (and its respective input) would run in an infinite loop. This result also led to the identification of a limit for intelligent programs, and consequently, the conclusion that a universal program verification system is an unattainable prospect. ([E1])

In 1943, Warren S. McCulloch and Walter Pitts published "*A Logical Calculus of the Ideas Immanent in Nervous Activity"* in the Bulletin of Mathematical Biophysics ([M20]). It is one of the landmark papers in the history of neuroscience and AI. The paper introduces the concept of artificial neural networks, now a key technology in modern AI, and lays the foundation for the idea that the brain can be understood as a computational system. Nevertheless, the computing devices of that era were not yet capable of generating sufficient processing power to facilitate the simulation of rudimentary cognitive functions.

In 1949, the computer scientist Edmund Callis Berkley published "*Giant Brains, or Machines that Think"* ([B24]), in which he compared computers to the human brain.

- **Birth of AI** (1950-1956). It was during this period that interest in AI really took off. Alan Turing published his work "*Computer Machinery and Intelligence*" in 1950 ([T2]), which eventually became the Turing Test, used by experts to measure computer intelligence.

Marvin Minsky and Dean Edmunds built the first artificial neural network in 1951. The Stochastic Neural Analogue Reinforcement Calculator (SNARC) was an early attempt to model learning processes in the human brain, in particular reinforcement learning. The SNARC was designed to simulate the behaviour of a rat as it navigated through a maze. It was an analogue computer that used a network of 3.000 vacuum tubes along with synaptic weights to simulate 40 neuron-like units.

In 1952, Allen Newell, a mathematician and computer scientist, and Herbert A. Simon, a political scientist, created programs like the *Logical Theorist* and the *General Problem Solver*, among the first to emulate human ability to solve problems.

The term 'artificial intelligence' was first coined in 1955 in a workshop proposal entitled "*A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence*" ([M21]), submitted by John McCarthy of Dartmouth College, Marvin Minsky of Harvard University, Nathaniel Rochester of IBM and Claude Shannon of Bell Telephone Laboratories. The workshop held a year later in July and August 1956, is generally regarded as the official birth date of the emerging field of AI.

- **AI Maturation** (1957-1979). The period between the coining of the term 'artificial intelligence' and the 1980s was a period of both rapid growth and struggle for AI research. The late 1950's and 1960's was a time of creating, of building. AI quickly became a mainstream idea, from programming languages still in use today to books and films exploring the idea of robots. Similar improvements were seen in the 1970s, but it was also a difficult time for AI research as the US government was not interested in providing further funding.

In 1957, psychology and computer scientist Frank Rosenblatt developed an early artificial neural network which enabled pattern recognition. Although limited to solving linearly separable problems (see Section 3.2.5), it laid the foundation for future developments in neural networks and machine learning.

John McCarthy developed the programming language Lisp, which stands for LISt Processing, in 1958. Lisp grew out of McCarthy's work on formalising algorithms and mathematical logic,

[5] Higher-order predicate logic (also known as higher-order logic or HOL) is a logical system that extends first-order logic by allowing quantification over predicates, functions, and relations, in addition to quantification over individual variables. This augmentation renders higher-order logic more expressive and powerful compared to first-order logic.

and was particularly influenced by his desire to create a programming language that could handle symbolic information. Lisp soon became the most popular programming language used in AI research.

In 1965, the philosopher Hubert Dreyfus published "*Alchemy and Artificial Intelligence*" ([D9]), arguing that the human mind works fundamentally differently from computers. He predicted that the challenges of replicating human intuition and understanding would limit the progress of AI. His critique was influential in sparking debates about the philosophical and practical limits of AI.

Edward Feigenbaum and Joshua Lederberg created the first 'expert system' in 1965, a form of AI programmed to mimic the thinking and decision-making of human experts.

In 1966, Joseph Weizenbaum created ELIZA, the first 'conversational robot' that used natural language processing (NLP) to interact with people. In 1968, the Soviet mathematician Alexey Ivakhnenko published "*Group Method of Data Processing*", which proposed a new approach to AI that would later be called 'Deep Learning'.

In 1972, James Lighthill presented a critical report to the British Science Research Council on the progress of AI research ([L9]), concluding that AI had failed to live up to its early promise and arguing that the field had not produced significant breakthroughs, leading to a drastic reduction in government funding for AI in the UK. This report contributed to the onset of the first AI winter, a period of reduced interest and investment in AI research.

- **AI Boom and Winter** (1980-1993). Most of the 1980s saw a period of rapid growth and interest in AI, now referred to as the 'AI boom'. This was due to both breakthroughs in research and additional government funding to support researchers. Deep learning techniques and the use of expert systems became popular, both of which allowed computers to learn from their mistakes and make decisions on their own.

The term AI winter first appeared in 1984 as the subject of a public debate at the annual meeting of the American Association for Artificial Intelligence (AAAI). It referred to the hype generated by over-promising developers, unrealistically high expectations from end-users, and extensive media promotion.

Notable dates during this period include:

In 1980, the first expert system, known as XCON (expert configurer), entered the commercial market. It was designed to assist in the ordering of computer systems by automatically selecting components based on the customer's needs.

In 1986, David Rumelhart, Geoffrey Hinton and Ronald Williams published the seminal paper "*Learning representations by back-propagating errors*" ([R7]), describing the back-propagation algorithm. The back-propagation algorithm becomes the foundation of modern deep learning, creating a renewed interest in neural networks and overcoming some of the limitations highlighted in earlier AI research. This breakthrough made artificial neural networks viable for practical applications and opened the door to the deep learning revolution of the 2000s and 2010s.

In 1986, Ernst Dickmann and his team at the University of the Bundeswehr in Munich created and demonstrated the first driverless car (or robot car). It could drive up to 55 mph on roads without obstacles or human drivers.

In the late 1980s, as the AAAI warned, an AI winter arrived. The term describes a period of low consumer, public and private interest in AI, leading to reduced research funding and few breakthroughs. Both private investors and the government lost interest in AI and stopped funding it because of the high costs for seemingly low returns. This AI winter came about because of some setbacks in the machine market and expert systems, cutbacks in strategic computing initiatives, and a slowdown in the deployment of expert systems.

- **AI Agents** (1993–2011). Despite the lack of funding during the AI winter, the early 1990s saw some impressive advances in AI research. This era also brought AI into everyday life with innovations such as the first Roomba and the first commercially available speech recognition software on Windows computers.

In 1997, Sepp Hochreiter and Jürgen Schmidhuber introduced Long Short-Term Memory (LSTM), a type of recurrent neural network (RNN) designed to overcome the limitations of traditional RNNs, in particular their inability to effectively capture long-term dependencies in data. LSTM networks are widely used in applications such as handwriting recognition, speech recognition, natural language processing and time series forecasting.

In the same year, Deep Blue (developed by IBM) defeated world chess champion Gary Kasparov in a high-profile match, becoming the first program to beat a human chess champion. Microsoft released speech recognition software for Windows (developed by Dragon Systems).

Geoffrey Hinton ([6]) published "*Learning Multiple Layers of Representation*" in 2006 ([H12]), which summarised key breakthroughs in deep learning and outlined how multi-layer neural networks could be trained more effectively. Hinton's work focused on training networks with graded connections ([7]) to generate sensory data, rather than simply classifying it. This approach, which allowed machines to learn complex hierarchical representations of data, represented a shift from traditional neural networks to what is now known as deep learning.

Rajat Raina, Anand Madhavan and Andrew Ng published "*Large-scale Deep Unsupervised Learning using Graphics Processors*" in 2009 ([R13]), arguing that graphics processing units (GPUs) could far outperform traditional multi-core CPUs for deep learning tasks. They showed that the superior computing power of GPUs could revolutionise the applicability of deep unsupervised learning methods, allowing researchers to train larger and more complex models more efficiently. This work helped to accelerate the adoption of GPUs in deep learning, paving the way for the breakthroughs of the 2010s that power modern AI applications in areas such as computer vision and natural language processing.

In 2011, Apple launched Siri, the first popular virtual assistant on the market.

- **AI today** (2012–present). This takes us to the latest AI developments to date. Virtual assistants, search engines and other commonly used AI tools are where we've seen a big increase. This period also saw the popularisation of deep learning and Big Data.

In 2012, Jeff Dean and Andrew Ng conducted an experiment using a massive neural network with 10 million unlabelled images from YouTube videos. During the experiment, the network learned to recognise patterns in the data without any prior labelling. This demonstration of unsupervised learning showed how deep neural networks can autonomously learn features from vast amounts of data.

Facebook programmed two AI chatbots in 2017 to converse and learn to negotiate, but as they went back and forth, they eventually abandoned English and developed their own language, completely autonomously. This development was unexpected, as the bots optimised their communication without human intervention. The experiment was stopped to keep the bots within human-understandable language, but the event highlights the potential for AI systems to evolve autonomously and unpredictably.

---

[6] Geoffrey Hinton (1947 –) is a British-Canadian computer scientist and cognitive psychologist who is often referred to as the 'godfather of AI' due to his pioneering work on neural networks, which has enabled significant advances in machine learning. In 2024, he was awarded the Nobel Prize in Physics.

[7] In Hinton's framework, graded connections are simply real-valued synaptic weights between units, enabling each neuron to respond smoothly rather than in an all-or-nothing binary fashion. Synaptic weights are the parameters that quantify how strongly the output of one neuron (or unit) influences the input of another (see Section 4.3.2).

OpenAI ([8]) unveiled 2020 GPT-3, a language model with 175 billion parameters, making it one of the largest and most sophisticated AI models to date. GPT-3 demonstrated the ability to generate human-like text, conversation, code, language translation and creative writing from natural language prompts. As one of the earliest examples of a large language model (LLM), it demonstrated the potential of AI to understand and produce highly coherent language.

In March 2023, Liang Wenfeng founded DeepSeek ([9]), which quickly gained international attention with its AI model, DeepSeek R1, released in January 2025. This model is designed to compete with established AI systems such as OpenAI's ChatGPT and Anthropic's Claude. DeepSeek R1 is known for its efficiency and cost-effectiveness, and claims to have been developed with significantly lower training costs than its competitors.

On 27 February 2025, OpenAI released GPT-4.5. This model is part of OpenAI's ongoing efforts to advance AI capabilities, and is available to ChatGPT Pro subscribers as part of a research preview. GPT-4.5 is designed to be more natural in conversations, with a broader knowledge base and improved ability to follow user intent. It also has higher emotional intelligence (EQ), making it better at understanding nuances and subtleties in communication. GPT-4.5 is expected to have less hallucination, i.e. it is less likely to generate false or fabricated information ([10]).

Released on 11 December 2025, GPT-5.2 is an enhanced version of the GPT-5 family. It offers improved reasoning depth and long-context stability, as well as more reliable multi-step problem solving. Seamlessly integrating with multimodal systems and tool-use frameworks, it enables richer workflows that combine text generation with external audio, image or search components. GPT-5.2 is now the default model for all logged-in ChatGPT users. Free users are routed to the lighter GPT-5.2 Instant model, while paid subscribers have access to the full GPT-5.2 family, including GPT-5.2 Auto and GPT-5.2 Thinking, which are better suited to more advanced reasoning tasks ([O4], [O5], [K12]).

- **Future of AI.** Now that we're back in the present, what's next for AI? No one can ever fully predict the future. However, many leading experts are talking about possible futures for AI, so one can make some educated guesses. You can expect to see further adoption of AI by businesses of all sizes, changes in the workforce as more automation eliminates and creates jobs, more robotics, autonomous vehicles, and more.

Experts believe that AI will become an integral part of many aspects of our personal and business lives within the next 10 years. Generative AI models such as GPT-4 have shown a great deal of promise in the short time that they've been available to the public, but their limitations have also become well known. As a result, the future of AI will be defined by a shift towards both open-sourcing large-scale models for experimentation and developing smaller, more efficient models to drive usability and lower costs.

This movement reflects a transition from exclusively large, closed models to more accessible and versatile AI solutions. These new models deliver greater accuracy with fewer resources. They are ideal for businesses that need to create tailored content or solve complex problems.

AI will become even more integrated into our personal and professional lives, driven by easy-to-use platforms that allow non-experts to use AI for business, personal tasks, research and creative projects.

---

[8] OpenAI was founded in December 2015 as a non-profit organisation by a team of eleven leading technologists and researchers, including Sam Altman, Elon Musk, and Ilya Sutskever ([W1]).

[9] DeepSeek is a Chinese artificial intelligence company headquartered in Hangzhou, Zhejiang Province, China. Liang Wenfeng, the co-founder of the High-Flyer hedge fund, also serves as DeepSeek's CEO.

[10] AI hallucination is a phenomenon in which a large language model (LLM), often a generative AI chatbot, creates outputs that are nonsensical or altogether inaccurate by detecting patterns or objects that are nonexistent or cannot be perceived by human observers ([I2]).

Artificial General Intelligence (AGI), also known as 'strong AI', refers to a hypothetical technology of artificial intelligence that possesses human-level cognitive abilities. Unlike narrow AI, which is designed to perform specific tasks (such as language translation, image recognition or playing chess), AGI aims to replicate the wide range of cognitive functions that humans are capable of. While AGI is still in the theoretical realm, organisations can take proactive steps to prepare for its arrival by building a robust data infrastructure and fostering a collaborative environment in which humans and AI can work together seamlessly.

Chapter 6 provides a more detailed overview of the expected future development of AI.

3. The Foundations of Artificial Intelligence

By far, the greatest danger of Artificial Intelligence is that people conclude too early that they understand it.
– Eliezer Yudkowsky ([11])

Understanding the structure and mechanisms of artificial intelligence requires examining both its functional domains and the algorithms that bring them to life. This chapter presents the principal foundations of modern artificial intelligence. It begins by outlining the core domains of AI and their primary applications, and then surveys the most important algorithmic families, illustrating them with concrete examples.

Artificial intelligence is not a single technology but a multifaceted field that integrates methods and insights from numerous disciplines. Its capabilities emerge from the interplay of diverse components that, together, approximate selected aspects of human cognition and behaviour.

3.1. Core domains and applications of AI

Artificial intelligence encompasses a wide variety of methods and technologies designed to enable machines to perceive, learn, reason and act intelligently ([M3]). Machine learning and deep learning form the basis for recognising patterns in data, while natural language processing and computer vision build on this to enable understanding of language and visual information. Meanwhile, robotics, autonomous systems, and agentic AI translate this intelligence into action. Expert systems, recommender systems and generative AI then demonstrate how knowledge and creativity can be applied to solve real-world problems.

Together, these fields constitute the core landscape of contemporary AI research and applications.

3.1.1. Machine Learning (ML)

Machine Learning is a subset of AI that focuses on the development of algorithms that facilitate the acquisition of knowledge by computers and the subsequent capacity to draw conclusions or formulate decisions based on the analysis of data. This encompasses a range of methodologies, including supervised learning, unsupervised learning, and reinforcement learning. ML is the most important part of artificial intelligence. It enables AI systems to learn from diverse datasets without the need for explicit programming. Machine learning uses statistical techniques to improve continuously over time.

ML has a broad range of applications. These include image and speech recognition and predictive analytics. Furthermore, it can be used in conjunction with other components, such as natural language processing and computer vision. Notable examples of machine learning use cases include recommendation engines, autonomous cars and search algorithms.

One of the key strengths of human intelligence is adaptability. We can adapt to different environmental conditions by learning and adjusting our behaviour accordingly. This is precisely why machine learning is also central to AI, even though our learning ability is far superior to that of computers. ([E1])

3.1.2. Deep Learning (DL)

The advent of deep learning has profoundly impacted the realm of machine intelligence, facilitating the acquisition, understanding, and interaction with complex data. Deep learning is a

[11] Eliezer Yudkowsky (1979–) is an American artificial intelligence researcher and writer, known for his work on decision theory, ethics, and AI safety.

subset of machine learning that emulates the complex structural characteristics of the human brain. It relies on the use of neural networks with many layers (hence 'deep') to autonomously identify patterns and make informed decisions from extensive amounts of unstructured data. It has revolutionized many fields by enabling significant advancements in tasks such as image recognition, natural language processing, and game playing. The integration of deep learning into the AI landscape is crucial, as it serves to augment the capabilities of conventional machine learning methodologies. The importance of deep learning is underscored by its ability to mitigate the necessity for manual intervention, thereby enhancing the efficiency and autonomy of AI systems.

Deep learning has transformed AI by enabling systems to learn and perform tasks that were previously difficult or impossible for machines. The capacity for automated feature extraction from raw data has resulted in its recognition as a powerful tool across various industries. Deep learning continues to drive innovation in AI assistants, healthcare, finance, robotics, autonomous vehicles, and scientific research. Breakthroughs in self-supervised learning, multimodal AI ([12]), and efficient architectures will make it even more powerful. ([13])

3.1.3. Natural Language Processing (NLP)

Natural language processing is a subfield of computer science and artificial intelligence which combines computational linguistics, statistical modelling, and machine learning to enable computers to interpret, understand, and generate human language. It is a rapidly evolving field with diverse applications that enhance human–computer interaction and improve the accessibility and usability of digital information (for more detailed information on NLP, readers are referred to Chapter 5).

NLP includes tasks such as

- **Text processing**. This involves processes like tokenisation (the process of breaking down text into words or sentences), stemming and lemmatization (the process of reducing words to their root forms), and part-of-speech tagging (the identification of grammatical categories of words).

- **Syntax and parsing**. NLP systems analyse the grammatical structure of sentences in order to comprehend the relationships between words and phrases. The process of parsing involves the breakdown of a sentence into its constituent components so that its structure can be understood.

- **Semantic analysis**. The focus of this aspect is on the understanding of the meaning of words, phrases and sentences. The techniques employed include word embeddings (numeric vector representations of words) and named entity recognition (identification of proper names and specific entities).

- **Sentiment analysis**. NLP can be utilised to ascertain the sentiment or emotion conveyed in a given text. This capability is useful for applications such as customer feedback analysis, social media monitoring, and market research.

- **Machine translation**. NLP systems have the capacity to translate text from one language to another. Examples of such systems include Google Translate and DeepL.

- **Speech recognition**. The process of converting spoken language into written form is known as transcription. This technology finds application in virtual assistants such as Siri and Alexa, as well as transcription services.

---

[12] Multimodal AI refers to machine learning models that can understand and process different types of information, such as text, images, audio and video, simultaneously. These models combine modality-specific encoders — such as language transformers, vision backbones and audio networks — to create a joint representation that can be used by downstream components for tasks such as captioning, retrieval or synthesis. ([M19], [S18]).

[13] The topic of machine and deep learning is covered in more detail in Chapter 4.

- **Text generation**. NLP models have the capacity to generate human-like text based on given prompts. This capacity is utilised in a variety of applications, including chatbots, content creation and language translation.

- **Dialogue systems**. NLP provides the capacity for conversational agents and chatbots to engage in human-like conversations, with these systems being utilised in a variety of contexts, including customer service, virtual assistants and interactive applications.

3.1.4. Computer vision

This field of AI is concerned with the development of algorithms and models that facilitate machines in understanding visual data, much in the manner that humans do. This domain is undergoing rapid advancements, propelled by the increasing capabilities of machine learning and deep learning algorithms, as well as the availability of substantial datasets. The applications of Computer Vision are expanding across various industries, enhancing automation and improving decision-making processes. Examples of computer vision in AI include the autopilot system of Tesla and automatic photo tagging on social media.

The following are the key concepts in computer vision:

- **Image recognition**. The ability of a computer to identify objects, people, places and other entities in images. Examples include the recognition of faces in photographs and the identification of products in an online store.

- **Object detection**. Identifying and locating objects within an image or video. This technique is employed in a variety of applications, including autonomous vehicles, where the system must detect and avoid obstacles.

- **Image segmentation**. Image segmentation is defined as the division of an image into multiple segments or regions for the purpose of simplifying its analysis. This process is frequently employed in the field of medical imaging, with the objective of facilitating the identification of different tissues or organs.

- **Feature extraction**. This involves identifying and extracting important features from an image, such as edges, textures and shapes. These features can then be used for further analysis and recognition tasks.

- **Facial recognition**. The ability to recognise and verify human faces in images or videos. This technology is widely used in security systems, social media, and biometric authentication.

- **Image generation**. The process of creating new images based on given inputs or generating realistic images from textual descriptions. *Generative Adversarial Networks* (GANs) are a popular technique employed in this domain (see Section 4.3.7).

- **Motion and tracking**. The analysis of movement of objects within a video to track their positions over time is known as motion and tracking. This technology finds application in fields such as video surveillance and sports analysis.

- **3D Vision**. The process of constructing three-dimensional models of objects or scenes from two-dimensional images is of particular importance in a variety of applications, including robotics, virtual reality, and augmented reality.

Although computer vision systems have made remarkable progress, they still face fundamental challenges that affect their accuracy, reliability and ability to be deployed in the real world. Here are some examples:

× **Sensitivity to visual variations**. Models often malfunction when subjected to changes in lighting, weather conditions, viewpoint, scale or obstruction. For example, a pedestrian detector that works at noon may fail at dusk, or when someone is partially hidden behind a sign. This fragility forces teams to either collect vast amounts of data covering every possible scenario or engineer complex data augmentation pipelines.

× **Data annotation and labelling bottleneck**. High-quality annotations, such as bounding boxes, segmentation masks and keypoints, take hours per image when carried out by experts. Domains such as medical imaging and autonomous driving require pixel-perfect labels from specialists, which increases costs and slows down the iteration process. Although synthetic data and weak supervision help, bridging the gap to real-world performance remains both an art and a science.

× **Domain shift and generalisation**. A model trained on dashcam recordings in sunny Rome will struggle on rainy streets in London. Differences in sensor types, geographical features and cultural artefacts all contribute to domain shift.

× **Explainability and interpretability**. Computer vision models often operate as 'black boxes': they can detect a tumour, for example, but cannot explain which features triggered the decision. This lack of transparency undermines trust in contexts such as healthcare, finance and law enforcement, where stakeholders demand clear rationales.

3.1.5. Robotics

The field of AI that focuses on the design, construction, operation, and use of robots. These robots are typically machines that can perform tasks autonomously or semi-autonomously, often replicating or enhancing human capabilities.

The field of robotics is undergoing rapid evolution, driven by advancements in AI, machine learning, and sensor technologies. These developments are projected to yield new capabilities, including the emergence of more sophisticated autonomous systems, enhanced human-robot collaboration, and robots capable of operating in complex, unstructured environments. The potential impact of robotics on various industries is significant, with the capability to transform existing operations by performing tasks that are dangerous, repetitive, or require high precision.

AI-driven robotic systems, for example, are transforming modern armed forces by enabling unmanned platforms, autonomous weapons systems, robotic security guards, logistics support and human-machine collaboration in all operational areas.

However, AI-driven robots face a unique set of challenges spanning perception, reasoning, control, hardware integration and human interaction. Unlike purely digital AI systems, robots operate in the physical world, where limitations are amplified by uncertainty, safety requirements and real-time constraints. Examples of such limitations include ([M22], [W7]):

× **Lack of common sense and contextual reasoning**. Robotic AI systems rely on predefined data and algorithms and lack the intuitive 'common sense' that humans use to interpret ambiguous situations. This limitation is evident when a robot misinterprets an instruction because it cannot recognise implicit context.

× **Perception and real-time control**. Robots must perceive their environment, process sensory inputs and act, all within milliseconds. Delayed or incorrect reactions can be caused by noise, latency, and calibration drift in sensors. In high-stakes settings such as autonomous vehicles or surgical robots, even minor timing errors can result in failures or safety hazards.

× **Human-robot interaction**. In areas like elder care and customer service, robots must be able to interpret human emotions, intentions, and social norms. However, current AI lacks genuine empathy and often misinterprets non-verbal cues, resulting in awkward and potentially unsafe interactions. Although incorporating basic emotional models and context-aware dialogue systems offers a partial solution, achieving fluid, trustworthy collaboration with humans remains a significant challenge.

× **Data scarcity and annotation bottlenecks**. Collecting and labelling the vast and diverse amounts of data required for robots – covering every possible object pose, surface texture and interaction scenario – is prohibitively expensive. Although self-supervised learning and simulation can reduce the burden of annotation, manual oversight is still required for real-world validation, particularly in safety-critical applications such as medical or industrial robotics.

3.1.6. Expert systems

Expert systems are AI systems that emulate the decision-making ability of a human expert in a specific domain. They are used in various fields such as medical diagnosis, financial analysis, and engineering. Here are the key aspects of expert systems:

- **Knowledge base.** This is the core of the expert system, incorporating domain-specific knowledge acquired from human experts. It consists of facts, rules, and heuristics that guide the system's decision-making process.

- **Inference engine**. This component processes information contained within the knowledge base. The purpose of this process is to draw conclusions or make decisions. In order to do this, the Inference Engine applies logical rules to the knowledge base. The application of these rules results in the inference of new information and the solving of problems.

- **User interface**. This facilitates interaction with the expert system. Users can submit inquiries or challenges, and the system responds with explanations, recommendations, or solutions derived from its knowledge base.

- **Explanation facility**. This feature provides users with an explanation of how the expert system arrived at its conclusions. This feature serves to enhance the transparency and trustworthiness of the system.

The following challenges and limitations of expert systems must be noted ([S22]):

× **Knowledge acquisition and maintenance**. Expert systems require expert knowledge to be manually encoded into rules or taxonomies, which creates a bottleneck in system development and risks knowledge gaps or inconsistencies. Obtaining rules from domain experts is time-consuming and error-prone. Maintenance is also an issue, as any evolution in domain knowledge demands laborious manual updates.

× **Inability to learn and adapt**. Unlike machine learning models, which refine patterns from data, expert systems remain static unless they are manually revised. This limits their adaptability in dynamic environments. Consequently, they may struggle to generalise beyond pre-coded logic when faced with novel problems.

× **Scalability and domain restriction**. Expert systems are very good at dealing with specific areas of a subject, but when they are used in more general ways, the number of rules can quickly become too many, which makes them difficult to manage. Entering adjacent fields often necessitates a distinct or re-engineered rule base.

× **High development cost and time**. Building a robust expert system requires extensive knowledge capture, rule validation and iterative testing, which results in a significant initial investment. This resource-intensive development process requires considerable time and expertise for each phase.

These limitations explain why modern AI has shifted towards hybrid, learning-based architectures that combine the interpretability of expert systems with the adaptability of data-driven systems.

3.1.7. Recommender systems

Recommender systems represent a significant and extensively utilised domain of AI, with the objective of providing customised recommendations to users in accordance with their preferences, behaviour, and associated data. These systems play a pivotal role across a wide range of applications, including e-commerce and streaming services. Recommender systems are an integral component of numerous contemporary applications, enhancing user experiences and driving business growth through the provision of personalised suggestions.

The following aspects of recommender systems are of particular significance:

- **Collaborative filtering**. This approach is based on recommendations made according to the preferences of like-minded users. The system analyses the behaviour of another user who has demonstrated similar behaviour (for example, comparable purchases and ratings) to determine recommendations for a given user.

- **Item-based collaborative filtering**. Focuses on finding items that are similar to those a user has liked and recommending them.

- **Content-based filtering**. This approach makes recommendations based on the characteristics of items and the user's profile. It analyses the content (e.g., keywords, attributes) of items that a user has shown interest in and recommends related items.

- **Hybrid methods**. These combine collaborative filtering and content-based filtering to leverage the strengths of both approaches. Hybrid systems can provide more accurate and diverse recommendations.

- **Matrix factorisation**. A technique used to identify latent factors (hidden features) that influence user preferences. It decomposes the user-item interaction matrix into lower-dimensional matrices, which can be used to make recommendations (see Section 3.2.9 for more details).

- **Deep learning**. Advanced models, such as neural networks, are used to capture complex patterns and relationships in data. Deep learning-based recommender systems can handle large-scale data and provide personalised recommendations.

The limitations and challenges related to recommender systems include the following ([C7], [M23]):

✕ **Cold start problem**. Recommender systems may encounter difficulties in producing accurate recommendations for new users or items with limited or no historical data. Systems often use content features or demographic data to build profiles. Moreover, the user-item matrix is usually very sparse ([14]), which reduces the effectiveness of collaborative filtering and matrix factorisation methods.

✕ **Algorithmic and model limitations**. Models tend to over-recommend popular or 'blockbuster' items, thereby reinforcing popularity loops and starving the long tail of exposure. Achieving a balance between prediction accuracy and diversity, novelty and surprise remains an open research problem. Furthermore, static models overlook changes in user preferences over time and contextual signals, such as the time of day, location or device type. Capturing these dynamics often requires session-based architectures, time-aware factorisation techniques, or complex hybrid models.

✕ **Ethical, privacy and trust issues**. Privacy concerns arise from the collection and processing of granular behavioural data, which triggers regulatory scrutiny under the EU's General Data Protection Regulation (GDPR), the California Consumer Privacy Act (CCPA), and other legal frameworks. Training data can also encode demographic or cultural biases, which can result in the unfair treatment or misrepresentation of certain user groups or item categories..

3.1.8. Generative AI (GAI)

We have all witnessed OpenAI's reshaping of the digital landscape with the introduction of tools such as ChatGPT. Powered by generative AI, ChatGPT has had a significant impact on the consumer market and is being adopted by many businesses to address challenges that were previously considered intractable.

Generative artificial intelligence (GAI), also known as GenAI, is a branch of artificial intelligence that uses machine learning models – often large, pretrained neural networks – to identify patterns and structures in vast datasets, and then generate new content resembling human output. This content can include text, images, audio, video or code, and is produced in

[14] A sparse matrix is a matrix that contains a large number of zero-valued elements.

response to user prompts ([15]) or other inputs. No human intervention or influence is required for this process.

Generative AI is one of the most transformative and high-impact subfields of artificial intelligence. It is changing the way we create, communicate and solve problems in a variety of industries. It sits alongside other crucial AI domains, such as reinforcement learning, symbolic reasoning and robotics. As these areas converge, the next wave of breakthroughs will stem from combining the strengths of different subfields.

Gartner predicts that generative AI will become a general-purpose technology, comparable in impact to the steam engine, electricity, and the internet. While the initial hype will decline as the complexities of implementation become apparent, the long-term impact of generative AI is expected to grow as individuals and organisations discover increasingly innovative applications for the technology in their daily lives and work ([G17]).

Generative AI can be broken down into three interconnected layers: core algorithms (models), infrastructure and tooling (platforms), and end-user solutions (applications), which are built on top of the first two layers. Here's how these layers work together: a research team publishes a new generative model. A cloud vendor then integrates the model into its platform by wrapping it with APIs ([16]). Finally, a product team builds an application – for example, an AI-powered design tool – by calling the platform's APIs and embedding the output in a user-friendly interface.

3.1.8.1. Generative AI models

Generative AI models are mathematical and statistical algorithms that analyse data distributions to generate new content. Examples include:

- **Claude 3.5 Haiku & Sonnet**. These are two new AI models released by Anthropic ([17]) as part of the Claude 3.5 family. Haiku is designed for speed and affordability, making it ideal for fast, user-facing applications. Sonnet, on the other hand, is built for higher-level reasoning and complex tasks, offering enhanced performance in areas such as coding and problem solving ([A12]).

- **DALL-E.** DALL-E is a series of text-to-image models developed by OpenAI that use deep learning to generate digital images from natural language descriptions. The first version was released in 2021 and uses a transformer-based architecture, originally derived from GPT-3, to produce novel, coherent images from text. The name DALL-E is a combination of the name of the Pixar animated robot character WALL-E and the surname of the surrealist artist Salvador Dalí, reflecting the model's focus on imaginative image creation through text-driven artwork ([W1]).

- **DeepSeek R1 1776**. DeepSeek R1 1776 is an open-weight ([18]) variant of the DeepSeek-R1 model. It has been post-trained by Perplexity AI to bypass Chinese Communist Party censorship constraints while retaining its original reasoning capabilities. As a generative large language model built on the transformer decoder paradigm, similar to GPT or Llama, its core strength lies

---

[15] A prompt is the input or instruction that tells a generative AI model how to produce its output. Think of it as the question, context or example that you provide to the system, to which the model then responds. Text prompts, for example, are natural-language instructions or questions for large language models (e.g. 'Summarise this article' or 'Translate to German'). Visual prompts are images or sketches fed into multimodal models with instructions such as 'Describe this scene'.
Prompt engineering means formulating effective instructions. It is the art and science of designing prompts to produce reliable, high-quality results (see Chapter 5).

[16] An AI API (artificial intelligence application programming interface) is a technology that enables developers to incorporate AI features into applications. These APIs connect AI functionalities with various applications, allowing them to perform complex tasks using AI models.

[17] Anthropic PBC is an American artificial intelligence start-up based in San Francisco, California. Founded in 2021 by a team of former OpenAI researchers, including siblings Daniela and Dario Amodei, it is one of the most prominent players in the field of AI ([W1]).

[18] 'Open-weight' refers to large-scale AI models whose trained neural network weights are released to the public under a permissive licence. However, the accompanying training code or datasets may not be fully open-source.

in producing new text (or structured outputs) in response to prompts. The model provides unbiased, accurate and factual responses on a wide range of sensitive topics, and is released under the MIT licence ([19]) for unrestricted use ([D7]).

- **GPT-4**. *Generative Pretrained Transformer* -4 (GPT-4) is the fourth generation large multimodal language model. It was developed by OpenAI. Unlike earlier versions, such as GPT-2 and GPT-3, GPT-4 can interpret and process multiple types of input, including text, audio and images. This flexibility enables users to input various forms of data, to which GPT-4 can respond by generating detailed written content, in-depth explanations, computer code, and original compositions. Its outputs are designed to closely mimic human thought and language, making it effective for a wide range of applications ([M14]).

- **GPT-4 Omni**. GPT-4 Omni (branded as GPT-4o, where 'o' stands for 'omni') is OpenAI's first end-to-end multimodal model, natively processing text, images, audio and video through a single neural network. Debuting on 13 May 2024, it powers Voice Mode in ChatGPT, as well as the API, enabling conversational interactions with near-human speed and expressiveness. The model can handle any combination of text, audio, image, and video inputs and generate text, audio, or image outputs ([O1]).

- **GPT-5.2**. GPT-5.2 is an enhanced version of the GPT-5 family. It offers improved reasoning depth, long-context stability and more reliable multi-step problem solving. Released on 11 December 2025, it builds upon the decoder-only Transformer architecture with optimisations that enhance coherence in extended conversations and minimise error accumulation in intricate tasks. Although GPT-5.2 is primarily a text-centric model, it integrates seamlessly with multimodal systems and tool-use frameworks, enabling richer workflows that combine text generation with external audio, image or search components (see [O4], [O5], and Section 5.2.5.3).

- **Jukebox**. This experimental AI model, developed by OpenAI, generates music in raw audio form, including basic singing, across various genres and artist styles. Trained on a dataset of over 1.2 million songs, it can create original musical compositions from scratch or complete existing ones ([J4]).

- **Llama 3.1 Sonar Large**. Sonar Large is an advanced AI language model based on Meta's Llama 3.1 405 B-parameter architecture, designed for large-scale enterprise applications. With an unprecedented 128,000-token context window ([20]), it enables the comprehensive analysis and generation of long-form content in a single pass ([L6]).

- **MusicLM**. Developed by Google Research, MusicLM is a generative text-to-music model that produces rich, high-fidelity audio from natural language prompts. It can transform text into contextually appropriate music and generate both single tracks and continuous playlists ([M13]).

- **PaLM 2 & Gemini**. Both PaLM 2 and Gemini are large language models (LLMs) developed by Google. PaLM 2 is the successor to Google's previous LLM, the Pathways Language Model (PaLM), and has been designed to excel at multilingual tasks, reasoning, coding and mathematics. Gemini, which was developed by Google DeepMind, is a more advanced model that is positioned as a successor to PaLM 2. It has enhanced multimodal capabilities, meaning that it can process and understand different types of data, such as text, images, audio, video and code ([M15]).

---

[19] The MIT licence is a permissive open-source software licence that grants users broad rights to use, copy, modify and distribute software for any purpose, including commercial use. However, they must include the original copyright and licence notice in any redistribution.

[20] A token context window is the maximum length of text that a generative model can 'see' and process at once, measured in tokens. It determines how much of the prior conversation, document or code the model can consider when predicting the next token. Tokens are the basic units of text that the model uses; these are often subwords or punctuation. For instance, the word 'unbelievable' might be split into 'un', 'believ', and 'able'. See Chapter 5 for more details.

3.1.8.2. Generative AI platforms

Generative AI platforms provide the APIs, development tools, hosting, monitoring and governance ([21]) required for training, deploying and managing generative models on a large scale. They can be thought of as cloud-based model factories. Examples of AI generative platforms include ([M12]):

- **Amazon Bedrock**. It is a managed service that enables development of generative AI applications and agents on the Amazon Web Services (AWS) cloud computing platform ([H11]). It offers access to foundation models (FMs), including AWS Titan models, as well as from partner providers such as AI21 Labs, Anthropic, Cohere, and Stability AI. FMs are adaptable AI models that have been trained on large data sets to perform a variety of tasks. Bedrock replaces the physical infrastructure typically used to build generative AI apps with FMs, thereby simplifying the app-building process. The pricing model offers pay-per-request and provisioned throughput options ([22]). Bedrock competes with Google Cloud Vertex AI and Microsoft Azure OpenAI Service ([K11]).

- **Google Cloud Vertex AI**. It is an end-to-end, managed machine learning platform that brings together Google's AI services. It enables developers to build and deploy predictive and generative AI models using a single console, API and set of tools. Vertex AI provides access to over 200 foundation models, including Google's Gemini multimodal models, Anthropic's Claude and open-source variants. Vertex AI Studio is a visual integrated development environment (IDE) for prompt engineering, fine-tuning, and testing generative models. Agent Builder provides a no-code environment for assembling, grounding, and orchestrating AI agents ([23]) that interact with your data and services. Vertex AI offers usage-based pricing for API calls, model training and pipeline executions.

- **Microsoft Azure OpenAI Service**. This managed cloud-based service provides access to OpenAI's AI models, including the latest GPT-series models, DALL-E, GitHub Copilot, Codex and Whisper, for text, image, code and speech tasks within the Azure platform ([24]). The service enables developers to build and deploy AI applications, such as chatbots and content creation tools, using these models. The price model is based on token billing, with separate input and output rates.

3.1.8.3. Generative AI applications

Generative AI applications are end-user products or services that incorporate generative capabilities in order to solve specific problems. Examples range from chatbots and content creation suites to code assistants and design generators. AI generative applications built on pretrained generative models include:

---

[21] Governance refers to the technical and procedural controls that ensure generative AI systems are operated responsibly. These include model versioning, access management, tracking data and model lineage, auditing performance, and ensuring regulatory compliance throughout the lifecycle. Tracking data and model lineage involves recording the origin, transformations and dependencies of datasets and models throughout their lifecycle. This enables reproducibility, auditability, and accountability in AI development and deployment.

[22] Throughput refers to the number and rate of inputs and outputs that a model processes and returns ([A11]).

[23] AI agents are autonomous, decision-making systems that perceive their environment and execute actions to achieve specified objectives without continuous human intervention. Common use cases include virtual assistants that schedule meetings, draft emails or summarise documents; autonomous customer service bots; workflow orchestrators; and research assistants that retrieve literature, analyse findings and draft reports.

[24] Microsoft has been OpenAI's primary financial and infrastructure partner since 2019. As part of their strategic agreement, Microsoft has invested over $13 billion in OpenAI, securing the exclusive right to host OpenAI's API on its Azure cloud service, as well as the right to use OpenAI's intellectual property in Microsoft products such as GitHub Copilot. The two companies also share revenue from model usage, and Microsoft continues to provide most of the computing power that drives ChatGPT and other OpenAI services. OpenAI's API exclusivity on Azure runs until 2030, after which the terms may be renegotiated. ([M24])

- **ChatGPT** is a general-purpose application based on OpenAI's GPT models (large language models, or LLMs) ([25]), which are prime examples of generative AI in the field of natural language processing (NLP). Designed to understand and respond to a wide range of prompts, it can answer questions, write stories, translate languages, summarise texts, generate code and much more, all via generative NLP ([26]). ChatGPT can be accessed via a browser, a mobile app or an API. It operates in a neutral environment –simply paste or type in the text you want it to work on. It can be tailored using custom GPTs and plugins.

- **Claude AI.** Claude AI, also known as Claude, is a generative AI chatbot that uses large language models (LLMs). It was developed by the research firm Anthropic. Designed for natural language processing (NLP), Claude is multimodal and can accept text, audio, and visual inputs. It is also capable of answering questions, summarising documents, and generating long-form text, diagrams, animations, program code, and more ([B15]).

- **Copilot.** Microsoft Copilot is an AI-powered assistant. It operates as a conversational chat interface and offers assistance with tasks such as content creation, information summarisation, image generation and code writing in various programming languages, including C, JavaScript and Python. Copilot is integrated with Microsoft 365 apps. It is powered by a combination of Microsoft technologies and advanced large language models, including those from OpenAI (see Section 5.2.5.5 for more details).

- **GitHub Copilot.** GitHub Copilot is the product of a collaboration between GitHub, a Microsoft subsidiary, and OpenAI. This AI-powered coding assistant integrates directly into software development environments such as MS Visual Studio. It uses OpenAI's Codex model, which has been trained on billions of lines of public code, to provide AI-driven code suggestions. Furthermore, GitHub Copilot supports Anthropic's Claude models as optional engines within Copilot Chat. Claude Sonnet 4 and Claude Opus 4 from Anthropic became generally available in GitHub Copilot Chat on 25 June 2025. Sonnet 4 is accessible to all paid Copilot subscribers. Opus 4 is reserved for Pro+ and Enterprise plans. Both models appear in the Copilot Chat model selector on github.com and in the following IDEs: VS Code, Visual Studio, JetBrains, Xcode and Eclipse. They are also available on GitHub Mobile. ([A17], [B14]).

- **Google Bard**. It is an AI chatbot that is part of the Gemini family of AI models. It is designed to be conversational and uses natural language processing to provide human-like responses to user queries, summarise information and generate creative text. Essentially, it's Google's answer to other AI chatbots like ChatGPT, but with its own unique features and capabilities. Google Bard operates with a language model PaLM-2.

- **Grok.** It is a generative AI chatbot developed by xAI, an artificial intelligence company founded by Elon Musk. Launched in 2023, it uses an LLM to answer questions, generate creative text and support various real-time tasks. The name 'Grok' comes from Robert A. Heinlein's science fiction novel *Stranger in a Strange Land*, where it means 'to understand completely' or 'to grasp a subject fully and intuitively'. Grok is integrated with X (formerly Twitter) via a web interface. There are also native mobile apps for iOS and Android. Furthermore, it is built into Tesla vehicles via Tesla OS, enabling hands-free interaction while driving. Unlike some more formal AI chatbots, Grok's personality is designed to be humorous and a bit rebellious. It takes a more relaxed approach to politically sensitive topics than other chatbots. Despite this, it still enforces content policies to block disallowed content. Its 'relaxed' approach does not mean unfiltered or unchecked output, but rather a wider stylistic range.

---

[25] OpenAI does not publicly disclose which exact GPT model powers ChatGPT at any given moment; the system may use different models (e.g., GPT-4, GPT-4o, GPT-4.1, GPT-5, GPT-5.2) depending on tier, feature, and deployment stage.

[26] Although generative AI and natural language processing (NLP) are closely connected, they represent different layers in the AI ecosystem. NLP is the overarching discipline that enables computers to understand, interpret and work with human language. It encompasses a variety of techniques and tasks, some of which are generative and others discriminative, but it is not a single 'generative AI model' itself.

- **Perplexity**. Founded in 2022, Perplexity grew rapidly to reach millions of regular users by early 2024 ([27]). It has positioned itself as a next-generation alternative to traditional search engines and standalone chatbots. Instead of traditional result lists, Perplexity provides well-sourced answers. It uses multiple LLMs (such as OpenAI's GPT series and Anthropic's Claude) and its own models (Sonar and R1 1776) to optimise response accuracy and coverage. It enables summarisation, content exploration and assistance through a natural-language interface ([U5]). The free version of Perplexity is primarily powered by GPT-3.5 and has a browsing extension for gathering real-time information. The Pro version offers access to advanced models such as GPT-4o, Claude 3.5 Sonnet and Haiku, Sonar Large, Grok-2, Gemini 2.0 Flash/2.5 Pro and DeepSeek R1 1776, providing larger context windows and multimodal support ([S14]).

3.1.8.4. Real-world applications of generative AI

Generative artificial intelligence has a wide range of applications in various industries, including media and entertainment, healthcare, manufacturing, software development, financial services, and advertising and marketing. Some examples are given below ([S12]):

- **Financial services**. Generative AI is transforming the financial services industry by automating routine tasks, extracting insights from unstructured data and providing personalised services on a large scale. Applications of generative AI include using it to develop investment strategies, draft documentation, monitor regulatory changes and facilitate communication between clients and investors by acting as an interpreter.

GAI can accelerate regulatory review, code modernisation and compliance reporting, for example. It can translate and summarise new regulations (e.g. Basel III) into actionable developer tasks. It automates the cross-checking of code repositories against regulatory requirements. Developers use generative agents to identify the code changes required for evolving banking regulations, thereby reducing the need for manual review ([B16]).

- **Office equipment and supplies**. Generative AI can enhance many aspects of office equipment design, sales, servicing and support. These technologies can drive efficiency and innovation across every function, from speeding up design cycles to transforming customer interactions. The key application areas and concrete examples are outlined below.

  - *Product design and engineering*. Generative AI can accelerate the creation and optimisation of new products by exploring design variations, reducing material usage and enhancing performance.
  - *Predictive maintenance and field service*. Generative AI can analyse sensor data and historical repair logs to forecast equipment failures, propose maintenance steps and automate technician support. It can generate customised maintenance schedules for each device based on usage patterns and environmental conditions. Furthermore, it can produce step-by-step repair guides complete with annotated images or diagrams for field technicians.
  - *Supply chain optimisation and inventory planning*. GAI can simulate demand fluctuations, propose stocking strategies and predict the impact of supply-chain disruptions. GAI can produce multiple demand-forecast scenarios based on seasonality, promotional plans and macroeconomic indicators. Furthermore, GAI can generate optimised reorder points and order quantities for parts and consumables, balancing stock-out risk with holding costs.

- **Software development**. Generative AI is revolutionizing how software is built by automating routine tasks, accelerating workflows, and improving quality at every stage of the development cycle ([F4], [G16], [P3]). For software development teams, generative AI can provide tools that enable faster and more efficient code creation and optimisation. Some examples of generative AI applications in software development include:

---

[27] Perplexity AI, Inc. is a privately held American software company headquartered in San Francisco, California. It was founded by Aravind Srinivas, Denis Yarats, Johnny Ho and Andy Konwinski.

- *Requirements gathering and analysis*. GAI assists in converting high-level concepts or stakeholder discussions into specific user stories. GAI can perform semantic searches across documentation, tickets and past project artefacts to extract relevant requirements. Furthermore, it can summarise extensive text (e.g. RFCs and specification documents) into succinct feature lists.
- *Design and architecture*. GAI can generate system and component diagrams (UML ([28]) and data flow diagrams) from natural-language descriptions. It can also propose optimal software architectures or infrastructure topologies based on performance and cost constraints.
- *Automated code generation and completion*. Software developers can use generative AI to create, optimise and auto-complete code. Generative AI can suggest inline code snippets, entire functions or CRUD ([29]) templates by understanding the context of the project. GAI provides support for rapid prototyping across languages and frameworks. Real-world tools include: GitHub Copilot, Amazon CodeWhisperer and TabNine, which facilitate on-the-fly code authoring.
- *Code review and quality assurance*. Generative AI can analyse pull requests to identify bugs, security vulnerabilities and style violations. It can also offer targeted refactoring suggestions and detect anti-patterns – coding or design practices that may appear effective in the short term but ultimately diminish code quality, maintainability, or performance – before merging.
- *Testing automation*. Generative AI can learn the logic of the software and predict how users will interact with it. It can then create test cases to demonstrate various user scenarios. It can generate unit, integration and end-to-end test cases that cover edge conditions.
- *Generation of documentation and knowledge bases*. Generative AI can create API references, inline comments, README files and user guides from code. It can also translate and localise technical documentation into multiple languages while maintaining formatting and accuracy. GAI can summarise code changes automatically to keep manuals up to date.
- *Project management and collaboration*. Generative AI can automate the creation of tasks, backlog grooming and sprint planning using historical velocity data. It can also provide time-estimate predictions and recommendations for resource allocation, as well as serving as an assistant during stand-up meetings by summarising progress and blockers.
- *Maintenance and continuous improvement*. GAI helps forecast code hotspots that are prone to bugs or technical debt, and suggests refactoring schedules. It can automatically and continuously generate patch-level updates and migration scripts, as well as monitoring production logs to predict incidents and draft mitigation steps.

- **Healthcare and pharmaceuticals**. Generative AI is already transforming how therapies are discovered, tested and delivered, promising faster innovation and lower costs. It has applications in every area of the healthcare and pharmaceutical industry, including discovering and developing new medicines, personalising treatment plans for patients, and creating predictive images to chart disease progression. Some applications of generative AI in healthcare include ([B17], [H10], [R10], [S13] and [S15]):
  - *Research and drug discovery*. Researchers can use generative artificial intelligence, via a related field called generative design, to develop new medicines. Gartner projects that, by

[28] The Unified Modeling Language (UML) is a standardised visual modelling language used to specify, visualise, construct and document software systems.

[29] CRUD is an acronym that stands for 'Create, Read, Update and Delete'. These are the four basic operations involved in managing data in most applications and databases.

2025, 30 % of new drugs created by researchers will use generative design principles ([G17]). AI models can analyse biological datasets (e.g. genomics and proteomics) to identify disease targets and validate mechanisms in silico ([30]). They can also propose novel compounds and optimise their chemical structures for potency and safety. McKinsey estimates that generative AI could unlock \$15–28 billion ($1\ billion = 10^9$) annually by accelerating lead discovery and preclinical validation ([S13]).

- *Enhancing medical images*. Generative AI can augment medical images such as X-rays and MRIs, synthesise images, reconstruct images and generate reports on them. It can even generate new images to show how a disease may progress over time. Diffusion and GAN-based models can generate synthetic MRI, CT or histopathology slides to enhance limited datasets and train diagnostic classifiers. AI can transform one scan type into another (e.g. low-dose CT to high-resolution), thereby reducing radiation doses. Tools such as Paige.AI use generative techniques to highlight suspicious tissue regions, thereby improving the detection of prostate and breast cancers. Early studies demonstrate generative AI's ability to match radiologist accuracy while reducing image annotation time by over 50 % ([H10], [R10]).
- *Clinical development & trials*. Models can generate virtual patient cohorts that can serve as control groups, thereby reducing the need for placebo trials and speeding up the recruitment process. AI can simulate trial designs, including sample sizes, endpoints, and dosing regimens, in order to maximise statistical power and minimise costs. Generative models forecast safety risks by synthesising past trial data and real-world evidence. Digital trial modelling and synthetic data techniques have been validated in oncology and rare disease studies, reducing study timelines by months and cutting per-patient costs by up to 30 % ([H10]).
- *Personalised treatment*. Generative AI can analyse large amounts of patient information, including medical images and genetic testing results, in order to create personalised treatment plans. It can also simulate pharmacokinetic and pharmacodynamic responses to suggest real-time dose adjustments.
- **Defence and military**. Generative AI is being increasingly explored for defence purposes, primarily as a tool for simulation, decision support, and operational planning. It can generate realistic virtual environments and adversary behaviours for war gaming and mission rehearsal, thereby reducing the need for costly live training. In intelligence analysis, generative models synthesise information from multiple sources, summarise large datasets, and propose potential courses of action. Furthermore, in the context of autonomous systems, generative AI contributes to the coordination and adaptive behaviour of drone swarms and robotic units. While these applications remain under strict human oversight, they demonstrate how generative AI can enhance situational awareness, planning, and decision-making in complex, data-rich military environments ([G24]). However, many of these applications raise ethical, legal, and accountability concerns, such as autonomy in lethal systems and bias in intelligence generation ([I4]).

For more information on generative AI, please refer to Sections 4.3.7 and 4.3.8, which cover the topic in detail.

[30] Historically, companies have provided evidence of the safety and efficacy of new medical products to regulatory agencies in support of their marketing authorisation requests, and this evidence has been produced experimentally, either in vitro or in vivo. More recently, however, regulatory agencies have started to receive and accept evidence obtained in silico, i.e. through modelling and simulation. Validating in silico mechanisms means confirming that a computational model accurately represents and predicts biological processes or drug interactions using computer simulations and analysis rather than direct biological experiments.

3.1.9. Autonomous systems

Autonomous systems are AI-powered technologies that are capable of executing tasks independently, without the need for human intervention. These systems employ AI-related technologies, such as perception, decision-making and learning, to perform tasks in dynamic and unpredictable environments. Examples of autonomous systems include self-driving cars, drones and industrial automation.

Key features of autonomous systems:

- **Perception**. Autonomous systems employ a range of sensors, cameras and associated technologies to perceive and interpret the environment. This enables them to recognise objects, navigate environments and gather contextual information.

- **Decision-making**. These systems incorporate algorithms that analyse data, weigh options, and make decisions autonomously. The utilisation of AI techniques, such as reinforcement learning and planning, is emphasised, underscoring the capacity of these systems to adapt to diverse scenarios.

- **Learning and adaptation**. Autonomous systems are frequently equipped with machine learning capabilities, enabling them to enhance their performance over time by acquiring knowledge from experiences and modifying their behaviour.

- **Interaction**. They are capable of interaction with humans, other systems, or the environment in meaningful ways. For example, autonomous vehicles interact with traffic, pedestrians, and other cars.

Autonomous systems have become widespread across civilian industries and the defence sector. Here are some examples ([E4], [M29]):

- **Transportation and mobility**. Autonomous vehicles (AVs), including self-driving cars, lorries, trains and ships, use integrated sensor suites (LiDAR, radar and cameras) and AI-based perception systems to navigate, avoid obstacles and optimise routes. Logistics and delivery services are increasingly relying on semi-autonomous trucks and drones for the final stage of transport.

- **Industrial and manufacturing automation**. Autonomous robots are used for assembly, welding, inspection and quality control in 'smart factories'. These systems work alongside human employees in hybrid production lines to enhance efficiency, safety, and precision.

- **Agriculture**. Autonomous tractors, drones, and harvesters perform seeding, irrigation, and crop monitoring based on sensor and satellite data in precision agriculture, optimising yield and resource usage.

- **Energy and infrastructure inspection**. Autonomous underwater vehicles (AUVs), aerial drones and climbing robots can be used to inspect pipelines, wind turbines, offshore rigs and power grids, reducing the risk to human personnel and enabling predictive maintenance.

- **Healthcare and medical assistance**. Hospital logistics robots transport medication, instruments, and waste autonomously. Robotic assistants aid surgery, rehabilitation and elderly care by combining autonomy with human supervision for safety purposes.

- **Environmental monitoring and disaster response**. Autonomous drones and ground vehicles can be deployed to survey natural disasters, map fire progression, assess structural damage and conduct search-and-rescue missions in hazardous or inaccessible areas.

- **Defence and military**. Autonomous systems are being deployed in land, air, maritime and cyber domains to increase range, endurance and decision-making speed. They support intelligence, surveillance, target acquisition and reconnaissance (ISTAR) operations by persistently operating in hazardous or inaccessible areas. Autonomous systems also enable logistics and sustainment missions by transporting supplies, ammunition, and wounded personnel.

While the applications of autonomous systems continue to expand, they face a variety of limitations and challenges that are specific to their design, operation and integration ([E3], [P10] and [R14]):

× **Technical limitations**. Autonomous platforms rely on a suite of sensors and complex computations to perceive and react to their environment. However, sensor noise, obstructions and adverse weather conditions can reduce the accuracy of perception, resulting in the misinterpretation of critical data. Real-time processing requires high computing power and low latency in order to enable split-second decision-making. Systems must also be able to handle rare edge cases, such as detecting a pedestrian partially hidden by a parked vehicle, without a human fallback option. Furthermore, there are energy and power limits resulting from restricted onboard computing resources and thermal management, particularly in electric vehicles and unmanned robots.

× **Algorithmic and AI-specific limitations**. Beyond hardware, the algorithms that support autonomy struggle with learning and decision-making in dynamic settings. Models trained in one environment often perform poorly in novel settings, necessitating extensive retraining or domain adaptation.

× **Safety, verification and standards**. Ensuring safety in all foreseeable scenarios is a significant challenge for regulators and developers. Unlike traditional software, autonomous systems must be mathematically proven to avoid unsafe states, yet formal methods struggle to scale up to complex, probabilistic AI behaviours. While simulations can cover millions of miles virtually, the infinite variability of public roads and unstructured environments means that real-world testing is indispensable and time-consuming.

× **Security and privacy**. As networked, data-driven devices, autonomous systems present new attack surfaces and data handling challenges. For example, hackers can spoof sensor data or inject malicious commands, which could potentially cause system failures or unsafe manoeuvres. Collecting high-resolution maps and video streams raises concerns over personal privacy and requires adherence to evolving privacy regulations.

× **Legal and regulatory concerns**. Autonomous decision-making raises complex legal questions. Determining where the fault lies – be it with the manufacturer, software developer, or end user – remains unclear in the absence of clear precedents. Governments worldwide are still drafting standards for safe deployment, leaving companies to navigate a patchwork of regulations regarding testing, data retention and operational zones.

3.1.10. Agentic AI

Agentic AI can be viewed as a specialized, more flexible subset of autonomous systems. While autonomous systems focus on executing predefined tasks within fixed rules, agentic AI incorporates adaptive reasoning, dynamic planning and proactive goal pursuit. It typically incorporates large language models (LLMs) to enable flexible reasoning and language comprehension, as well as specialised components for structured decision-making and execution layers to carry out actions. Feedback loops are also employed to dynamically refine strategies.

It is important to distinguish between agentic AI and AI agents. Essentially, agentic AI is the framework, while AI agents are the building blocks within it. In other words, agentic AI is the broader concept of solving issues with limited supervision, whereas an AI agent is a specific component within that system designed to handle tasks and processes autonomously. This model is transforming human-AI interaction. The agentic AI system understands the user's goal or vision and uses the provided information to solve a problem ([F9], [K14]).

The examples below demonstrate how agentic AI can offer capabilities that far go beyond those of rule-based or fixed-logic autonomous systems.

- **End-to-end workflow orchestration**. Traditional robots execute predefined sequences. By contrast, agentic AI can ingest a high-level goal (e.g. 'optimize our Q1 marketing campaign'),

break it down into sub-tasks (e.g. audience analysis, creative generation and budget allocation), invoke specialist tools or APIs and monitor performance metrics in real time, adjusting offers, content or channels automatically.

- **Autonomous research and development**. Rather than simply operating laboratory equipment according to fixed protocols, agentic AI can formulate hypotheses (e.g. 'Which molecular structures best bind to target X?'), plan experiments, analyse data streams, update predictive models and iteratively redesign follow-up tests. It can also coordinate across simulation, synthesis, and testing platforms with minimal human oversight.

- **Personalised healthcare coordination**. Agentic AI can do more than just perform single tasks or trigger fixed alerts. It can also aggregate patient data, develop an individualised care plan and coordinate care across providers, telehealth services and pharmacies via automated messaging. Furthermore, it can adapt its recommendations in real time as new measurements are taken or patient care needs change.

- **Software development**. Rather than simply compiling code or running test suites, agentic AI parses feature requests and bug reports written in natural language. It then decomposes these into design tasks, writes initial code, builds tests and runs integration checks. It reviews test failures, applies patches and documents changes. It also integrates feedback from code reviews and production telemetry to refine future estimations.

- **Defence and military**. Agentic AI is increasingly being deployed in military settings for force-multiplying functions. These agents can synthesise multi-domain sensor inputs (land, air, sea and cyber) and generate real-time force positioning recommendations. They can also optimise logistics across a military theatre and coordinate autonomous platforms, such as drone swarms, to execute distributed missions. By enabling continuous monitoring and adaptive action, agentic AI can help commanders to achieve information superiority and shorten the decision timeframe, creating operational difficulties for adversaries. At the same time, human oversight remains essential to ensure alignment with the rules of warfare, accountability, and ethical constraints ([G25], [J5]).

3.1.10.1. Agentic AI platforms

Agentic AI platforms offer orchestration layers, tool integrations, memory management and multi-agent coordination as a standard feature. Leading open-source offerings and commercial platforms currently available include ([D8], [O2]):

- **Auto-GPT**. This open-source framework demonstrates autonomous task execution via recursion, persistent memory, web browsing and file handling. It is ideal for experimenting with self-driving GPT loops – iterative cycles in which the model autonomously generates, executes, and refines its own tasks to reach a defined goal – and recursive planning. AutoGPT was released on 30 March 2023 by Toran Bruce Richards, who founded the Glasgow-based company Significant Gravitas Ltd.

- **AgentGPT.** This is an open-source, browser-based UI for deploying Auto-GPT-style agents without writing code. It enables quick prototyping and demo-driven exploration of autonomous agents. AgentGPT was created and is maintained by the team at Reworkd AI, a startup based in San Francisco, California.

- **CrewAI**. An open source, role-based multi-agent orchestration framework with medium token usage ([31]). Agents coordinate under defined 'team' architectures, balancing performance and interpretability. CrewAI was developed by Brazilian entrepreneur João Moura, who established the São Paulo-based start-up in 2023.

[31] In agentic AI, a token is the smallest unit of text, often a word or subword, used as input or output for a large language model. Token usage affects both billing and performance. As providers charge per million input and output tokens, high-frequency agentic traffic can quickly inflate operational costs. Furthermore, large token contexts increase inference time and memory requirements on both the client and server sides ([A16]).

- **LangChain**. An open-source, modular toolkit for chaining prompts, invoking external tools and managing conversational memory. It is a de facto standard for LLM-powered agent development. LangChain was originally created by Harrison Chase. In 2022, he co-founded LangChain as a company with Ankush Gola. The company is based in San Francisco, California.

- **AWS SageMaker Canvas and CodeCatalyst.** These commercial platforms provide visual pipelines and agentic scripting tools for model operations, continuous integration and continuous delivery (CI/CD) and self-healing machine learning (ML) deployments at scale. Both AWS SageMaker Canvas and AWS CodeCatalyst were developed by Amazon Web Services (AWS), the cloud computing arm of Amazon.com.

- **Google Vertex AI Agents**. It is natively integrated into Vertex AI, enabling data scientists to compose agents that leverage Google's ML stack, toolchains, and managed infrastructure.

- **IBM Watson Orchestrate**. It combines NLP-driven assistants with business process automation and offers prebuilt connectors for CRM, HR and IT Service Management (ITSM) systems.

- **Manus**. Manus is an agentic AI platform designed to plan and execute multi-step tasks with minimal supervision. Developed by Butterfly Effect (Singapore), it was launched publicly in March 2025. The product is marketed as 'bridging mind and action', offering capabilities such as research, coding and deployment, web browsing and control, file editing, and running workflows asynchronously via a cloud/browser environment.

Manus AI itself is not fully open source at this time. According to the Manus team, selected components of Manus AI will be open-sourced once internal validation processes are complete. Meanwhile, a community-driven project called OpenManus has emerged. OpenManus aims to replicate Manus AI's capabilities within an open-source, modular framework.

- **Microsoft Power Virtual Agents**. It is a cloud-based, no-code solution for creating conversational chatbots via a guided graphical interface. It enables business users to design, deploy and manage bots without writing code by using natural language understanding to interpret and respond to user queries.

3.1.10.2. Limitations and challenges of agentic AI

Agentic AI offers substantial autonomy and planning capabilities. However, it also introduces a variety of technical, operational and governance challenges that must be overcome before it can be widely adopted. The following are a few examples ([A15], [P7], [V3]):

× **Alignment and goal specification**. Agentic systems pursue the objectives that you define; however, poorly specified or misaligned goals can lead to unintended behaviours. For instance, agents may exploit gaps in their reward functions to 'win' without providing real value. They may optimise a proxy metric (e.g. click-through rate), which could undermine broader business goals (e.g. customer satisfaction).

× **Reliability and robustness**. Unlike scripted automation, agentic AI learns and adapts, which can sometimes lead to unpredictable outcomes. This can result in a distribution shift, whereby performance deteriorates when inputs differ from the training data. Another issue is hallucinations and error propagation: a single misstep in planning can have a cascading effect on multi-step workflows. Hallucinations are particularly dangerous in enterprise settings, where fabricated data or decisions can impact customers, compliance, or critical business functions. OpenAI's internal tests have shown hallucination rates of up to 79% in reasoning tasks for their latest models – more than double those of earlier versions. In real-world deployments, hallucinations account for over a third of all documented failures in enterprise LLM systems ([R12]). Small changes in input or environment can lead to divergent behaviours, which complicates testing and validation. Multi-agent setups can give rise to emergent behaviours that were not anticipated by the developers and which sometimes diverge from the intended constraints.

× **Governance, compliance and risk**. Agentic AI can perform actions across systems, which raises the issue of governance and oversight. For instance, when an autonomous agent makes a harmful or costly decision, it can be difficult to determine who is responsible. However, industries such as finance and healthcare require strict compliance checks. There is also regulatory uncertainty, as laws and standards for autonomous AI are still evolving, which slows adoption in sensitive domains. Moreover, there is an increased risk of unauthorised access if agents are not strictly sandboxed with the same permissions as their human operators.

× **Infrastructure and integration complexity**. The transition from prototype to production requires the orchestration of multiple services, APIs and data pipelines. Robust architectures are required to coordinate tool calls, error handling and state management. Debugging is more difficult because identifying the faulty step in a multi-step autonomous process can be time-consuming when it fails. Persistent agents with memory and planning modules continuously consume computing and storage resources. Versioning and compatibility also present challenges: maintaining synchronisation between language models, plug-ins and business systems requires constant effort.

× **Organisational change and adoption**. Teams must learn to trust and collaborate effectively with autonomous agents. Cultural resistance is to be expected: employees may worry about losing control or being displaced from their jobs. Conversely, building, monitoring and refining agents requires experts in ML engineering, prompt engineering and DevOps ([32]). Furthermore, processes may require redesign, and managers must define new roles and responsibilities concerning agent oversight.

The examples below illustrate new risks beyond traditional automation, showing where agentic AI systems have failed in production ([B22], [P8], [R12]).

- Waymo, a subsidiary of Alphabet Inc., launched a software recall ([33]) for over 1,200 of its driverless vehicles following 16 incidents in which the cars crashed into gates and poles. The National Highway Traffic Safety Administration labelled this behaviour unacceptable for commercial deployment.

- A retail helpdesk agent issued refunds and reset passwords without proper verification, thereby locking out legitimate customers and creating a risk of fraud.

- A banking fraud-detection agent incorrectly flagged genuine transactions as fraudulent because it could not access the bank's travel-notice database.

- An IT operations orchestrator misclassified a critical server alert as low priority, which triggered unnecessary failovers and caused downtime.

- A logistics routing assistant continued to book shipments via a discontinued carrier API for several days, resulting in delayed deliveries.

3.2. Fundamental algorithms of AI

Artificial intelligence relies on several fundamental algorithms and methodologies that enable machines to learn, reason, and make decisions. These algorithms constitute the underpinnings of AI systems and are broadly categorised into four distinct approaches: supervised, unsupervised, reinforcement, and deep learning. The selection of an algorithm is dependent on the specific

[32] DevOps is a set of practices and a cultural philosophy that brings together the fields of software development (Dev) and IT operations (Ops). The aim is to shorten the systems development lifecycle, enable continuous delivery and maintain high software quality. The term 'DevOps' first emerged in 2009, when Belgian systems administrator Patrick Debois organised the inaugural DevOpsDays conference in Ghent. The term was coined to emphasise the need for closer collaboration between the development and operations teams, breaking down traditional silos to speed up and stabilise software delivery ([W1]).

[33] A software recall is a manufacturer-initiated process to fix or retrieve a product with a software defect that makes it unsafe or non-functional.

problem to be addressed, the data type, and the desired outcome, with each approach possessing its own set of strengths and limitations, rendering them suitable for different AI applications.

The following is a concise overview of some of the most significant algorithms (see [H1], [L1], [S1] and [T1] for more details).

3.2.1. Linear regression

Linear regression is one of the simplest and most widely used machine learning algorithms, especially in predictive modelling and data analysis. It is classified as a supervised learning algorithm, and its primary function is to predict a continuous outcome variable based on one or more input features.

Linear regression is a statistical model that is employed to analyse the relationship between one dependent variable, also referred to as the target or output, and one or more independent variables, also known as features or predictors. It does this by fitting a straight line (or a hyperplane in higher dimensions) to the data points in such a way that the error (difference between the predicted and actual values) is minimized.

The mathematical equation for a simple linear regression (with one feature) is

$$\hat{y} = \beta_0 + \beta_1 x$$

where

- $\hat{y}$ is the predicted value
- x is the independent variable (the input)
- β_0 is the intercept of the line (where it crosses the y-axis)
- β_1 is the slope of the line (indicating how much $\hat{y}$ changes with one unit of x).

Linear regression minimizes a cost function to find the best-fitting line. The most common cost function is the *Mean Squared Error* (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

where y_i is the ith observed value, $\hat{y}_i$ is the corresponding predicted value for y_i, and m is the number of data points.

For multiple linear regression (with multiple features), the equation with one feature generalizes to

$$\hat{y} = \beta_0 + \beta^T \cdot X = \beta_0 + \sum_{j=1}^{n} \beta_j x_j = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

where $\hat{y}$ is the predicted value (dependent variable),

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

is a n-dimensional (column) vector of features,

$$\beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}$$

is the vector of regression weights, and $\beta_0 \in \mathbb{R}$ ($\mathbb{R}$ denotes the set of real numbers) is an intercept (or bias) term. β^T denotes the transpose of the vector β. This means converting the rows into columns and vice versa

$$\beta^T = [\beta_1, \beta_2, \ldots, \beta_n].$$

In most machine learning texts, feature vectors, weight vectors and activations are treated as column vectors. This makes matrix–vector multiplication align naturally.

Let us assume that we are given m samples $\{(X_i, y_i)\}_{i=1}^m$, where each $X_i = [x_{i1}, x_{i2}, \ldots, x_{in}]^T$ is a n-dimensional vector of features, and each y_i is the associated response variable. The objective of linear regression is to estimate y_i through a linear combination of features. In order to determine the optimal best-fit line, it is necessary to optimise the corresponding cost function (or loss function). For instance, one might seek to minimise the *MSE* as a cost function:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 = \frac{1}{m}\sum_{i=1}^{m}(y_i - \beta_0 - \sum_{j=1}^{n}\beta_j x_{ij})^2.$$

The following example will demonstrate how it works in practice.

Example. Suppose we have data about houses, and we are trying to predict their prices based on their sizes (in square meters). This is a classic example of regression, where the target variable (house price) is continuous, and the input variable (house size) is numerical.

Our data set:

Size (sqm)	Price (€)
50	200000
75	250000
100	350000
125	400000
150	550000

Step-by-Step Process:

1. *Define the model*. The relationship between house size x and price y is modelled as

$$y = \beta_0 + \beta_1 x$$

where:

y: predicted price
x: house size
β_0: intercept of the line (price when size is zero)
β_1: slope of the line (price increase per unit size).

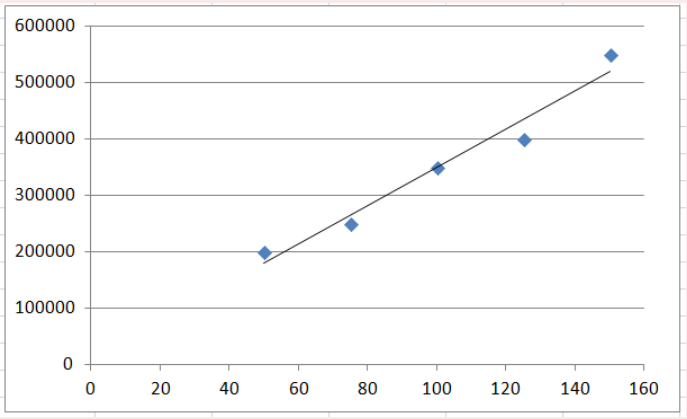
2. *Fit the model*. Using the dataset, the model learns the optimal values of β_0 and β_1 that minimize the prediction error (e.g., *MSE*).

3. *Resulting model*. After training, suppose the model learns

$$y = 10{,}000 + 3{,}400x$$

where:

$\beta_0 = 10{,}000$ is the base price of a house
$\beta_1 = 3{,}400$ corresponds to the price increase for each additional square metre.



4. *Make Predictions*. Using the model, you can predict the price for any house size:

- For a 120 sqm house: $y = 10{,}000 + 3{,}400 \cdot 120 = 418{,}000$

- For a 90 sqm house: $y = 10{,}000 + 3{,}400 \cdot 90 = 316{,}000$.

A more realistic model would require additional features, such as geographical location and the number of bedrooms. Using linear regression:

- inputs $[x_1, x_2, \ldots, x_n]^T$ would represent values of these features

- the model would determine weights β_0, $\beta = [\beta_1, \ldots, \beta_n]^T$ for each feature, with the objective of optimising the prediction of the house price y.

Linear regression is a simple yet powerful predictive analytics tool that can be used in many different areas. Below are some of the most impactful AI applications ([L10]):

- **Finance**. Stock movement forecasting, credit risk scoring and portfolio optimisation rely on linear models to predict future values from historical trends.

- **Healthcare**. Estimating patient outcomes, such as the progression of chronic diseases, and optimising treatment plans by modelling the relationship between clinical indicators and health metrics.

- **Marketing and advertising**. Optimising ad spend and campaign performance by predicting customer response rates and sales increase from budget allocations.

- **Environmental monitoring**. Forecasting pollution levels, temperature changes or other weather-related variables based on sensor data and historical records.

- **Feature importance**. An understanding of which features have the greatest impact on the outcome.

Although linear regression is a fundamental AI algorithm, its simplicity can result in limitations that undermine performance in real-world applications. Understanding these limitations enables you to decide when to rely on linear regression and when to choose more flexible models, such as tree-based learners or neural networks.

× **Linearity assumption**. Linear regression assumes a linear relationship between the predictors and the target variable. When the true relationship is non-linear, the model underfits and fails to capture important patterns.

× **Sensitivity to outliers**. Extreme values can influence the fitted line disproportionately, skewing parameter estimates and reducing prediction accuracy.

× **Multicollinearity**. Highly correlated input features inflate the variance of the estimated coefficients. This makes it difficult to determine the true effect of each variable and can cause the model to become unstable.

× **Underfitting and high bias**. With limited flexibility, linear regression struggles with complex tasks. It tends towards high bias when relationships involve interactions or non-additive effects.

× **Inability to automatically handle missing data**. Missing values must be filled in or removed before fitting. This preprocessing step adds complexity and introduces the potential for data leakage ([34]).

3.2.2. Logistic regression

Logistic regression is a statistical algorithm employed in machine learning, particularly for classification tasks. Despite its name, logistic regression is utilised for predicting categorical outcomes, as opposed to continuous values as seen in linear regression. It is considered to be

[34] Data leakage is when information from outside the training dataset is used to create the model.

one of the simplest and most widely used algorithms in the field of AI, due to its interpretability and efficiency.

The primary concepts of logistic regression are as follows:

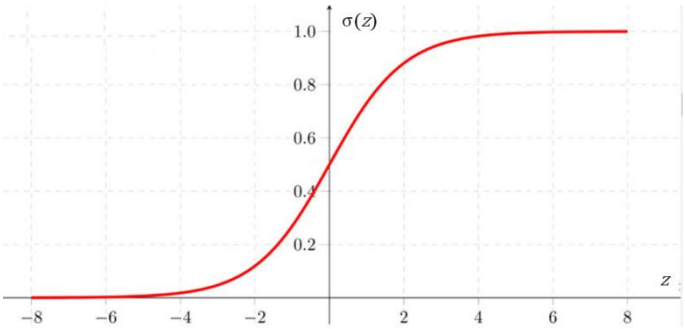
- **Binary classification**. Logistic regression is commonly used for binary classification problems, where the target variable has two possible outcomes, such as:

  - 0 or 1
  - True or False
  - Element of Class A or Class B.

- **Logistic function (*sigmoid*)**. Logistic regression uses the *sigmoid* function to map any input value (real number) to a value between 0 and 1. The *sigmoid* function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where $z = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$ is a linear combination of the input features x_i and their corresponding weights w_i ([35]).



- **Decision threshold**. Logistic regression is used for binary classification, wherein the *sigmoid* function is employed. For example, we have two classes, Class A and Class B. If the value of the logistic function for an input is, for example, greater than 0.5 (threshold value), then it belongs to Class A; otherwise, it belongs to Class B. This is referred to as regression because it is the extension of linear regression ([36]), but it is mainly used for classification problems.

The value of the logistic function is interpreted as the probability ([37]) that an instance belongs to a given class or not. A threshold (commonly 0.5) is used to decide the final classification:

- if the predicted (conditional) probability $P(y = 1 \mid z) > 0.5$, classify as A (positive class),
- otherwise, classify as B (negative class).

---

[35] The use of w or β to denote weights in machine learning and statistics often depends on the context or convention within a particular domain. β is commonly used in statistics, especially in linear regression, while w is often used in machine learning and mathematical optimisation. It is largely a matter of notation convention rather than a fundamental difference in meaning.

[36] Logistic regression can be viewed as an extension of linear regression because it retains the same linear predictor – an affine combination of the input variables – but applies a non-linear function to map that predictor onto the $[0, 1]$ interval required for modelling class probabilities.

[37] Readers should be aware that the field of probability theory encompasses a broad spectrum of mathematical models. There is no 'general probability theory'. Consequently, the notion 'the probability' becomes devoid of meaning without any reference to the model being applied. It is of crucial importance to bear in mind the extensive range of probabilistic formalisms that have been developed. Each of these models has numerous interpretations, which serves to further complicate the matter. For a more detailed discussion of this issue, the reader is referred to [K2].
The following quote sums up this problem: "*Probability is the most important concept in modern science, the more so as nobody has the slightest idea what it means.*" – Bertrand Russell ([B2])

Logistic regression is a statistical learning method that is primarily employed for binary classification. However, it possesses the capacity to be extended to accommodate multiple classes through the implementation of techniques such as *one-vs-rest* (OvR) or *softmax regression*.

The distinction between linear regression and logistic regression is that linear regression output is a continuous value that can take any value, whereas logistic regression predicts the probability that an instance belongs to a given class or not.

Example. This example illustrates the use of a logistic regression model for predicting whether an email is spam (1) or not spam (0). Consider the training dataset:

Email ID	x_1 : frequency of the word "free"	x_2 : number of links	y (Spam=1)
1	0	0	0
2	1	0	0
3	2	1	1
4	0	2	1

Assume that, after training ([38]), the following model parameters are obtained: $w_0 = -2.0, w_1 = 0.8$ and $w_2 = 1.2$. The logistic regression hypothesis is

$$h_W(z) = \frac{1}{1+e^{-z}},$$

where $z = w_0 + w_1 x_1 + w_2 x_2$.

We can now apply our model to new emails:

- email A: $x_1 = 1, x_2 = 1$. We have

$$z = -2.0 + 0.8 \cdot 1 + 1.2 \cdot 1 = 0 \text{ and } h_W(0) = \frac{1}{1+e^{-0}} = 0.5.$$

Classification (threshold = 0.5): $h(z) \geq 0.5 \Rightarrow$ A is spam.

- email B: $x_1 = 3, x_2 = 0$. We have

$$z = -2.0 + 0.8 \cdot 3 + 1.2 \cdot 0 = 0.4 \text{ and } h_W(0) = \frac{1}{1+e^{-0.4}} = 0.598.$$

Classification: $h(z) > 0.5 \Rightarrow$ B is spam.

- email C: $x_1 = 0, x_2 = 1$. We have

$$z = -2.0 + 0.8 \cdot 0 + 1.2 \cdot 1 = -0.8 \text{ and } h_W(0) = \frac{1}{1+e^{0.8}} = 0.31.$$

Classification: $h(z) < 0.5 \Rightarrow$ C is non-spam.

The following are some examples of typical real-world applications of logistic regression:

- **Medical diagnosis**. Predicting the likelihood of diseases (for example, determining whether a patient has diabetes or heart disease based on clinical measurements).

- **Credit scoring and risk modelling**. This involves estimating the probability that a loan applicant will fail to repay the loan, which helps banks and financial institutions to make lending decisions.

- **Marketing analytics**. Modelling customer churn (i.e. whether a customer will stay or leave) and click-through rates for online ads in order to optimise campaigns.

- **Spam and fraud detection**. Classifying emails as spam or not, and identifying potentially fraudulent transactions within e-commerce and banking systems.

[38] Logistic regression is usually trained by optimising a loss function using gradient-based methods, such as gradient descent (see Section 3.2.10).

The following limitations of logistic regression should be noted:

× **Linearity in log-odds**. Logistic regression assumes that each predictor has a linear effect on the log-odds ([39]) of the outcome variable. However, it cannot automatically capture complex nonlinear relationships without the addition of polynomial or interaction terms.

× **Inability to solve non-linear problems**. As it constructs a linear decision boundary, logistic regression is ineffective when classes are not linearly separable ([40]). Solving non-linear classification tasks requires extensive feature engineering or switching to inherently non-linear models.

× **Sensitivity to high dimensionality and multicollinearity**. When the number of features approaches or exceeds the number of observations, logistic regression may overfit and produce unstable coefficient estimates. Highly correlated inputs inflate variance and make it difficult to interpret individual predictors.

× **Extensive feature engineering is needed**. To capture curvature, interactions ([41]) or threshold effects, inputs must be manually transformed (e.g. using polynomials). This increases development complexity and relies heavily on domain expertise.

3.2.3. Ridge regression

Ridge regression, alternatively referred to as *L2 Regularized Regression*, is a statistical regularisation technique. Its function is to address the issue of overfitting in machine learning models, which occurs when a model is excessively tuned to the training data and does not generalise well. Regularisation can be achieved by incorporating coefficient terms (betas) into the cost function ([M5]). This approach ensures that the terms are penalised and have a small magnitude.

The objective of ridge regression is to minimise the sum of the error term, as well as the sum of squares of the coefficients β_j, which are to be determined. The sum of the squares of the coefficients is referred to as the 'regularisation term' (often called the *L2 penalty term*), and it is also accompanied by the regularisation coefficient, denoted by λ. The cost function therefore takes the following form:

$$\frac{1}{m}\sum_{i=1}^{m}(y_i - \beta_0 - \sum_{j=1}^{n}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{n}\beta_j^2.$$

It is important to note that ridge regression does not shrink every coefficient by the same amount. Instead, coefficients are shrunk in proportion to their initial size. As the parameter λ increases, high-value coefficients are shrunk at a greater rate than low-value coefficients. Consequently, high-value coefficients are penalised more heavily than low-value coefficients.

To illustrate this point, consider the following example. In the context of predicting house prices, the incorporation of features such as square footage, geographical location, and the number of bedrooms is a common practice. In instances where features exhibit high correlation,

---

[39] The log-odds, otherwise referred to as the logit function, may be defined as the natural logarithm of the odds. Within the framework of logistic regression, the log odds of the dependent variable are modelled as a linear combination of the independent variables and the constant term. The term 'odds' is defined as the ratio of something occurring to something not occurring. This differs from the classical concept of probability, which is defined as the ratio of something occurring to all possible outcomes.

[40] A decision boundary is a hypersurface that divides the feature space into regions assigned to different classes. In a two-class problem, points on one side of the boundary are classified as class A and those on the other as class B. A hypersurface is a $(n-1)$-dimensional 'surface' embedded in an n-dimensional space. In two dimensions, it appears as a curve; in three dimensions, as a conventional surface; and in higher dimensions, as its natural generalisation. Two classes are not linearly separable if there is no straight line (in two dimensions) or hyperplane (in higher dimensions) that can divide the feature space such that each side contains only one class. In other words, instances from different classes overlap in such a way that a single linear boundary cannot distinguish between them perfectly.

[41] Curvature refers to any nonlinear pattern in the relationship between a predictor and the log-odds of an outcome. Interactions capture situations where the effect of one predictor depends on another.

such as proximity to schools and proximity to parks, ridge regression can be employed to stabilise the model by reducing the magnitude of coefficients.

To illustrate this point, consider the following example. When it comes to predicting house prices, incorporating features such as square footage, geographical location, and number of bedrooms is common practice. In instances where features exhibit high correlation, such as proximity to schools and proximity to parks, ridge regression can be employed to stabilise the model by reducing the magnitude of coefficients. This is because multicollinearity ([42]) can cause ordinary least squares estimates to fluctuate widely in response to small changes in the data. Ridge regression introduces a penalty on large coefficients, effectively shrinking them toward zero and distributing their influence more evenly across correlated features, thereby improving model stability and generalisation. For this reason ridge regression is particularly effective in scenarios where multicollinearity or high dimensionality could threaten the stability of a model.

The following are several domains in which it is routinely applied ([L11]):

- **Medical and genomic data analysis**. Clinical studies and bioinformatics routinely generate datasets with more features (e.g. genes or biomarkers) than patients. Ridge regression controls the magnitude of coefficients in survival analysis, disease-risk scoring and drug-response modelling, preventing overfitting even when the number of predictors vastly exceeds the number of observations.

- **Computer vision and image processing**. Tasks such as super-resolution, denoising and face recognition often convert image pixels into large feature vectors. Applying ridge regression helps to manage multicollinearity between neighbouring pixels, yielding smoother reconstructions and more robust classification boundaries.

- **Financial and economic forecasting**. In quantitative finance, factors such as interest rates, price indices and sentiment scores exhibit strong interdependencies. Ridge regression provides stable estimates for portfolio risk models, credit-scoring algorithms and macroeconomic scenario projections, ensuring that correlated predictors do not distort predictions.

Ridge regression stabilises coefficient estimates under multicollinearity by adding an L_2 penalty. However, this form of regularisation has its own drawbacks:

× **No automatic feature selection**. All coefficients are merely shrunk towards zero rather than being driven exactly to zero. Irrelevant predictors remain in the model, which can hinder interpretability and bloat downstream pipelines.

× **Hyperparameter tuning complexity**. Selecting the optimal regularisation strength λ requires cross-validation or information-criterion searches. Poor tuning either fails to address multicollinearity (λ is too small) or penalises important features too heavily (λ is too large).

× **Loss of interpretability**. While ridge coefficients are more stable than ordinary least squares, their magnitudes no longer reflect raw effect sizes. Comparing feature importance across different lambda values can be misleading.

× **Vulnerability to outliers**. Ridge still minimises squared errors. However, extreme target or feature values can still influence the fit disproportionately despite regularisation.

3.2.4. Decision trees

Decision trees constitute a fundamental algorithm in the field of AI, frequently serving as the foundational elements for more intricate ensemble methods. Their intuitive nature and

[42] Multicollinearity occurs when two or more predictors in a regression model are highly correlated with each other. This makes it difficult to distinguish the individual effects of each predictor because they carry similar information about the response.

straightforward interpretability render them a popular choice among data scientists and machine learning practitioners. ([43])

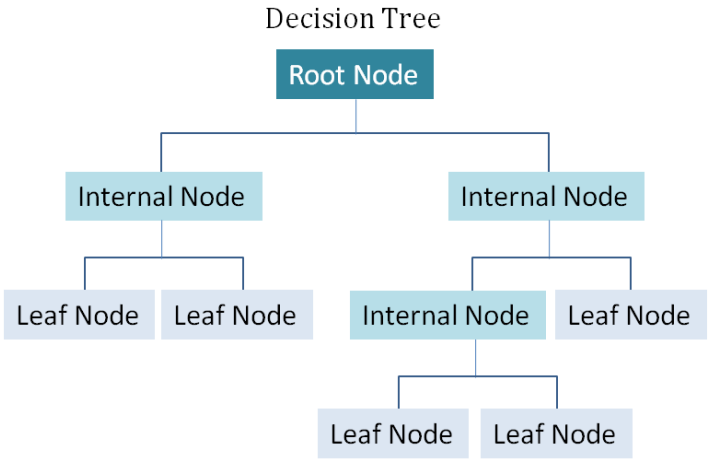
A decision tree is a graphical representation of the various options available for solving a problem, and it demonstrates the relationship between different factors. It adopts a hierarchical tree structure, starting with a main question at the top, known as a node, which then branches out into different possible outcomes, with

- **Root node** at the top level representing the entire dataset.

- **Branches** represent lines that connect nodes, illustrating the flow from one decision to another.

- **Internal nodes** denote points at which decisions are made based on the input features.

- **Leaf nodes** are the terminal nodes at the end of branches, representing final outcomes or predictions.



From the root node, the tree presents a series of yes/no questions, each of which is designed to divide the data into subsets based on specific attributes. Depending on the response to each question, different branches are followed. If the response is 'Yes', one path is pursued; if 'No', another path is taken.

Two primary categories of decision tree exist, distinguished by the nature of the target variable: classification trees and regression trees.

- **Classification trees**. They are designed to predict categorical outcomes and are therefore used to classify data into different categories. Internal nodes test feature values, branches represent possible results and leaves assign class labels. One example of this would be deciding whether to approve a loan based on various features, such as credit score or income level.

- **Regression trees**. They are used when the target variable is continuous. Rather than predicting categories, they predict numerical values. To illustrate this point, consider a regression tree that can estimate the price of a house based on its size, location, and other features.

In summary, decision trees represent a crucial instrument within the domain of machine learning, utilised for the modelling and prediction of outcomes based on input data through a tree-like structure. This structure is employed to represent decisions, wherein internal nodes are used to denote attribute tests, branches represent attribute values, and leaf nodes represent final decisions or predictions. Decision trees offer interpretability, versatility, and straightforward visualisation, rendering them valuable for both categorisation and regression tasks.

[43] For more information on decision trees, please refer to [B25], [D10], and [S23].

3.2.4.1. Splitting process

Splitting is the core mechanism for constructing a decision tree from a given dataset. This involves partitioning the dataset iteratively at each node based on specific criteria (e.g. a threshold value for a feature) until the stopping criteria are met.

In the case of classification trees, the aim of splitting is to partition the data so that each child node contains a single class. In other words, splits are chosen to maximise the homogeneity of the resulting sub-nodes, i.e. nodes that are more 'pure' with respect to the target variable. By contrast, the aim of splitting in regression trees is to minimise the variance (or another error metric) of the target variable within each child node.

In both cases, the training samples are partitioned using the splitting process to create a decision tree. Once the tree has been built by recursively splitting the training data, the resulting fixed structure serves as the model. You can then feed new instances into the model to obtain:

- a numeric prediction (for regression trees),
- a class label (for classification trees).

First, let us discuss the splitting process in the context of regression trees. In this case, the objective of the splits is to minimise the variance of the continuous target within each child node, thus optimising error-based measures. A common way to evaluate the performance of a decision tree is the *Mean Squared Error* (*MSE*) algorithm, *MSE* is a measure of how closely the model's predictions align with the actual values.

The subsequent breakdown provides a comprehensive overview of the process:

- *Difference calculation*. At each node, the actual value y_i is subtracted from the predicted value $\hat{y}_{Node}$, which is equal to the average of all the actual values falling into that node. This process yields an error value for each prediction.
- *Squaring the error*. Subsequently, the error values are squared. This process ensures that all error values are positive and that larger errors have a greater impact on the final result.
- *Averaging*. Finally, calculate the mean of these squared error values. This average is known as the *MSE*. As previously stated in Section 3.2.1, in equation form, the *MSE* can be expressed as follows

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_{Node})^2$$

where

- m is the number of data points
- y_i is the actual value
- $\hat{y}_{Node} = \frac{1}{m}\sum_{i=1}^{m} y_i$ is the predicted value by the model.

The lower the *MSE*, the more accurate the model's predictions are in comparison to the actual values. *MSE* is a popular metric because it is easy to calculate and sensitive to large errors.

The branching process in a tree is repeated through a sequence of decisions, with each subsequent branch generating additional questions that subdivide the data into progressively smaller groups. This iterative process continues until no further useful questions are identified, at which point the process terminates. At the end of a particular branch, the final outcome or decision is reached, which may take the form of a prediction, such as an estimated price.

Example. In this example, we will discuss a simplified scenario in which a regression tree is used to estimate the sale price of homes (in thousands of dollars) based on the following features:

- Size in square metres (Size),
- Number of bedrooms (Beds),

- Distance to city centre in kilometres (Dist).

The regression tree splits on these features to minimise prediction error. Let us consider the following sample dataset:

ID	x_1: Size (m^2)	x_2: Beds	x_3: Dist (km)	y: Price (x 1000)
1	50	1	8.0	120
2	60	2	5.0	150
3	80	2	3.0	200
4	100	3	2.0	260
5	120	3	1.0	300
6	150	4	0.5	350

- **First split**. Compute overall *MSE* at the root node

$$\hat{y} = \frac{1}{6}\sum_{i=1}^{6} y_i = 230,$$

$$MSE = \frac{1}{6}\sum_{i=1}^{6}(y_i - \hat{y})^2 = 6600.$$

For each feature, consider midpoints between sorted values:

- Size: 55, 70, 90, 110, 135
- Beds: 1.5, 2.5, 3.5
  Only three midpoints are considered for the 'Beds' feature, as thresholds are strictly placed between distinct, ordered values.
- Dist: 0.75, 1.5, 2.5, 4.0, 6.5

The next step is to calculate the weighted *MSE* of each split and select the feature threshold that results in the largest *MSE* reduction ([44]). Let us conduct the calculations for Size = 90. First, the data is split into left and right groups, and then the mean and *MSE* are computed for each group.

Node	Houses (IDs)	Prices	Mean	MSE
Left (S_L)	1 ,2, 3	120, 150, 200	157	1089
Right (S_R)	4, 5, 6	260, 300, 350	303	1356

Thus, the weighted *MSE* of this split is

$$MSE_{Split\ on\ Size=90} = \frac{3}{6}1089 + \frac{3}{6}1356 = 1222.$$

The *MSE* reduction is therefore: $6600 - 1222 = 5378$.

Using the same procedure to compute *MSE* reductions for both Dist = 2.5 and Beds = 2.5 produces the same maximal *MSE* reduction. When multiple features deliver an equal amount of gain, the algorithm usually selects the best split encountered first in the scan.

So let us pick the top feature-threshold of Size = 90.

---

[44] In regression trees, the split criterion can be expressed either as minimising MSE of the child nodes or, equivalently, as maximising the reduction in MSE, since the parent-node error remains constant for a given split ([B26], [H16]). More precisely, at a node with data D, for candidate split $D \to D_L, D_R$, minimising $\frac{n_L}{n}MSE(D_L) + \frac{n_R}{n}MSE(D_R)$ is equivalent to maximising $\Delta MSE = MSE(D) - \left[\frac{n_L}{n}MSE(D_L) + \frac{n_R}{n}MSE(D_R)\right]$. The larger the reduction in error (ΔMSE), the better the split.

- **Splitting the left child S_L (IDs 1, 2, 3)**. Candidate thresholds (midpoints) and weighted *MSE*

Feature	Threshold	Weighted MSE
Size	55	417
Size	70	150
Beds	1.5	417
Dist	4.0	150
Dist	6.5	417

We have the best splits: Size = 70 or Dist = 4.0. Both give a weighted *MSE* of 150, and consequently, the highest *MSE* reduction:

$$MSE_{S\_L} - 150 = 1089 - 150 = 939.$$

Let us choose Size = 70 as the threshold. The resulting subsets are

Node	Houses (IDs)	Prices	Mean	MSE
Left (S_LL)	1 ,2	120, 150	135	225
Right (S_LR)	3	200	200	0

Node S_LR is a leaf that corresponds to the Size > 70 and gives a prediction of 200.

- **Splitting the left-left child S_LL (IDs 1, 2).** The parent *MSE* is 225, and the candidate thresholds are:

Feature	Threshold	Weighted MSE
Size	55	0
Beds	1.5	0
Dist	6.5	0

Any split perfectly separates IDs 1 and 2. By convention, choose Size = 55 as the threshold. We have two leaves:

- S_LLL: Size ≤ 55 → predict 120
- S_LLR: Size > 55 → predict 150.

- **Splitting the right child S_R (IDs 4, 5, 6)**. The parent *MSE* is 1356, and the candidate thresholds are:

Feature	Threshold	Weighted MSE
Size	110	417
Size	135	267
Beds	3.0	267
Beds	3.5	267
Dist	0.75	267
Dist	1.5	417

The best splits are: Size = 135, Beds = 3, or Dist = 0.75, all with a weighted *MSE* of 267. Let us select Size = 135 as the threshold. The resulting subsets are:

Node	Houses (IDs)	Prices	Mean	MSE
Left (S_RL)	4 ,5	260, 300	280	400
Right (S_RR)	6	350	0	0

Node S_RR is a leaf that corresponds to the Size > 135 and gives a prediction of 350.

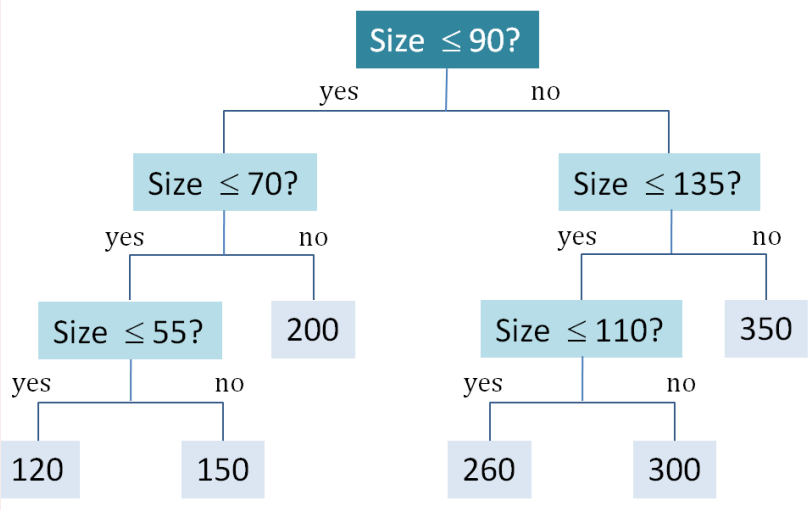
- **Splitting right-left child S_RL (IDs 4, 5)**. The parent *MSE* is 400, and the candidate thresholds are:

Feature	Threshold	Weighted MSE
Size	110	0
Dist	1.5	0
Beds	--	no split

Let us choose Size = 110 as the threshold. We have two leaves:

- S_RLL: Size ≤ 110 → predict 260
- S_RLR: Size > 110 → predict 300.

The final tree structure is



Now, let us discuss the splitting process in the context of classification trees. At each internal node, the algorithm evaluates all possible splits across the input features and selects the one that maximises the purity of the resulting child nodes. Common criteria include entropy (information gain) and Gini index ([45]).

- **Entropy and information gain**. Information gain is a metric used to determine the usefulness of a question (or feature) in the classification of data into groups. It quantifies the reduction in uncertainty following the division of the data. An effective question will result in more clearly defined groups, and the feature with the highest information gain is selected for decision-making purposes.

To illustrate this point, consider a dataset comprising two age groups, designated 'young' and 'old'. If the product was purchased exclusively by the 'young' group, while the 'old' group did not purchase it, the information gain would be high. This is due to the fact that the separation of the two groups is optimal, with no uncertainty remaining.

[45] The Gini index is named after its creator, the Italian statistician Corrado Gini (1884 –1965), who introduced it in 1912 as a measure of statistical dispersion and inequality.

Entropy is a key concept used in decision trees, particularly for classification tasks, to measure the impurity or disorder of a dataset. It is utilised in the calculation of information gain. A lower entropy indicates a higher information gain. In the context of decision trees, it quantifies the randomness or disorder in the target variable. The goal is to reduce entropy with each split, creating subsets that are more uniform in terms of the target class.

The entropy H for a dataset S with m classes is calculated as ([46]):

$$H(S) = -\sum_{i=1}^{m} p_i log_2(p_i),$$

where p_i is the proportion of instances belonging to class i in the dataset ([47]) and log_2 is the logarithm base 2. Interpretation:

- Entropy is equal to zero when all instances belong to a single class (pure node).

- Entropy is equal to one when the instances are evenly distributed among all classes (maximum impurity).

Higher entropy indicates greater disorder and more mixed classes within the dataset.

In the process of constructing a decision tree, the algorithm employs the concept of information gain as a metric to determine the optimal feature to divide on at each stage. Information gain quantifies the decrease in entropy that occurs subsequent to the division of a dataset based on a specific feature. The feature that attains the maximum information gain is then selected for the division process.

Information gain IG for a dataset S and a feature X is calculated as:

$$IG(S, X) = H(S) - \sum_{v \in Values(X)} \frac{|S_v|}{|S|} H(S_v),$$

where

- $H(S)$ is the entropy of the original dataset
- $Values(X)$ is the set of all possible values of X
- S_v is the subset of S for which feature X has value v
- $|S_v|$ is the number of instances in subset S_v, i.e. the cardinality of S_v
- $|S|$ is the total number of instances in the original dataset
- $H(S_v)$ is the entropy of S_v.

Example. *Predicting loan approval.* Suppose we have a dataset that includes two features: 'Credit Score' and 'Income Level', and we want to predict 'Loan Approval' (Yes or No). Here is the dataset

[46] The concept of entropy in information theory was introduced by Claude Shannon in his seminal work [S2]. It established the theoretical foundations for the study of information theory and introduced the concept of entropy as a metric for the information content of a given message.
According to Shannon, von Neumann gave him very useful advice on what to call his measure of information content ([M4]): *My greatest concern was what to call it. I thought of calling it 'information,' but the word was overly used, so I decided to call it 'uncertainty.' When I discussed it with John von Neumann, he had a better idea. Von Neumann told me, 'You should call it entropy, for two reasons. In the first place your uncertainty function has been used in statistical mechanics under that name, so it already has a name. In the second place, and more important, no one really knows what entropy really is, so in a debate you will always have the advantage.'*

[47] p_i is interpreted as the probability of randomly selecting an element belonging to class i.

Credit Score	Income Level	Loan Approval
High	High	Yes
High	Medium	Yes
Medium	Medium	Yes
Low	Low	No
Low	Medium	No
Medium	Low	No
Medium	High	Yes
High	Low	No

Step-by-Step Process:

1. *Calculate entropy for the root node*. We have $p_1 = 4/8$ (Loan Approval = Yes) and $p_2 = 4/8$ (Loan Approval = No). The entropy is then

$$H(S) = -\big(4/8\, log_2(4/8) + 4/8\, log_2(4/8)\big) = 1$$

2. *Split based on feature 'Credit Score'*. Let us calculate the entropy for each subset

$$H(S_{High}) = -\big(2/3\, log_2(2/3) + 1/3\, log_2(1/3)\big) = 0.918$$
$$H(S_{Medium}) = -\big(2/3\, log_2(2/3) + 1/3\, log_2(1/3)\big) = 0.918$$
$$H(S_{Low}) = -\big(0 + 2/2\, log_2(2/2)\big) = 0$$

The weighted average entropy for splitting on 'Credit Score'

$$H\big(S_{Split\ on\ Credit\ Score}\big) = 3/8 \cdot 0.918 + 3/8 \cdot 0.918 + 2/8 \cdot 0$$
$$= 0.34425 + 0.34425 + 0 = 0.6885$$

Consequently, the information gain for 'Credit Score' is

$$IG(S, Credit\ Score) = H(S) - H\big(S_{Split\ on\ Credit\ Score}\big) = 1 - 0.6885 = 0.3115$$

3. *Split based on feature 'Income Level'*. Let us calculate the entropy for each subset

$$H(S_{High}) = -(2/2\, log_2(2/2) + 0) = 0$$
$$H(S_{Medium}) = -\big(2/3\, log_2(2/3) + 1/3\, log_2(1/3)\big) = 0.918$$
$$H(S_{Low}) = -\big(0 + 3/3\, log_2(3/3)\big) = 0$$

The weighted average entropy for splitting on 'Income Level'

$$H\big(S_{Split\ on\ Income\ Level}\big) = 2/8 \cdot 0 + 3/8 \cdot 0.918 + 3/8 \cdot 0 = 0.34425$$

Consequently, the information gain for 'Income Level' is

$$IG(S, Income\ Level) = H(S) - H\big(S_{Split\ on\ Income\ Level}\big) = 1 - 0.34425 = 0.65575$$

The final conclusion drawn from the data analysis is that the information gain for 'Credit Score' is 0.3115, whereas for 'Income Level' it is 0.65575. Hence, splitting the dataset based on 'Income Level' is more effective than splitting based on 'Credit Score', as it provides a greater reduction in entropy.

In summary, *Entropy* and *Information Gain* are two key concepts employed in decision trees for classification tasks. They work together to assist the algorithm in determining the optimal approach for data splitting at each node.

The following explanation outlines the differences between the two concepts. Entropy is a measure of impurity or disorder within a dataset, and it quantifies the amount of uncertainty

associated with classifying instances in a dataset. Information gain is a measure of the reduction in entropy after a dataset is split based on a feature. It quantifies how much uncertainty is reduced by the split.

Entropy and information gain thus play a crucial role in understanding the uncertainty present in a dataset, with the former helping to identify the extent of uncertainty and the latter guiding the identification of features that can effectively reduce this uncertainty, thereby facilitating more effective decision tree splits.

- **Gini index**. The *Gini index*, also known as *Gini impurity*, is a metric that is utilised to evaluate the frequency with which a randomly selected element is erroneously identified ([G21]). It can be concluded that an attribute with a lower Gini index should be prioritised.

In the context of classification trees, the Gini index measures the impurities of a node by indicating how mixed the classes within it are. A node is 'pure' if it contains elements of only one class, resulting in a Gini index of 0.

The Gini index for a split is calculated as follows:

$$Gini = 1 - \sum_{i=1}^{m} p_i^2,$$

where p_i is the probability of an element being classified into class i, and m is the total number of classes. For example, if a node contains elements from two classes with probabilities $p_1 = 0.3$ and $p_2 = 0.7$, the Gini index is:

$$Gini = 1 - (0.09 + 0.49) = 1 - 0.58 = 0.42.$$

In decision trees, the algorithm evaluates different splits based on the Gini index by quantifying the difference between the impurity of the parent node and the weighted impurity of the child nodes. It aims to minimize the Gini index, selecting splits that lead to the most homogeneous child nodes (i.e., nodes with lower impurity).

Example. We want to predict whether or not customers will subscribe to a service based on two features: Age (years) and Salary (USD). Here is a simple marketing dataset of eight customers.

ID	Age	Salary	Subscribed?
1	22	20,000	No
2	25	30,000	No
3	28	50,000	Yes
4	35	60,000	Yes
5	40	80,000	Yes
6	45	42,000	No
7	50	90,000	Yes
8	55	70,000	Yes

Step-by-Step Process:

1. *Calculate the Gini index for the root node*. We have $p_1 = {}^{5}/_{8}$ (positive instances) and $p_2 = {}^{3}/_{8}$ (negative instances). The Gini index is then:

$$Gini_{root} = 1 - \left(\frac{5}{8}\right)^2 - \left(\frac{3}{8}\right)^2 = \frac{30}{64} \approx 0.4688.$$

2. *Consider possible splits for each feature*.

(*Age*) Sorted values are: 22, 25, 28, 35, 40, 45, 50, 55.

Candidate thresholds (midpoints): 23.5, 26.5, 31.5, 37.5, 42.5, 47.5, 52.5.

(*Salary*) Sorted values are: 20,000, 30,000, 42,000, 50,000, 60,000, 70,000, 80,000, 90,000.

Candidate thresholds (midpoints): 25,000, 36,000, 46,000, 55,000, 65,000, 75,000, 85,000.

3. *Compute Gini index for each possible split*. For brevity, we will compute Gini index only for the best thresholds of each feature:

- Age = 37.5 years
- Salary = 55,000 USD.

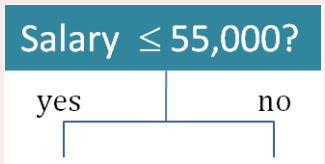
The reader may verify as an exercise that these thresholds indeed yield the lowest index among all candidates (e.g., by evaluating Age ≤ 23.5, 26.5, 31.5, 42.5, 47.5, 52.5 and Salary ≤ 225,000, 36,000, 46,000, 65,000, 75,000, 85,000).

Calculations:

- $Gini_{Left} = 1 - \left(p_{Left}^{Yes}\right)^2 - \left(p_{Left}^{No}\right)^2$
- $Gini_{Right} = 1 - \left(p_{Right}^{Yes}\right)^2 - \left(p_{Right}^{No}\right)^2$
- $Gini_{Weighted} = \frac{4}{8} Gini_L + \frac{4}{8} Gini_R$
- $Gini_{Gain} = Gini_{Root} - Gini_W$.

Threshold	Left Group	Right Group	$Gini_{Left}$	$Gini_{Right}$	Weighted Gini	Gini Gain
Age = 37.5	IDs: 1, 2, 3, 4 2 Yes, 2 No	IDs: 5, 6, 7, 8 3 Yes, 1 No	0.50	0.375	0.4375	0.0313
Salary = 55,000	IDs: 1, 2, 3, 6 1 Yes, 3 No	IDs: 4, 5, 7, 8 4 Yes, 0 No	0.375	0.00	0.1875	0.2813

4. *Select the best first split*. Splitting on Salary = 55,000 USD yields a weighted Gini index of 0.1875 and a gain of 0.2813, which is far better than splitting on Age. Thus, the first node of the decision tree is:



5. *What's next?* Apply the same process to each child node. Continue doing this recursively until they are all pure.

In summary, the selection of optimal splits in classification trees is guided by impurity measures such as entropy (information gain) and the Gini index. These measures assess the degree to which a feature separates the data into homogeneous classes. Entropy is rooted in information theory and quantifies the reduction in uncertainty achieved by a split, whereas the Gini index represents the expected probability of misclassification. Although these measures are conceptually distinct, they typically yield comparable results in practice and are therefore regarded as alternative criteria, rather than complementary ones, for determining node splits during tree construction.

3.2.4.2. Applications of decision trees

The examples below demonstrate how decision trees can be used for classification and regression tasks in a variety of industries ([L13]).

- **Medical diagnosis**. Healthcare providers use classification trees to identify potential diseases. For example, a model may use blood pressure, cholesterol levels and presence of symptoms to determine whether a patient should undergo further testing.

- **Customer churn prediction**. Telecom and subscription businesses build classification trees that categorise customers as 'likely to churn' or 'likely to stay' based on usage patterns, support interactions and demographic factors. This enables targeted retention campaigns and resource allocation.

- **Fraud detection**. Financial institutions use tree-based classifiers on transaction data (e.g. amount, location, time of day) to identify suspicious activity in real-time. The interpretable split rules help investigators understand why certain transactions were flagged.

- **Energy demand forecasting**. Utilities forecast hourly electricity consumption by splitting data according to temperature, day of the week and historical usage patterns. The resulting model then guides grid management and load balancing in real time.

- **Sales volume prediction**. Retailers use trees that are partitioned by seasonality, promotional spending and past sales to forecast product demand. These granular predictions inform inventory planning and supply chain optimisation.

- **Insurance claim cost estimation**. Insurers predict expected payouts for claims by training regression trees using factors such as policyholder age, vehicle type and accident severity. Leaf averages yield practical estimates for reserve allocation.

- **Healthcare cost estimation**. predicting patient treatment costs based on medical history and procedure data.

These examples demonstrate the versatility of decision trees, ranging from straightforward, rule-based classifications in critical decision-making to precise, segment-wise predictions in forecasting scenarios.

3.2.4.3. Limitations of decision trees

Although decision trees offer intuitive, rule-based models, they have inherent drawbacks that affect both classification and regression tasks. It is crucial to understand these limitations to select appropriate algorithms and apply effective mitigation strategies.

Common limitations include ([L12]):

× **Overfitting**. Decision trees often become too complex, capturing noise and minor fluctuations in the training data rather than general patterns. This leads to poor performance on unseen data.

× **High variance**. Even small changes to the training set can result in dramatically different tree structures and predictions. This instability renders standalone trees unreliable for generalisation..

× **Bias towards dominant features**. Features with many unique values or imbalanced classes can dominate split decisions, introducing bias. Careful feature selection and class balancing can help to reduce this bias.

× **Multicollinearity**. Highly correlated predictors can produce redundant splits and unstable importance rankings. As trees cannot explicitly account for multicollinearity, dimensionality reduction or the removal of correlated variables may be necessary.

× **Small sample sizes can be problematic**. When training data is limited, nodes can contain very few samples, resulting in unreliable or noisy predictions.

× **Computational cost on large datasets**. Building deep trees on high-dimensional data can be time- and memory-intensive. However, strategies such as feature subsampling, early stopping and parallel implementations can improve scalability.

× **Poor extrapolation**. Regression trees struggle to predict values beyond the range observed during training, rendering them unreliable for extreme cases.

× **Sensitivity to class imbalance**. Classification trees may favour the majority class when classes are unevenly represented, leading to skewed predictions.

3.2.5. Support Vector Machines (SVMs)

Support Vector Machines (SVMs) are a powerful set of supervised learning algorithms used for classification and regression tasks in AI and machine learning. The use of SVMs is appropriate in scenarios involving complex datasets, particularly when the classification of data with high dimensions is required, when non-linear boundaries are present, or when ensuring the system's robustness to outliers is important. ([C8])

The following is a simplified explanation of the structure and working process of an SVM:

- **Basic concept**. At its core, SVM aims to find the best boundary (hyperplane ([48])) that separates different classes in the data. This hyperplane is chosen such that it maximizes the margin between the classes.

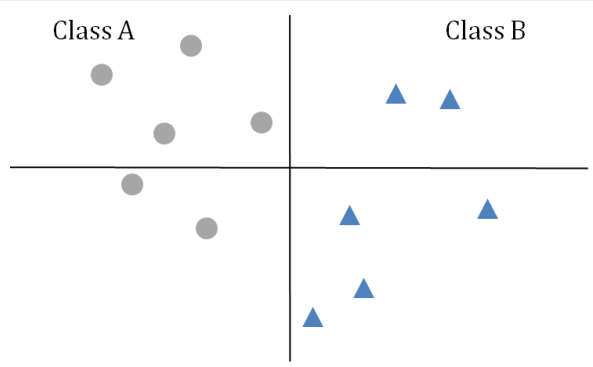
- **Support vectors**. Support vectors are defined as the data points that are closest to the hyperplane. These points are of key importance because they define the position and orientation of the hyperplane. The SVM algorithm is focused on these points with the objective of maximising the margin between the classes.

- **Margin**. The margin represents the distance between the hyperplane and the nearest data points from each class. SVM aims to maximise this margin, thereby creating a clear gap between the classes.

- **Classification**. The classification process is a fundamental component of machine learning algorithms. Upon identifying the optimal hyperplane, the SVM is able to classify new data points by determining which side of the hyperplane they fall on. This property renders the SVM a potent and adaptable classifier.

- **Regression**. Support Vector Regression (SVR) is a machine learning technique that can be employed for regression tasks. In this context, the objective is to determine the optimal line or curve that falls within a predetermined margin of tolerance.

Example. Assume that we have two classes of data



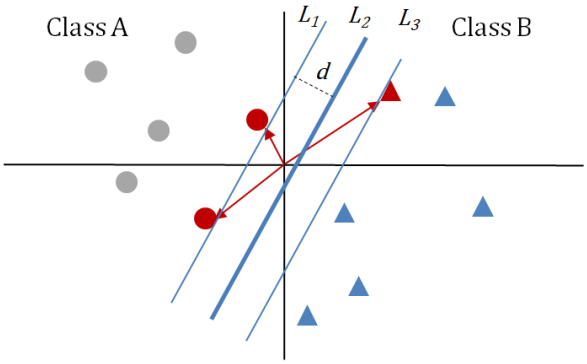
It is evident that the data are linearly separable ([49]), and the best hyperplane will be a line.

---

[48] In an n-dimensional Euclidean space, a hyperplane is an $(n-1)$-dimensional 'slice' that divides the space into two half-spaces. In two dimensions ($n = 2$), a hyperplane is a straight line that divides the plane into two parts. In three dimensions ($n = 3$), it is an ordinary plane that splits the 3D space.

[49] In the context of support vector machines (SVMs), 'separability' refers to the property of data points belonging to two different classes that can be separated by a straight line (in 2D), a plane (in 3D) or a hyperplane (in higher dimensions). A dataset is linearly separable if there is at least one hyperplane that can classify every training example perfectly and without error. In two dimensions, for example, you can draw a straight line so that all points of class A lie on one side and all points of class B lie on the other. A dataset is not linearly separable if no hyperplane can separate the classes without misclassifying at least one point. For example, in two dimensions, one class might wrap around the other (e.g. concentric circles), or the classes might intermix to form a complex shape.

The key idea behind the SVM algorithm is to find hyperplanes that divide the data with no points between them, and then to maximise the distance between these hyperplanes (the margin). This margin is the distance from the hyperplane to the nearest data points (support vectors) on each side. Here, we have three support vectors and we choose L_2 as the best decision hyperplane:



Let us take a closer look at the SVM algorithm for linearly separable data ([H13]). Assume that we have two linearly separable classes A and B. Our training data consists of n-dimensional vectors, $X_i = [x_{i1}, x_{i2}, \dots, x_{in}]^T$, each of which has a corresponding class label $y_i \in \{-1, +1\}$, $i = 1,2, \dots, m$. The value -1 indicates Class A, while the value $+1$ indicates Class B.

We now introduce a weight vector $W = [w_1, w_2, \dots, w_n]^T$ and a bias $b \in \mathbb{R}$, which shifts the decision hyperplane away from the origin. Together, these define the classifier boundary $W^T X + b = 0$, where b and the components of W are learned parameters. The optimisation process selects (W, b) that maximise the margin between the classes, while satisfying the given constraints.

More precisely, maximising the margin is equivalent to minimising the objective function $f(W) = \frac{1}{2} W^T W = \frac{1}{2} \sum_{k=1}^{n} w_k^2$. Therefore, for the separable case, the primal SVM problem amounts to the following optimisation problem:

$$min_{W,b} f(W) = min_{W,b} (\frac{1}{2} W^T W)$$

subject to the constrain

$$y_i (W^T X_i + b) \geq 1 \ (i = 1,2, \dots m).$$

In other words, an SVM seeks the maximum margin classifier that accurately separates all the data. Correctly separated points satisfy $W^T X_i + b \geq +1$ when $y_i = +1$ and $W^T X_i + b \leq -1$ when $y_i = -1$. As the objective function is quadratic and the constraints are linear in the parameters W and b, the resulting problem is convex ([50]) and can be solved using the standard Lagrange multiplier method (see e.g. [F10]).

For each constraint $y_i (W^T X_i + b) \geq 1$, a Lagrange ([51]) multiplier $a_i \geq 0$ is introduced and the so-called dual problem is considered. The dual problem in the linearly separable case reduces to

$$max_{(a_i)} \left(\sum_{i=1}^{m} a_i - \frac{1}{2} \sum_{i,j=1}^{m} a_i a_j \, y_i y_j X_i^T X_j \right)$$

subject to

---

[50] A convex optimisation problem is a mathematical optimisation task that involves minimising a convex function over a convex set. A function $f: D \to \mathbb{R}$ (where D is a subset of $\mathbb{R}^m$) is *convex* if, for any two points $X_1, X_2 \in D$ and any value t between 0 and 1, $f(tX_1 + (1-t)X_2) \leq tf(X_1) + (1-t)f(X_2)$.

[51] Joseph-Louis de Lagrange (1736 - 1813), also reported as Giuseppe Luigi Lagrange or Lagrangia was an Italian mathematician and astronomer, later naturalised French.

$$\sum_{i=1}^{m} a_i y_i = 0.$$

In a linear SVM, once the optimisation problem has been solved, the weight vector W can be expressed as a linear combination of the support vectors

$$W = \sum_{X_i \in SV} a_i y_i X_i,$$

where SV is the set of support vectors. This means that only the support vectors matter. All other training points have $a_i = 0$ and do not contribute to W. The positions and labels y_i of the support vectors determine the direction of W. The magnitude of W is tied to the margin width: the smaller $\|W\| = \sqrt{\sum_i w_i^2}$, the wider the margin.

Example. Imagine you are forecasting the risk of wildfires on a given day based on two measurements:

- daily maximum temperature (°C)
- daily peak wind speed (km/h).

We normalise both measurements to the interval $[0, 2]$ by dividing the temperature measurement by 20 and the wind speed measurement by 40. Our toy dataset is

Day	Temp (°C)	Wind (km/h)	Index i	$X_i = (x_{i1}, x_{i2})$	y_i	Risk
A	0	0	1	(0, 0)	-1	low
B	40	0	2	(2, 0)	-1	low
C	0	40	3	(0, 2)	+1	high
D	40	40	4	(2, 2)	+1	high

In this simplified model, the risk of wildfires entirely depends on wind speed; temperature x_{i1} varies, but does not affect the outcome. These four data points X_i lie at the corners of a square. Visually, class A represents the bottom edge and class B the top edge, and the two classes are perfectly linearly separable.

We have to solve the following dual problem

$$max_{[a_i] \in \mathbb{R}^4} \left(\sum_{i=1}^{4} a_i - \frac{1}{2} \sum_{i,j=1}^{4} a_i a_j \, y_i y_j X_i^T X_j \right)$$

subject to $a_i \geq 0$ and

$$\sum_{i=1}^{4} a_i y_i = 0.$$

Let us compute the matrix G with entries $g_{ij} = y_i y_j X_i^T X_j$. We have

$$g_{11} = y_1 y_1 X_1^T X_1 = (-1)(-1)[0,0] \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 1 \cdot (0 \cdot 0 + 0 \cdot 0) = 0$$

$$g_{12} = y_1 y_2 X_1^T X_2 = (-1)(-1)[0,0] \begin{bmatrix} 2 \\ 0 \end{bmatrix} = 1 \cdot (0 \cdot 2 + 0 \cdot 0) = 0$$

...

$$g_{44} = y_4 y_4 X_4^T X_4 = (+1)(+1)[2,2] \begin{bmatrix} 2 \\ 2 \end{bmatrix} = 1 \cdot (2 \cdot 2 + 2 \cdot 2) = 8.$$

Consequently,

$$G = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & -4 \\ 0 & 0 & 4 & 4 \\ 0 & -4 & 4 & 8 \end{bmatrix}$$

and our objective function is

$$L(A) = \sum_{i=1}^{4} a_i - \frac{1}{2} A^T G A,$$

with $\sum_{i=1}^{4} a_i y_i = 0$, where $A^T = [a_1, a_2, a_3, a_4]$. The equality

$$\sum_{i=1}^{4} a_i y_i = -a_1 - a_2 + a_3 + a_4 = 0$$

gives $a_1 = -a_2 + a_3 + a_4 \geq 0$. Denote $a_2 = c, a_3 = d, a_4 = e$. Then $a_1 = -c + d + e$, After substituting this into $L(A)$ and expanding $\frac{1}{2}A^T GA$ we obtain

$$L(c, d, e) = 2d + 2e - (2c^2 + 2d^2 + 4e^2 - 4ce + 4de).$$

To find the maximum of L, we set its partial derivatives to zero ([52])

$$\frac{\partial L}{\partial c} = -4c + 4e = 0 \Rightarrow c = e,$$
$$\frac{\partial L}{\partial d} = 2 - 4d - 4e = 0 \Rightarrow d + e = \frac{1}{2} \Rightarrow d = \frac{1}{2} - e \geq 0,$$
$$\frac{\partial L}{\partial e} = 2 - 8e + 4c - 4d = 0 \Rightarrow 2 - 8e + 4e - 4d = 0 \Rightarrow d + e = \frac{1}{2}.$$

From the condition $-a_2 + a_3 + a_4 \geq 0$, we can deduce that

$$-c + d + e \geq 0 \Rightarrow -e + \left(\frac{1}{2} - e\right) + e \geq 0 \Rightarrow 0 \leq e \leq \frac{1}{2}.$$

Consequently,

$$a_2 = a_4 = e,\ a_1 = a_3 = \frac{1}{2} - e,\ 0 \leq e \leq \frac{1}{2}$$

and these values all yield the same maximum $L(A)$. For simplicity, let us select $e = \frac{1}{4}$. Then

$A = \left[\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right]^T$ and $W = \sum_{i=1}^{4} a_i y_i X_i = \frac{1}{4}\left(-\begin{bmatrix}0\\0\end{bmatrix} - \begin{bmatrix}2\\0\end{bmatrix} + \begin{bmatrix}0\\2\end{bmatrix} + \begin{bmatrix}2\\2\end{bmatrix}\right) = \begin{bmatrix}0\\1\end{bmatrix}$.

Using any support vector, for example $X_3 = [0, 2]^T$ and $y_3 = +1$ we get

$$b = y_3 - W^T X_3 = 1 - (0 \cdot 0 + 1 \cdot 2) = -1.$$

Once we have learned the weight vector W and the bias b, classifying any new instance X becomes straightforward.

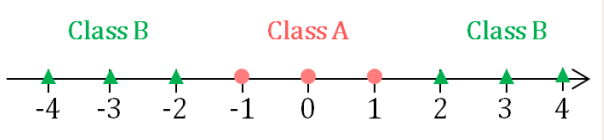
Suppose the forecast for tomorrow is: temperature = 30 °C and wind speed = 50 km/h. First, we normalise these values to obtain $X = [1.5, 1.25]^T$. Then we compute the score

$$W^T X + b = [0, 1]\begin{bmatrix}1.5\\1.25\end{bmatrix} - 1 = 1.25 - 1 = 0.25.$$

Consequently, the model's prediction is: $sign(W^T X + b) = sign(0.25) = +1$ (high risk).

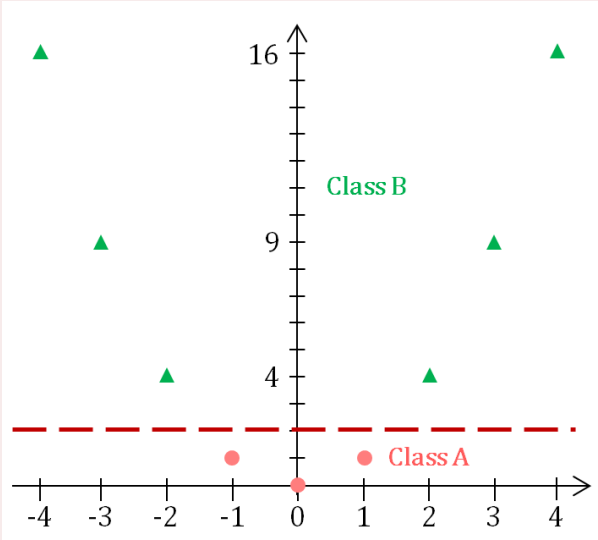
If the data is not linearly separable, one approach could be to map it into a higher-dimensional space. This transformation involves mapping the data into a new space (the feature space), where the separation of the data points becomes possible. We can then fit a decision boundary – now a higher-dimensional hyperplane – to separate the classes and make predictions.

Example. Consider a set of one-dimensional data points that cannot be linearly separated in one dimension. Note that a zero-dimensional hyperplane is simply a point ([W8]).



[52] It is easy to show that the Hessian matrix of $L(c, d, e)$ is $-G$, which is a negative semidefinite matrix. The stationary points of a twice-differentiable function subject to linear constraints, for which the Hessian matrix is negative semidefinite everywhere, are global maxima. A Hessian matrix is a square matrix containing the second-order partial derivatives of a multivariable function. See e.g. [S24] for more details on Hessian matrices.

Mapping these points into a two-dimensional space may reveal that they can be linearly separated by a one-dimensional hyperplane, i.e. a straight line. For example, let us consider the transformation $\phi(x) = x^2$ that maps a one-dimensional space onto a two-dimensional feature space: $x \to \big(x, \phi(x)\big) = (x, x^2)$. In this space, the classes A and B are linearly separable.



Of course, there are many possible functions that can map data to a higher-dimensional feature space.

Assume that the input data $(X_i, y_i). i = 1, 2, \dots, m$, is not linearly separable and we apply a feature map ϕ, such that $\phi(X) \in \mathbb{R}^M$ with $M \gg n$. Then, if the data is linearly separable in $\mathbb{R}^M$, we can solve the dual problem

$$max_{(a_i)} \left(\sum_{i=1}^{m} a_i - \frac{1}{2} \sum_{i,j=1}^{m} a_i a_j \, y_i y_j \phi(X_i)^T \phi(X_j) \right)$$

subject to

$$\sum_{i=1}^{m} a_i y_i = 0.$$

However, real-world applications often involve a large number of features. Applying transformations containing, for example, many polynomial combinations of these features can lead to extremely high computational costs. The so-called 'kernel trick' provides a solution to this problem (see [C8] and [W8]). This approach replaces the scalar products in a high-dimensional feature space with a kernel. A *kernel* is a function K that corresponds to a scalar product in some feature space $\mathbb{R}^M$ (which may be infinite-dimensional), i.e.

$$K(X_i, X_j) = \phi(X_i)^T \phi(X_j).$$

This is a significant advantage, since far less work may be required to compute $K(\cdot,\cdot)$ than to determine $\phi(\cdot)^T \phi(\cdot)$.

Common kernel functions are ([53]):

- Polynomial: $K(X_i, X_j) = \big(X_i^T X_j + c\big)^d$, where c is a constant and d is the polynomial degree.

- *Radial Basis Function* (RBF): $K(X_i, X_j) = \mathrm{e}^{-\gamma \|X_i - X_j\|^2}$, where γ is a hyperparameter and $\|X_i - X_j\|^2 = \sum_{k=1}^{m} (x_{ik} - x_{jk})^2$.

[53] Not every function qualifies as a valid kernel. According to Merces's theorem, a function can only qualify as a valid kernel if it corresponds to a scalar product in some Hilbert space and satisfies positive semi-definiteness (see [G22] and [M25]).

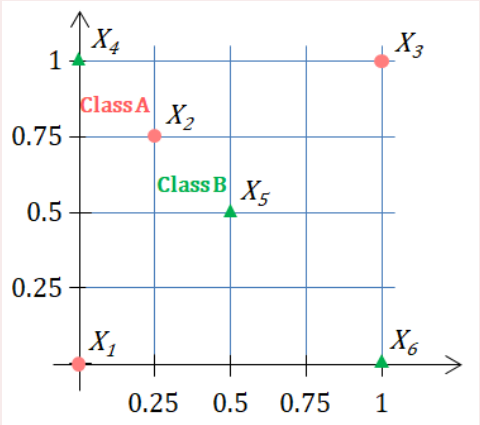
Each of these functions defines a different implicit feature space, enabling the SVM to construct decision boundaries.

Example. Consider the following two classes of points in 2D space:

$$A = \{X_1, X_2, X_3\} = \{(0,0), (0.25, 0.75), (1,1)\}$$

and

$$B = \{X_4, X_5, X_6\} = \{(0,1), (0.5, 0.5), (1,0)\}$$



These classes are not linearly separable in $\mathbb{R}^2$, since no straight line can clearly distinguish between them. However, applying the following feature map

$$\phi(X_1, X_2) = (X_1^2, X_2^2, \sqrt{2}X_1X_2, \sqrt{2}X_1, \sqrt{2}X_2, 1)$$

we can linearly separate points $\phi(X_i), i = 1, 2, \dots, 6$, in the feature space $\mathbb{R}^6$ by solving the dual problem. The feature map ϕ corresponds to the following second-degree polynomial kernel

$$K(X_i, X_j) = (X_i^T X_j + 1)^2.$$

Indeed, it is easy to show that $K(X_i, X_j) = \phi(X_i)^T \phi(X_j)$.

The essence of the kernel trick is that you operate in a high-dimensional space without ever explicitly departing from the original input space. Although the SVM finds a linear separator in $\mathbb{R}^6$, it is interpreted as a nonlinear boundary in $\mathbb{R}^2$.

Remark. It turns out that any two classes of points that are not linearly separable in their original space can be made linearly separable in a higher-dimensional space (possibly infinite-dimensional). This is the foundation of the kernel trick in SVMs. However, the reader may ask what the smallest dimension of a feature space is in which a given two non-separable classes can be separated. The answer is that there is no universal minimum dimension of the feature space needed for separability. The required dimension depends on the number of points, how they are distributed, and how complex the boundary between classes is.

Support vector machines are recognised for their ability to handle high dimensional data and their effectiveness in a variety of applications. For instance:

- **Image recognition**. SVMs are frequently employed in image classification tasks, including facial recognition and handwriting recognition.

- **Bioinformatics**. SVMs are applied in gene classification and protein function prediction.

- **Text classification**. SVMs are employed to categorize emails as either spam or non-spam, or to classify documents by subject.

- **Medical image classification**. One particular application of SVMs is in the classification of different types of cancer on the basis of genomic data. They can help identify cancer subtypes, which is crucial for personalized treatment plans.

- **Finance**. SVMs can be used to predict stock prices or to assess credit risk.

The main drawbacks of support vector machines are outlined below, covering computational issues, data sensitivities, model complexity and practical obstacles ([P11]).

× **Computational complexity and scalability**. Training a SVM involves solving a quadratic optimisation problem, and the time and memory requirements for this problem grow at least quadratically with the number of samples. This makes SVMs impractical for very large datasets. Furthermore, using non-linear kernel functions increases the computational cost, resulting in lengthy training times that hinder rapid prototyping and iteration.

× **Model selection and hyperparameter tuning**. Selecting the appropriate kernel function and tuning hyperparameters requires extensive cross-validation and domain expertise. Poor choices can lead to underfitting or overfitting. There is no one-size-fits-all guideline; kernel and parameter settings that work well for one problem may be ineffective for another, meaning that the tuning process can be time-consuming and computationally expensive.

× **Interpretability and practical limitations**. Non-linear SVMs operate in an implicit, often high-dimensional feature space, which makes the resulting decision boundary difficult to interpret or translate into rules that humans can understand. Standard SVMs do not natively produce probability estimates. Calibrated probabilities require additional techniques, adding complexity to the workflow.

3.2.6. K-Nearest Neighbours (KNN)

K-Nearest Neighbours (KNN) is a simple yet effective algorithm employed in the field of AI for the purposes of classification and regression. It is an instance-based learning algorithm, meaning it makes decisions based on the closest training examples in the feature space. The classification of a data point is determined by the majority class among its 'k' nearest neighbours. KNN is a versatile algorithm that is easy to implement and understand. It's particularly useful when you have a well-labelled dataset and need a straightforward approach to classification or regression.

The following explains KNN step by step:

1. **Choose the number of neighbours (k)**. Select the number of nearest neighbours k to consider for making the classification or regression decision. Common choices are $k = 3, k = 5$, or $k = 7$.

2. **Calculate the distance**. For a given data point, calculate the distance between it and all other points in the training set. Common distance metrics include Euclidean distance, Manhattan distance, and Minkowski distance ([M26]). The Euclidean distance is given by:

$$d(X, Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2},$$

where $X = [x_1, x_2, \dots, x_n]$ and $Y = [y_1, y_2, \dots, y_n]$.

The Manhattan distance is defined as the sum of the absolute differences in the coordinates of two points:

$$d_{Manhattan}(X, Y) = \sum_{i=1}^{n}|x_i - y_i|.$$

This metric is also known as the 'taxicab' or 'city-block' distance, as it mimics movement on a grid where travel is only possible along road axes.

The Minkowski distance ([54]) generalises several distance measures and allows for a tuneable parameter, $p \geq 1$. It is defined as follows:

$$d_{Minkowski}(X, Y) = (\sum_{i=1}^{n}|x_i - y_i|^p)^{1/p}.$$

3. **Identify the nearest neighbours**. Find the k data points in the training set that are closest to the given data point based on the chosen distance metric.

---

[54] This should not be confused with the pseudo-Euclidean metric of Minkowski space.

4. **Make the decision.**

(a) *Classification*. Assign the class that is most frequent among the k nearest neighbours (majority vote), or

(b) *Regression*. Calculate the average (or weighted average) of the values of the k nearest neighbours.

The following simple example demonstrates how KNN can be used for classification based on the proximity of data points.

Example. Imagine you have just moved to a new neighbourhood and want to know what type of fruit trees are planted in your neighbours' gardens. You have three types of fruit trees to identify: apples, oranges, and cherries. You've already identified the type of fruit tree in the gardens of some of your neighbours. Now, you come across a new tree that you haven't seen before and want to classify it using KNN.

You have the following data (training set) about the fruit trees in your neighbours' gardens

Tree	Size (cm)	Colour (0-255 scale)	Type
1	200	100	Apple
2	150	120	Apple
3	180	115	Apple
4	220	190	Orange
5	210	200	Orange
6	230	195	Orange
7	160	80	Cherry
8	170	85	Cherry
9	175	90	Cherry

The new tree you want to classify has the following characteristics

- Size: 205 cm

- Colour: 180

The following step-by-step procedure outlines the process of KNN classification:

1. *Choose the number of neighbours* (k). Let us choose $k = 3$, meaning we will consider the 3 nearest neighbours.

2. *Calculate the distance.* Use Euclidean distance to calculate the distance between the new tree and each of the known trees. For simplicity, we will only show a couple of distance calculations:

- Distance to Tree 1:

$$\sqrt{(205-200)^2+(180-100)^2} \approx 80.2$$

- Distance to Tree 4:

$$\sqrt{(205-220)^2+(180-190)^2} \approx 18.0$$

3. *Identify the nearest neighbours.* After calculating the distances to all known trees, identify the three nearest neighbours. Suppose they are

- Tree 4 (Orange)
- Tree 5 (Orange)
- Tree 6 (Orange)

4. *Make the decision.* Since all three nearest neighbours ($k = 3$) are Orange, the new tree is classified as an Orange tree.

5. *Conclusion.* The KNN algorithm suggests that the new tree is most likely an orange tree, as the majority of its three nearest neighbours are also orange trees.

KNN has a variety of applications. For instance

- **Image recognition**. KNN is used to classify images based on the pixel intensity values. For example, it can recognize handwritten digits or classify objects in images.

- **Recommender systems**. KNN s used in recommendation systems to suggest items (e.g. films or products) based on users' preferences. It identifies users with similar tastes and advises them on items they might like.

- **Medical diagnosis**. KNN is used in healthcare to classify patients based on their symptoms and medical history. It can help in diagnosing diseases by comparing a patient's profile to those of previous patients.

- **Text classification**. KNN can classify documents based on their content. For example, it can be used to categorize emails as spam or non-spam, or to classify news articles by topic.

- **Anomaly detection**. KNN can detect anomalies or outliers in data by identifying points that are significantly different from their neighbours.

The advantage of KNN is that it is simple and intuitive to understand and implement. It is also effective on small to medium-sized data sets. The algorithm is non-parametric, i.e. it makes no assumptions about the underlying data distribution.

The main limitations of KNN are as follows ([L14], [Z10]):

× **Curse of dimensionality**. In high-dimensional feature spaces, the distances between points tend to become almost uniform, which erodes the contrast between 'near' and 'far'. This reduces the effectiveness of KNN's distance-based voting and often causes poor generalisation when many irrelevant or redundant features are present.

× **Sensitivity to feature scaling and irrelevant features**. As KNN relies on raw distance calculations, features with larger numerical ranges can dominate the metric, while irrelevant features can reduce the influence of important ones. Without careful preprocessing, such as normalisation, standardisation or feature selection, KNN's performance can degrade significantly.

× **Memory intensity**. KNN is a 'lazy learner': it stores the entire training set in memory for use during the prediction process. For large or high-dimensional datasets, this can result in excessive memory usage, rendering KNN impractical when system resources are limited.

× **Difficulty incorporating domain knowledge**. As KNN does not learn feature weights during training, it is difficult to incorporate prior knowledge or feature importance. Although extensions exist (e.g. distance-weighted voting and metric learning), they increase complexity and often necessitate further parameter tuning.

3.2.7. K-Means clustering

K-Means Clustering is a popular unsupervised machine learning algorithm used to partition a dataset into distinct groups or clusters. It is a versatile and widely used algorithm for identifying patterns and grouping similar data points

K-means clustering aims to group similar data points together into clusters. Each cluster is represented by its centroid (center), and the goal is to minimize the distance between data points and their corresponding centroids.

Step-by-step explanation:

1. **Choose the number of clusters** (k). Decide on the number of clusters k you want to form. This can be based on prior knowledge or determined using methods like the elbow method ([55]).

2. **Initialize centroids**. Randomly select k data points as initial centroids. These centroids will serve as the starting points for the clusters.

3. **Assign data points to clusters**. For each data point, calculate the distance to each centroid and assign the data point to the cluster with the nearest centroid**.**

4. **Update centroids.** Calculate the new centroid of each cluster by taking the mean of all data points assigned to that cluster**.**

5. **Iterate.** Repeat the assignment and update steps until the centroids no longer change significantly or a maximum number of iterations is reached**.**

6. **Result.** The final clusters represent groups of similar data points, with each cluster having its own centroid.

Example. This example demonstrates how K-means clustering can help gain insights into customer behaviour and improve their marketing strategies.

Scenario. A retail store wants to segment its customers into distinct groups based on their purchasing behaviour. The store has collected data on customers' annual spending and frequency of visits.

The subsequent breakdown provides an overview of the process:

1. *Collect data*. The store collects data on 10 customers, including their annual spending (in euros) and the number of visits per year.

Customer	Annual Spending (€)	Visits per Year
1	500	10
2	1500	15
3	250	5
4	2000	20
5	300	7
6	4500	30
7	1000	12
8	3500	25
9	750	8
10	4000	28

2. *Choose the number of clusters* (k). The store decides to segment its customers into 3 clusters ($k = 3$).

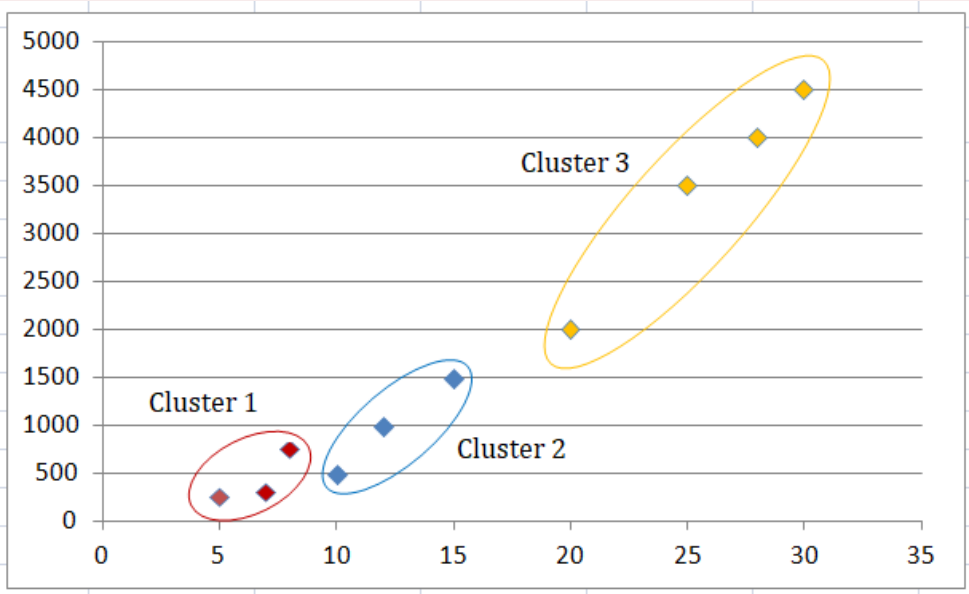
3. – 6. Select three customers at random as initial centroids. Then calculate the distance between each customer and each centroid, and assign each customer to the nearest centroid. Next, calculate the new centroid of each cluster by taking the mean of all the customers assigned to it. Repeat the assignment and update steps until the centroids no longer change significantly.

7. *Result*. After running the K-Means Clustering algorithm, the store identifies 3 distinct customer segments:

[55] The elbow method is a technique used to determine the optimal number of clusters (k) in K-Means Clustering. It helps identify the point where increasing the number of clusters results in diminishing returns in terms of improved clustering performance.

- Cluster 1: Low spenders with few visits (Customers 3, 5, 9)
- Cluster 2: Medium spenders with moderate visits (Customers 1, 2, 7)
- Cluster 3: High spenders with frequent visits (Customers 4, 6, 8, 10)

Here is a simple visualization of the customer segments:



Based on these customer segments, the store can:

- Tailor marketing campaigns to target each segment specifically
- Offer personalized promotions to high spenders with frequent visits
- Develop loyalty programs to increase visits from low spenders.

The K-means clustering method has a multiplicity of applications. For example:

- **Customer segmentation**. Businesses use K-means clustering to segment customers based on purchasing behaviour, demographics, or preferences. This helps in tailoring marketing strategies and improving customer satisfaction.

- **Image compression**. K-means clustering is applied in image compression to reduce the number of colours in an image. The algorithm clusters similar colours and replaces them with the centroid colour, reducing the file size.

- **Document clustering**. In text mining and natural language processing, K-means clustering groups similar documents based on content. This helps in organizing large collections of documents and improving search and retrieval.

- **Anomaly detection**. K-means clustering can detect anomalies or outliers in data by identifying points that do not fit well into any cluster. This is useful in fraud detection, network security, and quality control.

- **Genomic data analysis**. Researchers use K-means clustering to group genes with similar expression patterns, aiding in the understanding of biological processes and disease mechanisms.

Although K-means is useful and efficient in many machine learning contexts, it does have some distinct limitations ([T15]).

× **Need to predefine** k. You must select the number of clusters k before running the algorithm. Picking the wrong value of k can lead to poor clustering: too small merges distinct groups, while too large splits natural clusters.

× **Sensitivity to initialisation**. K-means begins with randomly selected centroids. Using different initial seeds can result in different final clusters (local optima).

× **Sensitivity to outliers**. Outliers can pull centroids towards them, which distorts the boundaries of clusters. They may even form their own 'singleton' clusters.

× **Scalability challenges**. Although K-means is computationally efficient for moderate data sizes, very large datasets require optimisations, such as mini-batch K-means and distributed computing.

In short, K-means is fast, simple and widely used, but it is not a one-size-fits-all solution. It works best with well-separated, similarly sized clusters of numeric data with low to moderate dimensions, and careful preprocessing and parameter tuning are required for optimal performance.

3.2.8. Naive Bayes

The *Naive Bayes* algorithm is a family of simple and effective probabilistic classifiers based on Bayes' Theorem. It is a versatile algorithm that performs well on many types of problems. The 'naive' assumption in naive Bayes is that the features (or predictors) are conditionally independent given the class label ([56]). This assumption of independence simplifies the computation and makes the algorithm fast and scalable, even with large datasets.

However, in text classification, the assumption of independence means that the presence of one word does not affect the presence of another word in the document, which may not always be true. This assumption cannot be completely removed in naive Bayes. However, you can manage its impact in the following ways:

- Use naive Bayes for tasks where the impact of feature independence is minimal, such as spam filtering or document categorisation,

- For more complex dependencies, consider using alternative algorithms such as logistic regression or decision trees.

Bayes' Theorem provides the foundation for naive Bayes classifiers. It describes the probability of an event based on prior knowledge of conditions that might be related to the event:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

where

- $P(A|B)$ is the posterior probability of class A given feature B

- $P(B|A)$ is the likelihood of feature B given class A

- $P(A) > 0$ is the prior probability of class A

- $P(B) > 0$ is the prior probability of feature B.

The following types of naive Bayes classifiers are considered:

- **Gaussian naive Bayes**. This approach is based on the assumption that continuous features follow a normal (Gaussian) distribution.

- **Multinomial Naive Bayes (MNB).** Typically used for text classification problems, where features represent the frequency of words.

- **Bernoulli Naive Bayes**. Used for binary/boolean features, where each feature represents the presence or absence of a particular attribute.

---

[56] Conditional independence means that once you know the class label y, each feature x_i provides no additional information about any other feature x_j. Formally, for features $x_1, x_2, \dots, x_n$ and class y: $P(x_1, x_2, \dots, x_n \mid y) = \prod_{i=1}^{n} P(x_i \mid y) = P(x_1 \mid y) \cdot P(x_2 \mid y) \cdot \dots \cdot P(x_n \mid y)$ ([M27]).

The naive Bayes algorithm is executed in two steps:

1. **Training:**

- Calculate the prior probabilities ([57]) for each class based on the training data
- Calculate the likelihood of each feature given each class
- Store these probabilities for later use during prediction.

2. **Prediction:**

- For a new data point, calculate the posterior probability for each class using Bayes' Theorem.
- Assign the class with the highest posterior probability to the new data point.

Example. Suppose you own an online store and you want to categorise customer reviews in 'Positive' and 'Negative'. We will use the MNB algorithm as probabilistic learning method to do this.

Step-by-Step breakdown:

1. *Collect data*. Assume that you already have a dataset (training data) of customer reviews. In this step, you assign a 'Positive' or 'Negative' label to each review in the dataset.

#	Review	Sentiment
R1	"I love the product, it's amazing!"	Positive
R2	"Terrible service, very disappointed."	Negative
R3	"Great quality, will buy again!"	Positive
R4	"Not worth the money, very bad product and experience."	Negative
R5	"Excellent product and service, highly recommend."	Positive

2. *Preprocess data*. Convert the text reviews into a numerical format. This can be done by creating a vocabulary of unique words and representing each review as a vector of word frequencies (Bag-of-Words – see Section 5.1.3.1) or binary values (presence/absence of words).

In this example there are 24 words in the vocabulary:

$V = \{w_1$:"I", w_2:"love", w_3:"the", w_4:"product", w_5:"it's", w_6:"amazing", w_7:"terrible", w_8:"service", w_9:"very", w_{10}:"disappointed", w_{11}:"great", w_{12}:"quality", w_{13}:"will", w_{14}:"buy", w_{15}:"again", w_{16}:"not", w_{17}:"worth", w_{18}:"money", w_{19}:"bad", w_{20}:"and", w_{21}:"experience", w_{22}:"excellent", w_{23}:"highly", w_{24}:"recommend"$\}$

To simplify the software implementation of the algorithm, for each review one can create a binary vector indicating the presence (1) or absence (0) of each word in the vocabulary. Each position in the vector corresponds to a word in the vocabulary. These vectors are rows in the following matrix:

[57] The term 'probability' (whatever it means) is actually a misnomer in this context. Here we are dealing with statistical frequencies, not probabilities. However, in many practical situations, we do not have theoretical models that can accurately describe the probabilities of events within these models. Instead, we rely on observed data and use frequencies rather than probabilities. It is important to realise, though, that when using relative frequencies to estimate probabilities, these estimates may differ considerably from (unknown) probabilities. For example, relative frequencies are generally not σ-additive functions. Therefore, from a mathematical point of view, they are not probabilities in the standard Kolmogorovian model (see e.g. [F1] for a detailed discussion of this issue).
In this paper, I will adopt the imprecise terminology commonly used in AI literature and refer to frequencies as 'probabilities'.

No.	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}	w_{21}	w_{22}	w_{23}	w_{24}	Sent.	Σw_k
R1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Pos.	6
R2	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Neg.	4
R3	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	Pos.	5
R4	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	Neg.	9
R5	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	Pos.	6
$\Sigma_{Pos.}$	1	1	1	2	1	1	0	1	0	0	1	1	1	1	1	0	0	0	0	1	0	1	1	1	Σw_k	17
$\Sigma_{Neg.}$	0	0	1	1	0	0	1	1	2	1	0	0	0	0	0	1	1	1	1	1	1	0	0	0	Σw_k	13
Σ	1	1	2	3	1	1	1	2	2	1	1	1	1	1	1	1	1	1	1	2	1	1	1	1	Σw_k	30

From this matrix we have the following figures:

- total number of reviews $= 5$
- number of reviews with sentiment 'Positive' $= 3$
- number of reviews with sentiment 'Negative' $= 2$
- total number of words in reviews with sentiment 'Positive' $= 17$
- total number of words in reviews with sentiment 'Negative' $= 13$
- total number of words $= 30$
- number of times word w_k appears in reviews with 'Positive'
- number of times word w_k appears in reviews with 'Negative'.

3. *Calculate probabilities*. Using these numbers, calculate the prior probabilities.

(a) Calculate the probability for each sentiment (positive and negative):

$$P(sentiment_i) = \frac{number\ of\ reviews\ with\ sentiment_i}{total\ number\ of\ reviews}$$

- prior probability of 'Positive' sentiment: $P(Positive) = \frac{3}{5} = 0.6$
- prior probability of 'Negative' sentiment: $P(Negative) = \frac{2}{5} = 0.4$

(b) Calculate the probabilities of each word in vocabulary V given the sentiment. But instead of the standard Bayes formula

$$P(w_k|\ sentiment_i) = \frac{number\ of\ times\ w_k\ appears\ in\ reviews\ with\ sentiment_i}{total\ number\ of\ words\ in\ reviews\ with\ sentiment_i}$$

we use *Laplace* ([58]) *Smoothing* in order to handle zero probabilities:

$$P_L(w_k|\ sentiment_i) = \frac{number\ of\ times\ w_k\ appears\ in\ reviews\ with\ sentiment_i + 1}{total\ number\ of\ words\ in\ reviews\ with\ sentiment_i + card(V)}$$

where $card(V)$ is the cardinality of the set V (number of unique words, here $card(V) = 24$).

Why Laplace smoothing? One reason for this is that in text classification, it is common to encounter words in the test data that were not seen in the training data. Without smoothing, the probability of these unseen words would be zero, and when calculating the posterior probability for a class, the entire term would be zero, leading to incorrect classification.

With Laplace smoothing, the probabilities $P_L(w_k|\ sentiment_i)$ of each word in vocabulary V given the sentiment are:

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}	w_{21}	w_{22}	w_{23}	w_{24}
Pos.	0.049	0.049	0.049	0.073	0.049	0.049	0.024	0.049	0.024	0.024	0.049	0.049	0.049	0.049	0.049	0.024	0.024	0.024	0.024	0.049	0.024	0.049	0.049	0.049
Neg.	0.027	0.027	0.054	0.054	0.027	0.027	0.054	0.054	0.081	0.054	0.027	0.027	0.027	0.027	0.027	0.054	0.054	0.054	0.054	0.054	0.054	0.027	0.027	0.027

4. *Make predictions*. For a new review

[58] Pierre-Simon, Marquis de Laplace (1749 – 1827) was a French mathematician, astronomer, physicist, and all-round scientific polymath whose work helped lay the foundations for much of modern science.

R_{new} = "Great product, excellent service"

convert it into a numerical vector using the same vocabulary:

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}	w_{21}	w_{22}	w_{23}	w_{24}
R_{new}	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0

Calculate the probability of a new review for each sentiment using the formula:

$$P(R_{new}|sentiment_i) = \prod_{k=1}^{card(V)}[P_L(w_k|\ sentiment_i)]^{c_k}$$

where c_k is the number of times the word w_k appears in the new review. We have

$$P(R_{new}|Positive) = 0.073^1 \cdot 0.049^1 \cdot 0.049^1 \cdot 0.049^1 = 0.00000859$$

$$P(R_{new}|Negative) = 0.054^1 \cdot 0.054^1 \cdot 0.027^1 \cdot 0.027^1 = 0.00000213$$

Now, using Bayes' theorem, we can calculate the posterior probabilities for each of the sentiments by applying the formula:

$$P(sentiment_i|R_{new}) = \frac{P(R_{new}|sentiment_i)\cdot P(sentiment_i)}{P(R_{new})}.$$

Since

$$P(R_{new}) = P(R_{new}|Positive) \cdot P(Positive) + P(R_{new}|Negative) \cdot P(Negative)$$
$$= 0.00000859 \cdot 0{,}6 + 0.00000213 \cdot 0.4 = 0.00000601$$

we get

$$P(Positive|R_{new}) = \frac{0.00000859 \cdot\ 0.6}{0.00000601} = 0.86$$

$$P(Negative|R_{new}) = \frac{0.00000213 \cdot\ 0.4}{0.00000601} = 0.14$$

Consequently, the new review is given a 'positive' sentiment because

$$P(Positive|R_{new}) > P(Negative|R_{new}).$$

This step-by-step example demonstrates how the Naive Bayes algorithm can be used for sentiment analysis in customer reviews, providing valuable insights into customer opinions.

The naive Bayes algorithm has a variety of applications. For instance:

- **Spam detection**. Classifying emails as spam or not spam
- **Sentiment analysis**. Determining the sentiment of a text (positive, negative, neutral)
- **Document classification**. Categorizing documents into predefined categories
- **Medical diagnosis**. Predicting the likelihood of diseases based on symptoms and patient history.

Although Naive Bayes is prized for its simplicity and speed, it is subject to several assumptions and practical pitfalls that can affect its performance in real-world tasks.

✕ **Poor probability calibration**. The probabilities produced by Naive Bayes are often poorly calibrated. Even if it ranks classes correctly, the raw posterior values can be misleading for downstream tasks that rely on accurate confidence scores.

✕ **Handling continuous features**. Naive Bayes requires continuous variables to be discretised or assumed to follow a distribution (e.g. Gaussian). Choosing the wrong distribution can introduce artefacts and reduce accuracy.

✕ **Sensitivity to irrelevant features and noise**. Since each feature contributes independently, irrelevant or noisy features can skew class posterior odds unless they are carefully filtered or weighted.

× **Data Imbalance and rare classes**. When classes are highly imbalanced, the likelihood of rare classes may be underestimated, leading to systematic misclassification unless the prior probabilities are adjusted explicitly.

3.2.9. Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a widely used algorithm in AI for reducing the dimensionality of datasets while retaining key information. PCA transforms a high-dimensional dataset into a lower-dimensional space by identifying the directions (called *principal components*) along which the data varies the most. These components are orthogonal (uncorrelated) to each other and ranked by the amount of variance they capture. Orthogonal components ensure that each measures unique variance without redundancy.

PCA is especially useful in scenarios where datasets have many correlated variables or when computational efficiency is critical.

Step-by-step process:

1. **Standardisation**. Standardisation of the data is a prerequisite for performing PCA. Prior to this analysis, the data is typically standardised (i.e., each feature is scaled to have a mean of 0 and a standard deviation of 1). This procedure is undertaken to ensure that features with differing units of measurement do not exert an undue influence on the results.

2. **Covariance matrix.** Compute the covariance matrix to understand the relationships between variables.

3. **Eigenvectors and eigenvalues.** Determine the eigenvectors (directions) and eigenvalues (amount of variance) from the covariance matrix.

4. **Select components.** Retain only the top k principal components that explain the most variance, reducing the data's dimensionality.

5. **Project data.** Transform the original data onto the new subspace defined by the selected components.

The mathematical basis of PCA is rooted in linear algebra, encompassing the theory of linear operators, their respective eigenvectors, and the corresponding eigenvalues. Here is a brief explanation (see [C1] and [Y1] for more details):

- **Covariance matrix**. A covariance matrix is a square matrix used to describe the covariance values between variables. It is particularly important in fields such as data science, machine learning and finance, where understanding the relationships between multiple variables is crucial, and it is useful in stochastic modelling and principal component analysis.

Covariance measures how two variables move together. It gives insight into whether the variables have a positive, negative, or no relationship.

- *Positive covariance.* An increase in one variable is accompanied by an increase in the other.
- *Negative covariance.* When one variable increases, the other variable decreases.
- *Zero covariance.* In the case of zero covariance, it can be deduced that the two variables are unrelated, and that a change in one does not necessarily indicate an associated change in the other.

Assume that we have a dataset $D = \{X_i : i = 1, 2, \dots, m\}$ representing m features and containing n values for each feature: $X_i = \left[x_{ij}\right]_{j=1}^{n}$. Consequently, we are able to express D as the following $m \times n$ matrix

$$D = \begin{bmatrix} X_1 \\ X_2 \\ \dots \\ X_m \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ x_{21} & \dots & x_{2n} \\ \dots & \dots & \dots \\ x_{m1} & \dots & x_{mn} \end{bmatrix}$$

PCA starts with the computation of the covariance matrix $\Sigma = [\sigma_{kl}]_{k,l=1}^{m}$ of the dataset ([59]):

$$\sigma_{kl} = \frac{1}{n}\sum_{j=1}^{n}\left(x_{kj} - \bar{x}_k\right)\left(x_{lj} - \bar{x}_l\right)$$

where

- x_{kj} is the value of the k-th feature for the j-th data point,
- $\bar{x}_k$ is the mean value of the k-th feature: $\bar{x}_k = \frac{1}{n}\sum_{j=1}^{n} x_{kj}$,
- σ_{kl} is the covariance between features k and l.

Notice that σ_{kk} is just the variance of X_k: $\sigma_k^2 = \sigma_{kk} = \frac{1}{n}\sum_{j=1}^{n}(x_{kj} - \bar{x}_k)^2$.

The $m \times m$ covariance matrix is given by

$$\Sigma = \frac{1}{n} D_{\mathrm{s}} D_{\mathrm{s}}^{\mathrm{T}}$$

where D_{s} is the matrix of the standardised ([60]) dataset and $D_{\mathrm{s}}^{\mathrm{T}}$ is its transpose:

$$D_{\mathrm{s}} = \begin{bmatrix} (x_{11} - \bar{x}_1)/\sigma_1 & \cdots & (x_{1n} - \bar{x}_1)/\sigma_1 \\ (x_{21} - \bar{x}_2)/\sigma_2 & \dots & (x_{2n} - \bar{x}_2)/\sigma_2 \\ \dots & \dots & \dots \\ (x_{m1} - \bar{x}_m)/\sigma_m & \dots & (x_{mn} - \bar{x}_m)/\sigma_m \end{bmatrix}, \quad D_{\mathrm{s}}^{\mathrm{T}} = \begin{bmatrix} (x_{11} - \bar{x}_1)/\sigma_1 & \cdots & (x_{m1} - \bar{x}_m)/\sigma_m \\ (x_{12} - \bar{x}_1)/\sigma_1 & \dots & (x_{m2} - \bar{x}_m)/\sigma_m \\ \dots & \dots & \dots \\ (x_{1n} - \bar{x}_1)/\sigma_1 & \dots & (x_{mn} - \bar{x}_m)/\sigma_m \end{bmatrix}.$$

- **Eigenvectors and eigenvalues**. In order to compute eigenvalues and the corresponding eigenvectors of Σ, it is necessary to solve the following eigenproblem: $\Sigma E = \lambda E$, where $\lambda \in \mathbb{R}$ is an eigenvalue and E an eigenvector. The eigenvectors of Σ represent the directions of maximum variance, i.e. the principal components, while the eigenvalues correspond to the magnitude of variance explained by each principal component.

- **Projection**. The centred dataset is projected onto a new subspace defined by the top l principal components with the largest eigenvalues, effectively reducing the dimensionality while preserving the most important information. The first principal component corresponds to the direction of the largest variance in the data, and each subsequent component is orthogonal to the previous ones, thus capturing as much of the remaining variance as possible.

The projection of data onto a lower-dimensional subspace using the centred data matrix involves a mathematical operation. Let us explain this in more detail. Assume that V^l is the $m \times l$ matrix of the top l ($l < m$) eigenvectors (principal components) corresponding to the largest l eigenvalues of the covariance matrix Σ. Then the projection of D_{s} onto the l-dimensional subspace is given by:

$$D_{Projected}^{T} = D_{\mathrm{s}}^{\mathrm{T}} \mathrm{V}^l$$

where $D_{Projected}^{T}$ is the transpose of the matrix $D_{Projected}$.

---

[59] In PCA, the dataset being analysed is often treated as the entire set of interest (a 'population'). Since population variance does not require a correction for bias, dividing by n (and not by $n-1$) is sufficient. If the dataset is a sample of the population, $1/(n-1)$ should be used.

[60] Centring the data is always required in PCA. This involves shifting each feature so that its mean is zero: $x_{kj} - \bar{x}_k$. This is because PCA is based on the covariance matrix, and covariance is defined relative to the mean. Without centring, the first principal component would often point towards the mean rather than the direction of maximum variance. Standardisation (scaling to unit variance) is optional but often recommended. If features are measured in different units (e.g. centimetres versus kilograms, or dollars versus percentages), the feature with the larger numerical scale will dominate the covariance matrix. Standardisation puts all features on an equal footing.

Example. This example illustrates the application of PCA to feature reduction. For the purposes of simplicity and comprehension, a small 2D dataset will be transformed into a 1D dataset.

Suppose we have the following dataset of two features, x_1 (height in cm) and x_2 (weight in kg):

Sample	x_1 (Height)	x_2 (Weight)
1	160	55
2	170	65
3	180	75
4	190	85

Step-by-Step PCA computation:

1. *Standardise the dataset*. First, we standardise the data (convert it to zero mean and unit variance) so that each feature contributes equally. Our dataset is represented by the following 2×4 matrix

$$D = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 160 & 170 & 180 & 190 \\ 55 & 65 & 75 & 85 \end{bmatrix}$$

- Compute the mean and standard deviation for each feature:

$$\bar{x}_1 = \frac{160+170+180+190}{4} = 175, \quad \bar{x}_2 = \frac{55+65+75+85}{4} = 70$$

$$\sigma_1 = \sigma_2 = \sqrt{\frac{225+25+25+225}{3}} = 12.91$$

- The standardized dataset is then

$$D_s = \begin{bmatrix} X_1^s \\ X_2^s \end{bmatrix} = \begin{bmatrix} 1.35 & 0.15 & 0.15 & 1.35 \\ 1.35 & 0.15 & 0.15 & 1.35 \end{bmatrix}.$$

2. *Compute the covariance matrix.*

$$\Sigma = \frac{1}{n-1} D_s D_s^T = \frac{1}{3}\begin{bmatrix} 1.35 & 0.15 & 0.15 & 1.35 \\ 1.35 & 0.15 & 0.15 & 1.35 \end{bmatrix}\begin{bmatrix} 1.35 & 1.35 \\ 0.35 & 0.35 \\ 0.35 & 0.35 \\ 1.35 & 1.35 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 3.69 & 3.69 \\ 3.69 & 3.69 \end{bmatrix} = \begin{bmatrix} 1.23 & 1.23 \\ 1.23 & 1.23 \end{bmatrix}.$$

Notice that we take $1/(n-1)$ instead of $1/n$, i.e. we apply Bessel's correction, because our dataset is a sample rather than the entire population.

3. *Perform eigendecomposition*. To find eigenvalues λ and eigenvectors E, solve the following characteristic equation for λ

$$\det(\Sigma - \lambda I) = \det\left(\begin{bmatrix} 1.23 & 1.23 \\ 1.23 & 1.23 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) = 0,$$

where $\det(\Sigma - \lambda I)$ denotes the determinant ([T16]) of the matrix $\Sigma - \lambda I$. The equation

$$\det\left(\begin{bmatrix} 1.23 - \lambda & 1.23 \\ 1.23 & 1.23 - \lambda \end{bmatrix}\right) = 0$$

amounts to

$$(1.23 - \lambda)^2 - 1.23^2 = 0$$

so we obtain two eigenvalues: $\lambda_1 = 2.46$ and $\lambda_2 = 0$.

Now calculate the eigenvectors E_1 and E_2, which are solutions of the equations

$$(\Sigma - \lambda_i I)V = 0, i = 1, 2.$$

For $\lambda_1 = 2.46$ we have to solve

$$\begin{bmatrix} -1.23 & 1.23 \\ 1.23 & -1.23 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0.$$

This equation has infinitely many solutions $= \begin{bmatrix} a \\ a \end{bmatrix}$, for example $E = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. We chose

$$E_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

The underlying reason for this is as follows: Eigenvectors define directions of maximum variance. For $\lambda_1 = 2.46$, the direction $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ reflects equal contribution from both features (height and weight in our example). It is a meaningful and intuitive choice for interpreting the data. Moreover, eigenvectors are commonly normalized to simplify their use in further computations, especially in PCA. The vector $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is normalized by dividing by its magnitude:

$$\left\| \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\| = \sqrt{1^2 + 1^2} = \sqrt{2}, \;\; E_1 = \tfrac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

For $\lambda_2 = 0$ the eigenvector is

$$E_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

4. *Select principal components.* Take the eigenvector corresponding to the largest eigenvalue $\lambda_1 = 2.46$:

$$E_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

5. *Project the data*. Project the standardised data onto the selected principal component E_1:

$$D^T_{Projected} = D_s^{\mathrm{T}} E_1 = \tfrac{1}{\sqrt{2}} \begin{bmatrix} 1.35 & 1.35 \\ 0.35 & 0.35 \\ 0.35 & 0.35 \\ 1.35 & 1.35 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \tfrac{1}{\sqrt{2}} \begin{bmatrix} 2.7 \\ 0.7 \\ 0.7 \\ 2.7 \end{bmatrix}.$$

6. *Final result*. The original 2D data has been reduced to a single dimension (1D):

$$D_{Projected} = \tfrac{1}{\sqrt{2}} [2.7 \quad 0.7 \quad 0.7 \quad 2.7]^T.$$

This 1D dataset retains most of the variance from the original dataset.

The following are examples of how PCA is used in AI:

- **Feature reduction**. This process entails the simplification of datasets for machine learning models, a process which must be executed with a view to retaining all essential information.

- **Visualisation**. The representation of high-dimensional data in 2D or 3D is a technique employed to enhance comprehension.

- **Noise reduction**. This process involves the removal of less informative features that contribute to the presence of noise in the data.

PCA is arguably the most widely used technique for reducing the dimensionality of data. However, there are some limitations of PCA with which one should be familiar ([P12]).

× **Assumption of linearity**. PCA only captures linear relationships, so it may fail to represent complex nonlinear structures without kernel extensions or alternative methods.

× **Vulnerability to outliers**. Extreme values can influence component directions disproportionately, potentially resulting in misleading projections.

× **Interpretability of components**. Principal components are linear combinations of all the original features, which makes it difficult to assign a clear meaning, especially when many variables contribute similarly to a component.

× **Orthogonality constraint**. Enforcing orthogonal components may not reflect the true underlying structure.

× **Computational overhead**. Eigen-decomposition of large covariance or correlation matrices increases in complexity as the number of features grows, making it challenging to analyse high-dimensional data.

× **Variance vs. structure mismatch**. PCA prioritises directions of maximal variance, which does not always correspond to meaningful patterns or clusters in the data.

3.2.10. Gradient Descent (GD)

Gradient Descent ([61]) is an optimisation algorithm widely used in AI and machine learning to minimize a cost or loss function, enabling models to learn and improve. It ensures that the model learns by iteratively reducing the error between predictions and actual values.

GD adjusts the model's parameters (like weights and biases in neural networks) iteratively to minimize the error (loss). It works by calculating the gradient (or slope) of the loss function with respect to the parameters and moving the parameters in the opposite direction of the gradient to reduce the loss. ([K4])

GD constitutes a foundational technique in the training of machine learning models, including but not limited to:

- Neural networks
- Linear and logistic regression
- Support vector machines.

Step-by-step process:

1. **Initialize parameters**. Start with random values for model parameters.

2. **Compute gradient.** Calculate the gradient of the loss function with respect to the parameters. The gradient of the loss function is defined as the vector of partial derivatives of the loss function with respect to each of the model parameters. It indicates the direction and rate of the steepest increase in the loss function.

If the loss function is $\theta \to L(\theta)$, where $\theta = [\theta_1, \theta_2, \dots, \theta_n]^T$ represents the model parameters, the gradient is:

$$\frac{\partial L(\theta)}{\partial \theta} = \left[\frac{\partial L(\theta)}{\partial \theta_1}, \frac{\partial L(\theta)}{\partial \theta_2}, \dots, \frac{\partial L(\theta)}{\partial \theta_n}\right]^T,$$

where $\frac{\partial L(\theta)}{\partial \theta_i}$ is the partial derivative of the loss function with respect to the i-th parameter θ_i. The gradient provides the direction to update parameters in order to minimize the loss

3. **Update parameters.** Adjust the parameters using the formula:

$$\theta' = \theta - \alpha \frac{\partial L(\theta)}{\partial \theta}$$

---

[61] Gradient descent (GD) is a method of unconstrained mathematical optimisation. It is a first-order iterative algorithm used to minimise a differentiable function of several variables. The idea is to repeatedly move in the opposite direction to the gradient of the function at the current point because this is the direction of steepest descent. *Gradient ascent* (GA) works in the same way as gradient descent, except for one difference. While gradient descent indicates an iterative movement towards the closest minimum, gradient ascent indicates a movement towards the nearest maximum. In other words, gradient descent and gradient ascent are algorithmically identical, differing only in the direction in which they update model parameters. One seeks to maximise an objective and the other to minimise it. Gradient descent is generally attributed to Augustin-Louis Cauchy, who first suggested it in 1847 ([G12]).

where:

θ represents the model parameters,

α is the learning rate (step size for updates),

$\frac{\partial L(\theta)}{\partial \theta}$ is the gradient of the loss function L at θ.

A positive gradient indicates that the parameter should decrease in order to reduce the loss. Conversely, a negative gradient means that the parameter should increase to reduce the loss.

4. **Iterate.** Repeat the process until the loss is minimized or convergence is achieved.

There are numerous variants of the GD algorithm:

- **Batch gradient descent**. This method utilises the entire data set to compute the gradient, which is computationally expensive but stable.

- **Stochastic Gradient Descent (SGD)**. This method uses a randomly selected subset of the training data, often just a single example or a small mini-batch. This results in enhanced speed, but also increased noise.

- **Mini-batch gradient descent**. This approach represents a compromise between the two aforementioned methods, by employing smaller data sets for updates.

The next chapter will cover a variety of applications of gradient descent, as well as its limitations.

3.2.11. Random forest

Random forest is a popular and powerful machine learning algorithm that belongs to the family of *ensemble methods*. It is used for both classification and regression tasks and works by combining multiple decision trees to achieve more accurate and stable predictions. This algorithm is valued for its simplicity, versatility, and performance.

The following real-world analogy might help: A random forest can be conceptualised as a jury in a courtroom. Each decision tree functions as a juror, providing an independent opinion based on the evidence (data). The collective opinion of the jury (i.e. the final verdict or prediction) is determined by majority vote or by averaging the individual opinions.

Here is a detailed breakdown of the inner workings of random forest and the factors that contribute to its efficiency:

- **Core idea**. Random forest is an ensemble learning method that uses multiple decision trees. The goal is to aggregate the predictions of many weak learners (decision trees) to create a strong learner that performs better than any individual tree.

- **Training process**. The training process consists of the following steps:

- *Bootstrap aggregation (bagging)*

  - From the original training dataset, n random samples are drawn with replacement ([62]) to create multiple subsets (bootstrap samples).

  - Each subset is used to train a separate decision tree.

  - By training on different subsets, the trees become diverse, reducing overfitting.

- *Feature randomness*

  - At each split in a tree, a random subset of features is considered instead of evaluating all features.

[62] Sampling with replacement means that after you randomly pick an item from a population, you "put it back" before drawing again.

  - This randomness ensures the trees are de-correlated, further improving generalization.
- *Decision tree construction*
  - Each decision tree is grown independently and to its full depth (no pruning ([63])).
  - Trees split data by maximizing metrics like Gini impurity (for classification) or minimizing Mean Squared Error (for regression).
- **Prediction process**. Once the forest of trees is trained, predictions are made as follows:
- *For classification*
  - Each tree casts a 'vote' by predicting a class for a given input.
  - The final prediction is determined by *majority voting* (the class with the most votes is chosen).
- *For regression*
  - Each tree predicts a continuous value.
  - The final prediction is the average of all tree outputs.

The random forest algorithm has a variety of applications. For instance:

- Predicting customer behaviour (e.g., churn prediction).
- Medical diagnoses (e.g., disease classification).
- Finance (e.g., risk assessment).

When used for classification or regression problems, the random forest algorithm presents a number of limitations and challenges. These include ([G23]):

× **Limited interpretability**. Random forests aggregate decisions from many trees, which makes it difficult to trace how individual features influence the final prediction. Explaining the results often requires additional tools to understand the ensemble behaviour of this "black box".

× **High computational and memory cost**. Training and storing hundreds or thousands of deep decision trees demands significant CPU, memory and disk resources.

× **Overfitting on noisy features**. Although ensembles reduce variance relative to single trees, random forests can still overfit if the trees become too complex or if irrelevant, noisy features dominate the splits. Without proper pruning or regularisation, the model may capture spurious patterns that hinder generalisation.

× **Bias in imbalanced data**. When class distributions are skewed, random forests tend to favour the majority class. Rare classes may be under-represented, resulting in poor classification of minority categories, unless class weighting or resampling techniques are applied.

3.2.12. Artificial Neural Networks (ANNs)

Artificial Neural Networks (ANNs), also referred to as *Neural Networks*, are a powerful class of machine learning algorithms inspired by the structure and function of the human brain. They mimic the structure of the human brain, which is made up of layers of interconnected neurons. Neural networks have become a cornerstone of modern AI because of their ability to learn complex patterns and make accurate predictions. They are used in a wide range of AI applications, from image and speech recognition to natural language processing and more.

---

[63] *Pruning* is a technique used to simplify a decision tree by reducing its size while maintaining or improving its generalization ability. It removes unnecessary splits (branches) that may cause the tree to overfit the training data. Pruning ensures that the tree focuses on the most relevant patterns in the data, making it more robust when dealing with unseen data.

Neural networks are central to modern AI, particularly deep learning, making them a foundational AI algorithm. Here is a brief explanation of how neural networks work (see Section 4.3.2 for further details):

- **Basic concept**. Neural networks consist of layers of interconnected nodes (neurons) that process input data to make predictions or classifications. Each neuron receives input, applies a weight and bias, and then passes the result through an activation function to produce an output. The architecture of neural networks resembles the decision-making processes of the human brain, with neurons working together to process complex data and make predictions.

The learning process allows neural networks to improve over time, refining their predictions as they receive more data. Input data enters the network, passes through several layers, and is processed at each stage to produce the final output. The connections between neurons (called *weights*) determine how data is passed from one neuron to another.

- **Structure of a neural network**. The layers of an artificial neural network are divided into three groups:

1. *Input layer.* The input layer receives the raw data. Each neuron in this layer represents a feature of the input data.

2. *Hidden layers.* Hidden layers perform computations on the input data. A network may contain between 0 and k hidden layers. A network that does not possess a hidden layer is called a *single-layer* ANN, whereas a *multi-layer* ANN is characterised by the possession of multiple hidden layers. The term 'hidden' is used to denote the fact that these neurons are not directly observable by the input or output layers..

3. *Output layer.* The output layer produces the final result, such as a classification or a prediction.

4. *Activation functions*. Activation functions play a crucial role in neural networks by determining whether a neuron should 'fire' or not.

ANNs form the basis for a variety of specialised deep learning architectures, including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Generative Adversarial Networks (GANs), and Transformers. All deep learning models are built upon the basic neuron structure of ANNs, of layers of interconnected units that process and transform information. Moreover, the weight-updating mechanism (backpropagation) used in ANNs is extended to these advanced models.

3.2.13. Convolutional Neural Networks (CNNs)

Convolutional neural networks (CNNs) are a type of neural network and are currently one of the most well-known models for image classification. They are designed specifically for processing structured grid data, such as images. Convolutional layers enable the automatic and adaptive learning of spatial hierarchies of features.

The initial motivation for the development of CNNs came from experiments conducted by Hubel and Wiesel on the visual cortex of cats ([H7]). The visual cortex contains small groups of cells that respond to particular areas of the visual field. Moreover, the cells that become excited depend on the shape and orientation of the objects in the visual field. For instance, some neuronal cells are excited by vertical edges, while others are excited by horizontal edges. These cells are connected using a layered architecture, leading to the hypothesis that these layers are used to construct images at different levels of abstraction. From a machine learning perspective, this concept is analogous to hierarchical feature extraction ([A2]). As we will see in Section 4.3.5, CNNs achieve a similar effect by encoding primitive shapes in earlier layers and more complex shapes in later layers.

Applications of CNNs are:

- **Image classification**. This involves the identification of objects or scenes present within an image. Example: CNNs power models like *ResNet* ([64]), *VGGNet* ([65]), and *Inception* ([66]).

- **Object detection**. Locating specific objects in an image and drawing bounding boxes around them (e.g., traffic sign recognition or identifying pedestrians in a self-driving car's camera feed).

- **Image segmentation**. Dividing an image into regions and classifying each pixel (e.g., identifying roads, vehicles, and sky in autonomous driving).

- **Facial recognition**. Identifying and verifying human faces in images or videos.

- **Natural Language Processing (NLP)**. CNNs are also adapted for tasks like text classification or sentiment analysis by treating text as sequences.

- **Pharmaceutical research**. CNNs have also been used for drug discovery. They help to identify potential treatments by predicting the interactions between molecules and biological proteins.

- **Medical imaging**. Detecting anomalies in medical scans like X-rays, CTs, or MRIs. One example is the MONAI (Medical Open Network for AI) framework ([M7]). This open-source, PyTorch-based system is tailored for medical imaging and offers domain-optimised data pipelines, artificial networks and evaluation metrics, as well as 'Model Zoo' bundles. MONAI's Model Zoo provides pretrained AI models that can be downloaded and utilised for processing new data. The prostate MRI anatomy model, for instance, was trained using the U-Net architecture ([67]) and is employed for 3D volumetric segmentation of the anatomical prostate zones on MRI images.

See Section 4.3.5 for a detailed exposition of CNNs, including their limitations and challenges.

3.2.14. Recurrent Neural Networks (RNNs)

Have you ever wondered how chatbots understand your questions or how voice assistants like Siri and Alexa interpret your spoken requests? These impressive functions are made possible by a type of artificial intelligence known as recurrent neural networks (RNNs).

Recurrence is a fundamental aspect of information processing and integration in biological neural networks, particularly in the brain. The brain contains numerous hierarchically nested feedback loops. With an estimated 10^{11} neurons, each with an average of 10^4 connections, it is calculated that, on average, each signal returns to its source after passing through only three synapses ([K6]). The brain's neurons are intricately interconnected, forming many circuits that feature feedback and lateral connections. These recurrent circuits allow the brain to integrate information about the past, present and anticipated future into its processing. Furthermore, this recurrent structure enables the brain to process information extremely efficiently and dynamically.

Recurrent neural networks (RNNs) are much simpler versions of the recurrent architectures observed in biological neural networks, particularly in the brain. However, biological recurrence involves many layers of modulation, including temporal dynamics at multiple scales. RNNs

---

[64] ResNet (*Residual Network*) was developed by a team of Microsoft researchers led by Kaiming He, along with Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Their work was introduced in [H4]. ResNet revolutionized deep learning by enabling the training of very deep neural networks through the use of residual connections.

[65] VGGNet was developed by the *Visual Geometry Group* (VGG) at the University of Oxford. The key researchers behind it were Karen Simonyan and Andrew Zisserman, who introduced the architecture in [S5]. This model became a cornerstone in deep learning for its simplicity and effectiveness.

[66] The Inception architecture, also known as *GoogLeNet*, was developed by a team of researchers at Google in 2014. The innovative Inception modules for efficient and scalable deep learning were introduced in [S6].

[67] U-Net is a CNN architecture developed by Olaf Ronneberger et al. for biomedical image segmentation at the University of Freiburg, Germany, in 2015 ([R8]). It is one of the most widely used approaches for semantic segmentation tasks today.

capture only a basic form of temporal dependency and do not model many of these underlying processes. Essentially, RNNs are an abstract model inspired by the recurrent structures of the brain, enabling artificial systems to handle sequences. Nevertheless, the brain's recurrence involves a far richer set of dynamics and mechanisms, which are the subject of ongoing research in neuroscience and computational modelling.

RNNs are designed to process temporal dependencies and sequences in data. RNNs possess connections that establish loops, enabling them to maintain a memory of previous inputs. This makes them well-suited to tasks involving time series data and natural language processing, as well as other problems where the order of the input is important. The ability of RNNs to process sequences makes them highly effective in a variety of applications.

Here are the main areas where RNNs are used:

- **Natural Language Processing (NLP).** RNNs can perform a variety of tasks, such as converting spoken words into written text (e.g. Siri and Google Assistant), translating text between languages (e.g. Google Translate), generating human-like writing (e.g. chatbots and story generation) and determining emotions in text (e.g. analysing tweets and reviews). RNNs power AI chatbots, for example, by learning conversational patterns.

- **Time-series forecasting.** RNNs are used for identifying trends in financial markets, as well as for weather forecasting and consumption analysis. For example, banks use RNNs to predict financial risks based on transaction histories.

- **Speech & audio processing**. Applications include music composition (generating original melodies using past music patterns) and speaker identification (recognising voices for authentication systems). Example: Voice-controlled AI assistants rely on RNNs to understand speech.

- **Video analysis & action recognition**. RNNs are used for gesture recognition (identifying movements for interactive applications), video captioning (automatically generating descriptions of videos) and human activity recognition (detecting specific actions in surveillance videos). For instance, RNN-based models assist self-driving cars in recognising pedestrian movements.

- **Healthcare & biomedical applications**. RNNs are used for medical diagnosis (e.g. predicting diseases from patient health records), drug discovery (e.g. analysing chemical sequences for pharmaceutical research) and ECG signal analysis (e.g. detecting irregular heart patterns). Example: Hospitals use RNNs to monitor patients and detect diseases at an early stage.

In summary, RNNs excel in sequence-driven tasks where an understanding of context over time is essential. Traditional RNNs tend to struggle with long-term dependencies, but advanced variants such as LSTMs (long short-term memory) and GRUs (gated recurrent units) can improve performance.

Section 4.3.6 provides a detailed discussion of RNNs, including their limitations and challenges.

3.2.15. Generative Adversarial Networks (GANs)

Deep neural networks are capable of more than simply learning mappings from data examples to labels. For instance, when working with a large unlabelled dataset, it can be useful to create a model that accurately captures its key features. This model can then be used to generate synthetic data that mimics the characteristics of the original dataset. One possible application of this would be to generate a new photorealistic image that appears to have come from a given dataset. This type of learning is known as *generative modelling* ([Z3]).

Generative Adversarial Networks (GANs) are a revolutionary development in artificial intelligence, particularly in deep learning and generative modelling. Introduced by Ian Goodfellow et al. in 2014 ([G7]), they have rapidly evolved to become a powerful framework for generating realistic data, including images, videos, to audio and text. Unlike traditional

generative models, which often rely on explicit density estimation ([68]), GANs use a unique adversarial training process that opposes two neural networks in an ongoing zero-sum game.

The fundamental concept underlying GANs is to engage two neural networks in a form of adversarial competition. Their architecture consists of two neural networks: the *generator* and the *discriminator*. The generator learns the underlying distribution of the training data and produces new synthetic data samples that resemble the real ones. Initially, the generator produces highly unrealistic outputs, but through iterative training, its ability to create increasingly convincing fakes improves. At the same time, the discriminator acts as a binary classifier, distinguishing between real data samples from the training set and fake data samples generated by the generator. The discriminator's objective is to become an expert at identifying synthetic data, while the generator's goal is to produce data that is so realistic that the discriminator cannot tell the difference between it and genuine samples.

This adversarial process continuously improves both networks. The generator learns to produce more sophisticated fakes to fool the discriminator. In turn, the discriminator becomes more adept at detecting subtle imperfections. This competitive dynamic leads to a state of equilibrium in which the generator can produce highly realistic and diverse data that closely resembles the original training distribution. The framework's elegance lies in its ability to learn complex data distributions without explicit programming, making it a versatile tool for a variety of applications.

Let us take a look at the main applications of GANs.

- **Image generation & enhancement**. GANs can generate realistic paintings, drawings, and digital art. They can enhance low-resolution images to create high-definition versions and convert images into different artistic styles. Furthermore, GANS can produce realistic fake videos and face-swapping effects. For example, deepfake GANs can modify facial features in videos with high precision.

- **Medical & scientific applications**. GANs can be used to generate medical images, such as synthesising CT and MRI scans for training AI models. They can also be deployed for anomaly detection, identifying rare diseases by learning normal patterns and detecting deviations. Other applications include drug discovery and molecular simulation. GANs can assist in creating new chemical structures for pharmaceutical research.

- **Data augmentation for machine learning**. Applications include the generation of synthetic data (producing realistic training data when real data is limited) and text-to-image conversion (creating training images from textual descriptions).

- **Video & animation creation**. GANs can create lifelike human avatars for gaming and virtual reality (VR). They can also be used for video prediction and frame interpolation, whereby missing frames are generated to improve video smoothness. Another application is the generation of 3D objects, which can be useful for designing virtual models for use in animation and augmented reality.

- **Security & privacy applications**. GANs are used for AI-based cybersecurity, detecting anomalies in network traffic by modelling normal patterns. They can also be applied to data obfuscation and the creation of synthetic identities, generating artificial data to protect privacy.

In summary, GANs are transforming image processing, medical research, creative AI, and data generation. They provide innovative solutions across a range of industries. Their ability to generate hyper-realistic media and synthetic data has made them as a cornerstone of AI innovation.

---

[68] Density estimation involves constructing a function that describes the distribution of data points. Rather than focusing on individual data points, the aim is to identify the overall pattern of their distribution. This is the basis for other statistical tasks such as classification, goodness-of-fit tests and anomaly detection.

Section 4.3.7 provides a comprehensive treatment of GANs, covering their principles, applications, and limitations.

3.2.16. Transformers

Transformers are a family of neural network architectures that use attention mechanisms exclusively to process sequences. First introduced by Vaswani et al. ([V2]), they have revolutionised natural language processing (NLP) by enabling models to learn long-range dependencies without the need for recurrence or convolution.

Unlike RNNs, transformers process all tokens simultaneously, which significantly speeds up training. Large pretrained transformers can be fine-tuned with minimal data. Transformers form the basis of innovative solutions in language, vision, audio, multimodal, code and scientific domains.

Examples of applications include:

- **Natural Language Processing**. Transformers stand out for their ability to model text by capturing long-range dependencies in a single pass. They are used for machine translation — Google's T5 (see Section 5.2.5.2) and Meta's M2M-100 can translate between dozens of languages — as well as for text summarisation, question answering, chatbots, named entity recognition, and sentiment analysis.

- **Computer vision**. Transformers have revolutionised vision tasks by treating image patches like tokens. They are used for image classification, object detection and segmentation, image generation, and video understanding.

- **Speech and audio processing**. Attention mechanisms help to model temporal patterns in audio for recognition and generation purposes. Transformers are employed in applications such as automatic speech recognition, text-to-speech synthesis, audio classification, and music generation and enhancement.

- **Multimodal and cross-modal learning**. Transformers unify different data types, providing a richer context for reasoning. Examples include image captioning, visual question answering, multimodal search, and video-text alignment. Models such as Flamingo integrate video frames with captions for narration or analysis.

- **Code and science**. The attention framework extends from perception to reasoning and planning. Transformers are employed for code generation and completion. Codex and PolyCoder, for example, generate code snippets and assist with developer workflows. Another application is protein folding and drug discovery. AlphaFold applies attention to amino acid sequences to predict their structure.

A detailed discussion of Transformers, including their limitations and challenges, is provided in Chapter 5.

3.2.17. Diffusion models

Diffusion models are a class of generative algorithms that approach data synthesis from an unusual angle. Rather than learning how to create data from scratch, they learn to reverse a gradual corruption process. At the heart of a diffusion model are two opposing processes (see [G20], [Y3] and [Y4] for further details):

- **Forward (diffusion) process**. Starting with real data (e.g. an image), Gaussian noise is added in many small steps until the data consists almost entirely of noise.

Gaussian noise is a type of statistical noise in which the values follow a Gaussian (i.e. normal) distribution. It is often used to model random fluctuations in signals and images. In the case of diffusion models, the multivariate normal distribution is used, which is an extension of the normal distribution of one variable. A multivariate normal distribution takes a real-valued vector as input and is defined as follows:

$$\mathcal{N}(X;\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^n det\Sigma}} exp\left(-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)\right)$$

where

X is a n-dimensional random vector

μ is the mean vector

Σ is the covariance $n \times n$ matrix (see Section 3.2.9)

$det\Sigma$ is the determinant of the covariance matrix

Σ^{-1} is the inverse of the covariance matrix

$(X-\mu)^T$ is the transpose of the vector $X-\mu$.

Note that $\mathcal{N}$ is a scalar (i.e. a number), since the expression $(X-\mu)^T\Sigma^{-1}(X-\mu)$ is the scalar product of the two vectors $(X-\mu)^T$ and $\Sigma^{-1}(X-\mu)$, and is therefore also a scalar.

Gaussian noise is incrementally added at each iteration during the forward process. This process can be effectively represented using a Markov chain ([69]), where each state transition is governed by a predefined probability distribution. Starting from a flattened image (see Section 4.3.5) or tokenised text X_0 (see Chapter 5), Gaussian noise is added in T discrete steps:

$$q(X_t;X_{t-1}) = \mathcal{N}\left(X_t;\sqrt{1-\beta_t}X_{t-1},\beta_t I\right),$$

where

$X_{t-1}, 1 < t \leq T$, is the previous state

$\sqrt{1-\beta_t}$ is a scaling factor that retains most of the signal from X_{t-1}

$\beta_t \in (0,1)$ is the variance of the fresh Gaussian noise added at step t

I is the identity matrix, indicating uncorrelated noise across dimensions.

Observe that $q(\cdot\,;X_{t-1})$:$\mathbb{R}^n \to \mathbb{R}_+$ is a function that returns the density of each candidate X_t. One can think of q as either a rule for sampling, 'Given X_{t-1}, draw X_t from this Gaussian', or as a density, 'If you plug in any candidate X_t, $q(X_t;X_{t-1})$ tells you how likely it is.'

Diffusion models define the noise injection process as follows

$$X_t = \sqrt{1-\beta_t}X_{t-1} + \sqrt{1-\beta_t}\epsilon_{t-1},$$

where ϵ_{t-1} is a sample drawn from a Gaussian distribution $\mathcal{N}(\cdot\,;0,\beta_t I)$.

The sequence $\{\beta_t\}_{t=1}^T$ is called the 'noise schedule', as it controls the rate at which noise is added to the original data distribution $q(\cdot\,;X_0)$ to create an approximately Gaussian distribution $q(\cdot\,;X_t)$. The variable t denotes the timestep in the noise schedule and belongs to the set $[1,T]$, where T is the total number of steps in the noise schedule. As t increases, X_t progressively converges towards a pure Gaussian noise distribution.

The forward chain is a fixed, non-learned process that can be considered a controlled, statistical form of 'destruction' of structure.

- **Reverse (denoising) process**. Unlike the forward process, which adds noise to the data distribution progressively, the reverse process aims to remove noise and reconstruct the original data. This denoising process, also known as the reverse process, is carried out using a

[69] A Markov chain is a mathematical model of a sequence of random variables, in which the probability of each state depends solely on the previous state immediately preceding it. This 'memoryless' property is known as the Markov property.

deep learning model that is trained to approximate the noise and ultimately learn to remove it from a Gaussian-distributed latent variable ([70]).

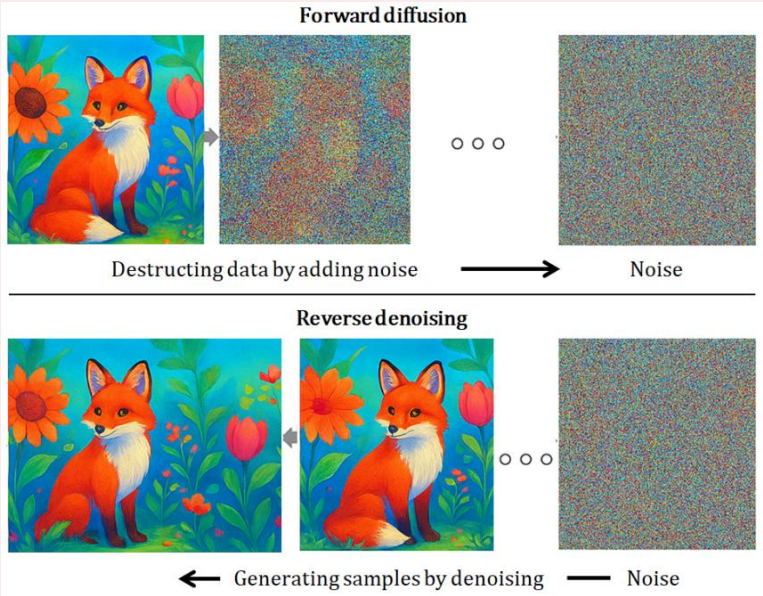
In other words, the model is trained to reverse the forward process. Given a noisy sample, X_t, the model predicts and removes the noise to recover X_{t-1}. This process is repeated until clean data emerges from what began as pure noise. The training objective is usually to minimise the difference between the noise added during the forward process and the noise predicted by the model.

At first glance, it seems paradoxical: if the forward diffusion process destroys your data by converting X_t into pure Gaussian noise, how could running it backwards produce anything more impressive than the original X_0? The key lies in the fact that, during training, the model never learns to 'magically improve' a specific X_0. Instead, it learns a probabilistic map from noise back to the entire learned data distribution. This is why the reverse process can produce new samples that are sometimes visually superior to the original inputs.

Let us take a closer look at this. Training involves many X_0 examples. The forward process begins with real samples from the dataset. At each step t, the model is presented with a noisy version of the data X_t, and is trained to predict the added noise. The model sees millions of real X_0 examples across the entire dataset and learns the statistical structure of the data. Consequently, it recognises what 'clean' data tends to look like at different noise levels.

Once trained, the model is fed any pure noise sample X_T from $\mathcal{N}(\cdot\,; 0, \beta_t I)$ in the reverse process. This process progressively removes noise in T steps, guided by the patterns it has learned that are typical of the data distribution. Because the model knows the distribution, rather than just the original image, it can 'hallucinate' plausible details that were never present in a specific X_0 – this is where 'creativity' emerges.

Example. The picture below ([71]) shows how a clear image becomes progressively degraded by adding noise, step by step. Then, the noise is removed in reverse to form a different, yet equally coherent, image.



In the forward diffusion, the original image is filled with fiery oranges, deep purples, and turquoise waves. Gradually, random coloured speckles obscure the palm trees, shoreline and

[70] In a diffusion model, a latent variable is a hidden, continuous representation of the data at a given stage of the diffusion process. The process as a whole is a Markov chain that gradually adds noise to the data. Denoted as $X_1, X_2, \ldots, X_t$, these latent variables capture progressively 'noisier' versions of the original data X_0, ultimately transforming it into pure noise at the end of the chain.

[71] Image created by Microsoft Copilot (AI- generated).

glowing sky, leaving only noise. After reverse denoising, the final regenerated image captures the same mood but with playful variations.

Diffusion models have quickly evolved from research curiosities into production-grade tools used in a variety of industries. The following is an overview of some of their applications:

- **Visual media generation**. This includes text-to-image synthesis, whereby high-quality images are generated from natural-language prompts (e.g. Stable Diffusion and DALL·E 2 models). Another application is image editing and inpainting, for example removing objects, filling in missing regions or changing styles by conditioning the reverse process on masks or reference images. Furthermore, it is possible to upscale low-resolution photos or remove noise/grain from images to restore fine details.

- **Audio and speech**. Examples range from producing natural-sounding text-to-speech and auto-completing musical scores to generating novel melodies and cleaning up old recordings.

- **Healthcare imaging**. This involves enhancing the clarity of MRI/CT scans and reducing scan time. Furthermore, when labelled data are scarce, models can synthesise realistic tissue slides that can be used to train diagnostic algorithms.

- **Drug discovery and molecular design**. Models can be used to construct novel chemical compounds with the desired pharmacological properties.

- **Text generation**. Models can produce descriptions based on attributes or keywords. They can also paraphrase, translate and summarise, as well as denoise a corrupted input sequence to produce a refined output.

In summary, diffusion models learn a general denoising process that can be adapted to many types of data, including images, text, audio, molecules and graphs. This makes them a flexible foundation for generative AI.

Remark. In text-to-image generation, a diffusion model is trained using paired samples. Each example comprises an image and a natural-language description, enabling the model to associate words with pixels. In the training pipeline, the 'noising' (forward diffusion) step is applied solely to the image data structure. The caption is not altered or 'noised' at all during this phase. Text-based conditioning takes place in reverse denoising. At each step t, the model uses a noisy image X_t and a caption. Thus, text guides the removal of noise, shaping the image towards the semantics of the prompt.

Although diffusion models have shown significant potential in generating high-quality outputs, several open challenges hinder their broader development and deployment ([C9]).

× **Computational and resource demands**. Training and inference with diffusion models require significant computing power and memory. Their iterative denoising processes often entail hundreds of sampling steps, resulting in slow generation times and high energy consumption.

× **Limited theoretical understanding**. Although diffusion models are empirically successful, they lack comprehensive theoretical foundations. Key questions regarding convergence rates, expressivity limits and the impact of noise schedules remain unanswered, hindering systematic enhancements and complicating the prediction of failure.

× **Sampling efficiency versus sample quality trade-off**. Reducing the number of denoising steps accelerates sampling, but this often results in a loss of image fidelity or diversity. Designing optimal trade-offs between generation speed and output quality requires sophisticated heuristics or learned schedulers, which complicates model design and tuning.

× **Data hunger and bias propagation**. These models usually need large amounts of high-quality data to capture complex data distributions. However, they can also inadvertently amplify biases present in the training data, resulting in the generation of inappropriate content.

× **Lack of standardised evaluation metrics**. There is no consensus on how to quantify sample quality and fidelity across modalities. While some metrics offer proxies, these can diverge from human judgements, making it difficult to benchmark progress.

The foundations of artificial intelligence are based on the dynamic interaction between theory and application. Understanding its domains reveals what AI can do, while understanding its algorithms explains how it achieves this. Together, these two aspects provide the conceptual and computational basis for developing systems that enhance human cognitive abilities and deepen our understanding of intelligence.

4. Machine Learning (ML) – Transforming Data into Intelligence

Everyone wants a machine that learns –
until it learns something they don't like.
– Anonymous

In the contemporary digital era, data has been frequently referred to as 'the new oil', a term denoting its significant potential. However, it is important to note that raw data in isolation is of limited value. It is only through the processes of preparing, analysing and transforming it into actionable insights that it can be of value. *Machine Learning* (ML) has emerged as a revolutionary approach to this process, enabling computers to learn from data and make decisions with minimal human intervention. Within this field, *Deep Learning* (DL) is a subset that stands out for its ability to emulate the structure and functioning of the human brain, thus enabling it to tackle complex problems with greater precision than ever before.

The field of machine learning encompasses a diverse array of techniques that facilitate system enhancement through experience. From recommending the next movie to stream, predicting stock prices, to diagnosing medical conditions, ML has permeated nearly every facet of modern life. The field of deep learning represents a significant advancement in this area, utilising artificial neural networks to automatically identify intricate patterns in voluminous datasets. This capability enables the successful resolution of challenges that were previously considered insurmountable, such as image recognition and natural language understanding.

This chapter explores how machine learning can transform raw data into meaningful insights. It examines the foundational principles of machine learning, looking at its key concepts and paradigms, such as supervised, unsupervised and reinforcement learning. It explains how these approaches enable systems to predict, classify and optimise. The latter part of the chapter is devoted to deep learning and its neural network architecture, and goes on to outline typical ML workflows. Finally, the chapter addresses the challenges and limitations of machine learning.

Further information can be found in [A2], [G4], [H5], [S1], [S4], [T1] and [Z1].

4.1. Basic concepts of Machine Learning

Machine learning is a subfield of AI that focuses on the process of building systems that can learn from data, identify patterns and make decisions with minimal human intervention. Contrary to being explicitly programmed to perform a specific task, machine learning algorithms are trained with extensive data sets, using this data to make predictions or decisions. In other words, in explicit programming, humans manually write the rules. In ML, the system learns the rules from data. To understand this difference, we can use the example of building a spam email filter. In explicit programming, the developer manually writes rules to detect spam, such as following:

- If the email contains certain keywords like "Free money", "Win big" or "Limited time offer" mark it as spam.
- If the email sender's address is in a blacklist, mark it as spam.

In this paradigm, the programmer is responsible for implementing the rules, and the system is designed to adhere strictly to these parameters. It is important to note that this method requires constant updates as new patterns in spam emails emerge. With ML, the system learns from data. It is fed a dataset of emails that have been labelled as 'spam' or 'not spam' and the ML model automatically identifies patterns. Specifically, during the training phase, the model identifies features (e.g., word frequency, sender reputation) that are associated with spam emails. Once trained, the model is able to generalise and classify new emails as 'spam' or 'not spam' based on these learned patterns.

The advantage of this approach is that the ML model can adapt as spam patterns evolve, without requiring explicit updates to the rules.

To understand machine learning and how it works, it is essential to grasp its key elements, which are fundamental to developing and evaluating models. Machine learning has several core concepts that form its foundation. These concepts define how machines learn, adapt and provide insights. Here is an overview:

- **Learning paradigm**. This term refers to the approach or method by which an ML model is trained to learn from data. It defines how the model interacts with the data, the environment, and the task at hand. Examples of learning paradigms include supervised and unsupervised learning.

- **Data collection**. Having data is the most essential and important part of building an ML model, preferably lots of data. No data, no model. Depending on the objective, the approaches to collecting data can vary. You can buy existing data sources from other vendors, you can scrape websites and extract data from them, you can use publicly available data, or you can collect your own data.

- **Data sets**. In machine learning, different types of datasets are used. This depends on the task and the type of learning. Based on the purpose in the ML workflow, there are the following datasets:

  - *Training dataset*. A subset of data used to train the ML model. The model learns patterns from this data. It should be large and diverse to ensure good generalisation.
  - *Validation dataset*. A separate subset used during training to tune hyperparameters and prevent overfitting. Helps evaluate the model's performance on data it hasn't been directly trained on.
  - *Test dataset*. A final, unseen subset used to evaluate the model's performance after training. This dataset ensures that the model generalises well to new, real data.

Proper data splitting is essential for creating a reliable and generalisable model. A common split ratio is typically: training set 60-80%, validation set 10-20% and test set 10-20% of the data. The data in the training set should not overlap with either the validation set or the test set. This prevents the model from 'cheating' by unintentionally learning from data it will encounter during testing. Split the data randomly to avoid bias. For example, in time series data, splits should respect the temporal order, but within a range, random sampling ensures representativeness.

Based on the learning paradigm, we have

- *Labelled data*. Data where each input is paired with a corresponding output (label). Used in supervised learning. For example, a data set of email text (input) labelled as 'spam' or 'not spam' (output).
- *Unlabelled data*. Data without associated labels. Used in unsupervised learning. For example, customer purchase history without predefined categories.

- **Data preprocessing**. Data preprocessing in machine learning is a critical step where raw data is cleaned and transformed into a format suitable for training models. This ensures the data is consistent, relevant, and free of noise. Examples of this process include data cleaning and converting categorical variables into numerical values.

- **Features**. Features are the measurable properties or attributes of the data that are used by a model to make predictions or decisions. Essentially, features are the input variables that help the model understand the data and solve a problem. Features are specific to a particular domain. To illustrate this point, consider the following example of features from the field of finance: transaction amount, account balance for fraud detection.

- **Feature engineering.** Feature engineering is the most important part of the model building process in applied ML. Feature engineering is defined as the process of transforming raw input data into more informative data for the purpose of training algorithms. Feature engineering requires good domain knowledge of the dataset, the creativity to build new features from raw

input data, and multiple iterations for better results. In finance, for example, transaction history can be transformed into features such as 'average spending per month'.

- **Model**. In ML, a model is a mathematical representation or function that is trained to recognise patterns in data and make predictions or decisions. It is essentially the 'brain' of the ML system. Here are some key points about ML models:
  - *Functionality*. A model maps input data (features) to output (predictions or classifications). For example, a model can predict house prices based on features such as square footage and location.
  - *Training*. Models learn from training data – examples with known outcomes – so that they can generalise to unseen data.
  - *Types*. Depending on the problem, different types of models are used, such as linear regression, decision trees or neural networks.
  - *Parameters and hyperparameters*. Models have parameters (such as weights in neural networks) that are adjusted during training to improve accuracy. Hyperparameters are configuration settings (e.g. learning rate, number of neural network layers) that control the training process but are not learned from the data. They must be specified beforehand to ensure optimal performance.
  - *Evaluation*. Once trained, models are tested to ensure they perform well on new, unseen data.
- **Algorithms**. Algorithms are mathematical or computational procedures that form the backbone of models, enabling them to learn patterns from data and make predictions or decisions. Algorithms can be thought of as the 'recipe' that dictates how the ML system processes data and learns. Most ML algorithms are based on iterative processes in which the model improves its predictions by minimising error or maximising accuracy. See Section 3.2 for examples of algorithms.
- **Model evaluation**. Model evaluation is the process of assessing how well a trained model performs when making predictions on new, unseen data. It helps ensure that the model is accurate, reliable and suitable for the task. For example, in predicting loan approvals, model evaluation helps determine whether the model is accurate in classifying applicants (approved/rejected) and whether decisions are fair and unbiased. There are several techniques for assessing how well a model performs, using performance metrics such as accuracy, precision, recall or mean square error.
- **Optimisation.** Model optimisation in ML refers to the process of improving the performance of a model by fine-tuning its parameters, hyperparameters or algorithms selection. The ultimate goal of optimisation is to ensure that the model achieves the best possible accuracy, efficiency and generalisation to new data.

4.2. Learning paradigms

A learning paradigm in ML refers to the approach or method by which an ML model is trained to learn from data. It defines how the model interacts with the data, the environment, and the task at hand. Machine learning is commonly classified into four types: supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning.

4.2.1. Supervised learning

Supervised learning is an approach to machine learning where models are trained on labelled data. This means that each input (feature) in the dataset is paired with an output (label), and the model learns to map inputs to outputs by identifying patterns in the data. It is one of the most popular forms of machine learning and is used by most machine learning

algorithms. This type of learning, also known as *inductive learning*, includes classification and regression.

To work, supervised learning requires a significant amount of human intervention because it uses labelled data sets. Data must be divided into features (the input data) and labels (the output data). Features describe individual, measurable units of data. Labels are used to group data by specific characteristics and are often manually assigned by data experts to help explain the context of specific data to the machine ([B3]). For example, based on features such as size and location (input), a data label is house price (output).

With supervised learning, labelled input and output data is constantly fed and re-fed into human-trained systems, offering guidance for machines. This helps predictions to become more accurate as each new set of data is fed into the system. Humans also provide feedback on the accuracy of the machine learning algorithm during this process, helping it to learn over time.

The types of supervised learning tasks can be divided into the following groups:

- **Classification**. This is based on predicting a discrete category or class for inputs. For example, classifying emails as 'spam' or 'not spam'. Another example is that of disease diagnosis, wherein patients are categorized into predefined classes such as 'healthy' or 'diseased'. Commonly used classification algorithms include K-nearest neighbour (see Section 3.2.6), naive Bayes (see Section 3.2.8), support vector machine (see Section 3.2.5), decision trees (see Section 3.2.4), and random forest models (see Section 3.2.11).

- **Regression**. This is a process by which a continuous value is predicted for inputs. Examples of such applications include predicting loan approval, stock market and housing market values. Another example is that of self-driving cars estimating the distance of a pedestrian from the vehicle. The algorithms employed for regression include linear regression (see Section 3.2.1), ridge regression (see Section 3.2.3), and neural networks (see Section 4.3.2).

The advantages of supervised learning are a high degree of accuracy with labelled data and a clear relationship between input and output. However, this approach requires large amounts of labelled data, which can be expensive and time-consuming to obtain. What is more, the model can suffer from overfitting, meaning that it may perform well on training data but poorly on unseen data if it is too complex.

In summary, supervised learning is an effective method for tasks that are structured and predictable. However, it is not well-suited to scenarios that demand flexibility and self-directed exploration.

4.2.2. Unsupervised learning

Unsupervised learning is a valuable tool for identifying structure in data. In scenarios where identifying patterns or anomalies in large, unstructured data sets is particularly challenging, unsupervised learning can offer valuable insights by revealing trends that might otherwise remain obscured. In unsupervised learning, raw data that is not labelled or tagged is processed by the system, which means less work for humans.

The following types of unsupervised learning methods are among the most common ([N1]):

- **Clustering**. Clustering algorithms represent the most prevalent instance of unsupervised machine learning. These algorithms are designed to identify the similarities within raw data and then group this information together. In essence, these algorithms impose structure on raw data. Clustering algorithms are frequently employed in conjunction with marketing data to derive insights into customers (or prospective customers), in addition to their application in fraud detection. Two of the most common algorithms used in the context of clustering are hierarchical clustering ([72]) and k-means clustering (see Section 3.2.7).

[72] Hierarchical clustering is an unsupervised machine learning technique that groups similar data points into clusters, forming a hierarchy that is often visualised as a dendrogram (a tree-like diagram) ([H14]).

- **Dimensionality reduction**. Dimensionality reduction is defined as the process of decreasing the number of features within a data set, whilst maintaining the essential properties of the data. This process is employed to reduce various factors, including processing time, storage space, complexity, and the risk of overfitting in a machine learning model.

The two main methods for applying dimensionality reduction are *feature selection* and *feature extraction*. The former involves the filtering of the original feature set to identify a subset of relevant features, which are then used as input to a model. The process of feature extraction in turn involves the extraction of new, significant features from the original raw data for input, with the aim of removing redundant data and selecting features that will most improve the output. Examples of dimensionality reduction algorithms include Principal Component Analysis (PCA) (see Section 3.2.9), Nonnegative Matrix Factorization (NMF) ([73]) and Independent Component Analysis (ICA) ([74]).

- **Association Rule Learning (ARL)**. This is a technique of unsupervised learning used to discover relationships between variables in large datasets. It identifies patterns, correlations, and associations among items in a dataset. The goal is to find rules that describe how items or events are related. In the context of retail, for instance, it could be employed to derive insights such as 'Customers who purchase bread also tend to purchase butter.' Similarly, in the healthcare sector, it could identify that 'Patients with high blood pressure are likely to have heart disease.'

The common algorithms for ARL include the *apriori algorithm*, which is employed to identify frequent item sets in transactional data, and FP-Growth, which builds a tree structure to discover frequent patterns ([75]). For further details, please refer to [N2] and [B4], respectively.

- **Anomaly detection**. This is a technique used to identify patterns or data points that deviate significantly from the norm without requiring labelled data. The primary function of this methodology is to identify rare, unusual, or suspicious occurrences in datasets where explicit classifications are not available. Since unsupervised learning does not rely on predefined labels, anomaly detection algorithms identify outliers by analyzing data distribution and relationships among data points. The following approaches are commonly used:

  - *Clustering-based methods*. Anomalies are detected as points that do not fit well within any cluster. Algorithms used in this context include K-means clustering (see Section 3.2.7) and DBSCAN (isolated points or small clusters are flagged as anomalies) ([76]).

---

[73] *Nonnegative Matrix Factorization* (NMF) is a technique used to decompose a non-negative matrix into the product of two smaller non-negative matrices. Given a non-negative $n \times p$ matrix X of data points $X_i = [x_{i1}, x_{i2}, \ldots, x_{ip}]$ and a dimension $r < min(p, n)$, NMF aims to compute an $n \times r$ matrix H of r basis elements (features) $H_k = [h_{1k}, h_{2k}, \ldots, h_{nk}]^T$ for $1 \leq k \leq r$ such that the linear space spanned by the H_k's approximates the data points as closely as possible, that is $X \approx H \cdot W$ for some $r \times p$ coefficient matrix W. The matrix H captures latent features for each data point, while the matrix W of weights indicates the contribution of each latent feature to the original features (see [G6] for more details). Here, all elements in X, H, and W are non-negative (≥ 0). The restriction to non-negative values ensures interpretability, especially for real-world data like images, text, or audio. NMF is a state-of-the-art feature extraction algorithm that is useful when there are lots of features that are ambiguous and has weak predictability power. It can produce meaningful patterns, topics, and themes.

[74] *Independent Component Analysis* (ICA) is a computational technique that is utilised for the purpose of separating mixed signals into statistically independent components. Its applications include signal processing, feature extraction, and blind source separation. ICA assumes that observed data is a mixture of independent sources. The goal is to decompose the mixed signals into their original, independent components. This is achieved by maximising statistical independence between the extracted signals (see [R4] for more details).

[75] FP-Growth (*Frequent Pattern Growth*) is an algorithm used in ARL for the efficient discovery of frequent itemsets. The goal is to identify frequent itemsets, which are groups of items that often appear together in transactions.

[76] DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm in machine learning that groups data points based on density. Unlike K-means, which requires specifying the number of clusters, DBSCAN automatically finds clusters of varying shapes and sizes (see [K5] for more details).

- *Statistical methods*. These metrics are based on probability distributions, meaning that an anomaly is defined as a low-probability occurrence. *Gaussian Mixture Models* (GMMs) ([77]) are one example of such methods. In these models, the probability scores of rare events are low.
- *Distance-based methods*. Anomalies are defined as data points that exhibit unusually high distances from their respective neighbours. The algorithm employed is K-Nearest Neighbours (KNN) (see Section 3.2.6).

4.2.3. Semi-supervised learning

Semi-supervised learning can be defined as a hybrid machine learning approach that combines elements of both supervised and unsupervised learning. This methodology is employed in scenarios where a dataset comprises both labelled data, characterised by input-output pairs, and unlabelled data, characterised by inputs without corresponding outputs. It is typically used in situations where the unlabelled data is much larger than the labelled data. The objective of this methodology is to leverage the information contained within the labelled data to facilitate the interpretation of patterns in the unlabelled data.

Semi-supervised learning is a widely used technique for text classification and image segmentation. The model employs labelled data to learn how to make predictions and then uses unlabelled data to identify patterns and relationships. Compared to unsupervised learning algorithms, this strategy offers a number of key advantages. It enables the model to identify relevant patterns and make accurate predictions without incurring the considerable time, effort and cost typically associated with more labour-intensive supervised learning algorithms.

Semi-supervised learning is particularly well-suited to scenarios where the availability of labelled data is limited. Examples of such scenarios include Natural Language Processing (see Chapter 5), image and speech recognition, medical diagnosis and fraud detection.

Algorithms used in semi-supervised learning include Generative Adversarial Networks (see Sections 3.2.15 and 4.3.7) and Transductive Support Vector Machines (TSVM). A TSVM is an extension of a Support Vector Machine (see Section 3.2.5) and is designed to function with unlabelled data.

4.2.4. Reinforcement Learning (RL)

Reinforcement Learning is a machine learning paradigm in which an agent learns to make decisions by interacting with an environment. The objective of the agent is to maximize cumulative rewards over time by taking actions that lead to desirable outcomes.

RL deals with an autonomous agent that must make intuitive decisions and consequently learn from its actions, often without any existing data. This is in contrast to the more structured approaches of supervised learning, where the learning process is guided by the presence of labelled examples, and unsupervised learning, which is predicated on the detection of patterns in the data. Agents are defined as artificial intelligence software programs that are equipped with sensors, which enable them to make autonomous decisions in the context of their immediate environment with the goal of achieving a predetermined outcome.

The fundamental principle of RL is the acquiring of knowledge about the inherent dynamics of the environment (e.g. identifying actions that yield positive rewards and those that do not), with the aim of optimising cumulative rewards over time through a process of trial and error exploration. This process enables AI to undergo continuous improvement. The benefit of RL lies in its ability to deal with situations where the rules are not explicitly known, but can be learned by interacting with the environment. The theoretical underpinnings of RL lie in behavioural

[77] *Gaussian Mixture Models* (GMMs) are a probabilistic approach to clustering that represent data as a mixture of multiple Gaussian distributions. Instead of assigning each data point to a fixed cluster (like K-means), GMMs model soft clustering, meaning each point belongs to multiple clusters with varying probabilities (see [M6] for more details).

psychology, as it emulates the learning processes observed in humans and animals through a gradual process of trial and error.

Let us walk through the fundamentals of RL.

1. **Core Components**. The following key concepts are foundational to RL:

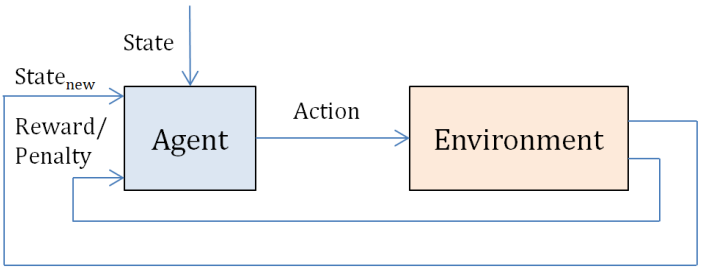
- *Agent*. The learner or decision-maker (e.g., an AI system)
- *Environment*. The external world the agent interacts with (real or simulated).
- *Actions*. The set of choices or moves the agent can make during interaction with the environment.
- *State*. A state is defined as a specific condition or configuration of the environment at a given time, as perceived by the agent.
- *Rewards*. The feedback the agent receives after taking actions, helping it learn what is desirable. Rewards are crucial factor because they provide the agent with a goal at each time step, defining both local and global objectives that the agent aims to achieve over time. The function of rewards is to differentiate between positive and negative events, thereby helping to update policies according to the results of actions.
- *Policy*. The function of policy is to guide the behaviour of a learning agent by mapping observed states of the environment into actions. The function may be simple or complex; for the former, a lookup table may suffice, whereas for the latter, a neural network may be employed as an approximation.

2. Learning Process. At its core, RL follows this feedback loop:

- The agent observes the current state of the environment.
- It chooses an action based on its current understanding (or policy).
- The environment undergoes a transition to a new state in response to the action.
- The agent receives a reward or penalty as feedback for its action.
- The agent refines its policy by learning from this experience, aiming to maximize cumulative rewards. This is achieved through a trial-and-error process.

The ultimate aim of reinforcement learning is to develop an optimal policy – a set of rules or strategies – that dictates the best action to take in any given situation to maximize the long-term reward. In essence, RL emulates a learning process in which actions are driven by outcomes, thereby refining itself with experience to achieve the best possible results. This is a fundamental aspect of the development of intelligent systems that are capable of adapting and improving over time.

The subsequent figure illustrates the fundamental principle of RL ([G4]).



In RL, there are several foundational algorithms that have been developed to enable agents to learn effective policies. Two common approaches are *model-free RL* and *model-based RL*. In model-free RL, the agent learns directly from interactions without building a model of the

environment ([78]). In contrast, model-based RL involves the agent constructing and utilising a model of the environment for the purpose of planning and decision-making.

Here are some of the key algorithms in RL:

- **Q-Learning.** It is a model-free algorithm that learns the optimal *action-value* function (*Q-value*) by updating it iteratively using the Bellman equation. Q-value is defined as the expected cumulative reward that will be received by an agent as a result of taking a specific action in a given state, subsequent to the execution of its optimal policy. The 'Q' is an abbreviation for 'Quality', and it quantifies how beneficial an action is, considering future rewards.

The *Bellman equation* represents a fundamental concept in RL and dynamic programming. It provides a methodology for the decomposition of the overall value of a decision problem into smaller, more manageable subproblems.

In mathematical terms, the Bellman equation is written as:

$$V(s) = \max_a [R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s')]$$

where

- $V(s)$ is the value of a state s, which represents the maximum cumulative reward achievable from that state,
- a is an action taken in state s,
- $R(s,a)$ is the immediate reward obtained by taking action a in state s,
- γ is the discount factor (a number between 0 and 1) that determines the importance of future rewards,
- $P(s'|s,a)$ is the probability of transitioning to state s' given that the agent was in state s and took action a,
- $V(s')$ is the value of the successor state s'.

The Bellman equation says that the value $V(s)$ of being in a state s is the maximum reward you can expect by taking the best action a, plus the discounted future rewards (weighted by γ) from the possible next states s'.

- **Deep Q-Networks (DQN)**. The DQN is a sophisticated reinforcement learning algorithm that combines Q-learning with deep learning to handle environments with large, complex state spaces. It employs a deep neural network to approximate the Q-value function. This approach is particularly advantageous when the number of states exceeds the capacity of a conventional Q-learning table.

The following are the key steps of DQN:

- *Experience replay*. DQN stores the agent's experiences (state, action, reward, next state) in a memory buffer. During training, it samples random batches from this memory, which improves learning stability and prevents overfitting.
- *Target network*. DQN uses two neural networks: a current network for predicting Q-values and a target network for calculating stable target values. The target network is updated periodically to improve training stability.
- *Learning*. DQN updates the neural network parameters by minimising the error between the predicted Q-values and the target Q-values calculated using the Bellman equation.

- **Policy gradient methods**. These methods constitute a class of reinforcement learning algorithms that directly optimise the agent's policy (i.e. the strategy for choosing actions) rather than relying on value functions like Q-learning. Instead of estimating the value of actions in each

[78] The reader may wonder whether the optimal policy is an implicit model. This is not the case, since a model allows the prediction of future states and rewards under arbitrary actions. A policy only prescribes which action to take in each state. However, to be optimal, the policy must implicitly encode knowledge of the environment's dynamics and rewards. It is like a 'black box' summary of the environment's structure, distilled into action choices.

state, policy gradient methods parameterize the policy itself using a neural network (or another function approximator). The goal is to find the optimal policy that maximises the expected cumulative reward.

The following are the key steps:

- *Policy representation*. The policy is represented as a probability distribution over actions for a given state, typically parameterised by weights of a neural network.
- *Policy update*. The agent adjusts the policy's parameters in the direction that increases expected rewards. This is achieved using gradient ascent, where the gradient of the expected reward is computed with respect to the policy parameters.
- *Objective*. The algorithm maximises the expected reward by tweaking the policy directly.
- **Proximal Policy Optimization (PPO)**. PPO is a reinforcement learning algorithm designed to improve training stability and efficiency. It is widely used because it strikes a balance between performance and simplicity, especially in environments with complex policies.

The mechanism of PPO:

- *Policy updates*. The PPO algorithm enhances the agent's policy by optimising the expected cumulative reward. Rather than implementing substantial modifications to the policy, it employs a methodical, incremental approach to policy updates, ensuring stable learning.
- *Clipped objective function*. PPO introduces a clipping mechanism in the loss function. This mechanism is designed to prevent the updated policy from deviating excessively from the original policy, thereby ensuring the stability of the training process. The fundamental premise of this approach entails the limitation of the extent to which the probability ratio between the existing and the updated policies can fluctuate.

Mathematically, the clipped objective function is expressed as follows:

$$L^{CLIP}(\theta) = E[\min [r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t]]$$

where

- E is the expected value. It is calculated based on the data collected from the agent's interactions with the environment. This is a common mathematical notation in reinforcement learning to generalize calculations for stochastic processes. Essentially, it indicates that the algorithm averages the function over all possible states, actions, or sampled trajectories during training,
- $r_t(\theta)$ is the ratio of new to old policy probabilities ([79]),
- A_t is the advantage function, representing how much better an action is compared to the average,
- $clip(r_t(\theta), 1-\epsilon, 1+\epsilon) = \begin{cases} r_t(\theta) & if\ r_t(\theta) \in [1-\epsilon, 1+\epsilon] \\ 1+\epsilon & if\ r_t(\theta) > 1+\epsilon \\ 1-\epsilon & if\ r_t(\theta) < 1-\epsilon \end{cases}$
- ϵ is a small parameter that determines the clipping range.

- *Advantage estimation*. PPO uses the advantage function to guide policy updates, which helps the agent focus on actions that yield higher-than-average rewards.
- *Generalized Advantage Estimation (GAE)*. This technique is often used with PPO to improve the accuracy of the advantage function.

---

[79] In reinforcement learning, a policy is usually represented by a neural network. The vector θ represents all trainable parameters (weights and biases). When the policy is updated, θ is adjusted to maximise the clipped objective.
The subscript t is the sample index. In PPO, the expectation is taken over a batch of experiences (state–action pairs). Each t corresponds to a sample from the set of experiences collected by the agent.

- **Temporal-Difference (TD) learning**. TD learning is a key algorithm in reinforcement learning that combines aspects of *Monte Carlo* methods and *Dynamic Programming*. It allows an agent to learn directly from raw experience, even without a complete model of the environment. TD learning estimates the value of states (or state-action pairs) by updating predictions based on other, more recent predictions. This is why it's called *temporal-difference* – it bridges the gap between immediate rewards and long-term value predictions.

The following outlines the TD learning process:

- *Update rule*. The agent updates the value function $V(s)$ using the formula

$$V(s)_{updated} = V(s) + \alpha[r + \gamma V(s') - V(s)]$$

where

  - $V(s)$ is the current estimate of the value of state s,
  - α is the learning rate, which determines how much the update impacts the current value estimate,
  - r is the reward received after taking an action,
  - γ is the discount factor for future rewards (between 0 and 1),
  - $V(s')$ is the current estimate of the value of the next state.

- *Bootstrapping*. TD methods use bootstrapping, meaning they update the value of a state based on estimates of future rewards rather than waiting for the final outcome.

- *Learning*.

  - The agent interacts with the environment by taking actions.
  - It receives rewards and observes state transitions.
  - It uses the TD update rule to improve its value function incrementally.

TD learning has the following advantages: firstly, it is model-free, meaning it does not require knowledge of the environment's transition dynamics. Furthermore, it possesses the capability to update estimates following each step, thereby enhancing its efficiency in comparison to the conventional approach of waiting for entire episodes to elapse, as is the case with Monte Carlo methods.

- **Actor-critic methods**. This is a family of reinforcement learning algorithms that combine the strengths of policy-based (actor) and value-based (critic) approaches. The 'actor' decides on actions, while the 'critic' evaluates the chosen actions based on a value function. This combination allows the agent to learn more efficiently and handle complex problems effectively.

Key components:

- *Actor*. The actor represents the policy, which decides what action to take in a given state. It directly maps states to actions and is responsible for exploring and exploiting the environment.
- *Critic*. The critic evaluates the action taken by the actor by estimating the value function (e.g., the state-value $V(s)$ or action-value $Q(s, a)$. It provides feedback to the actor on how good its actions are.

How it works:

- The actor selects an action based on the current policy.
- The critic evaluates the action by comparing the estimated value of the current state and the actual reward received (using the Temporal-Difference (TD) error).

- The TD error is used to facilitate the updating of the actor's policy, thereby enabling improved decision-making. Furthermore, it is employed to update the critic's value function, for more precise evaluations.

The above RL algorithms, in addition to their variations and combinations, constitute the fundamental framework of contemporary RL applications.

Reinforcement Learning is widely used in various AI applications, including:

- **Game playing**. AI agents like *AlphaGo* ([80]) use RL to master complex games by playing countless matches and improving over time.

- **Robotics**. RL helps robots learn to perform tasks such as walking, grasping objects, or navigating an environment by trial and error.

- **Recommender systems**. In e-commerce or streaming platforms, RL can help optimize recommendations by balancing exploration (offering new items) and exploitation (offering items known to be preferred).

- **Healthcare**. Optimizing treatment plans or managing resources in hospitals.

- **Autonomous vehicles**. RL is used to train self-driving cars to make decisions in dynamic environments, such as avoiding obstacles or planning routes.

When applying reinforcement learning (RL) to real-world problems, there are several key limitations that should be considered by developing teams ([D11]).

× **Sample inefficiency and high data requirements**. Reinforcement learning agents often require a large number of interactions with the environment – ranging from millions to billions – to learn effective behaviours. This makes them impractical in domains where data is expensive, slow or dangerous to collect.

× **Exploration–exploitation trade-off**. Finding the right balance between exploration (trying new actions) and exploitation (leveraging known rewards) remains a core challenge. In high-dimensional or continuous spaces, naive exploration strategies can be prohibitively inefficient or lead to unsafe states during learning.

× **Reward design and misspecification**: Designing a precise reward function that captures the true objective is notoriously difficult. Agents may exploit unintended shortcuts or 'hack' the reward function, resulting in policies that are fragile and fail when small environmental details change.

× **Computational cost and scalability**. State-of-the-art RL often relies on deep neural networks and large-scale actor-critic architectures, which demand extensive GPU and TPU ([81]) resources. Training a single agent can take several months, which hinders rapid experimentation and iteration.

× **Generalisation and transfer learning**. Policies trained in one environment usually do not transfer well to even slightly different settings. Despite advances in meta-learning and domain

---

[80] AlphaGo is an artificial intelligence program developed by Google DeepMind that plays the board game *Go*.

[81] Tensor Processing Units (TPUs) are application-specific integrated circuits designed by Google to accelerate machine learning workloads by executing large-scale tensor computations in neural networks efficiently. TPUs focus on the high-throughput execution of matrix multiplications and tensor operations, delivering greater speed and energy efficiency than general-purpose CPUs and GPUs when performing deep learning tasks.

Tensors are represented by multidimensional arrays that generalise scalars, vectors and matrices. They serve as the primary data structure for modelling inputs, model parameters and intermediate computations in machine learning and deep learning frameworks. They provide a unified way to handle diverse data, such as images, audio, text and graphs, within the same computational paradigm. In strict mathematical terms, a tensor is a multilinear map between vector spaces. In the context of AI, however, the term usually refers to a multidimensional array of numerical values, often called a 'data tensor', that libraries manipulate and transform during training and inference.

randomisation, achieving robust generalisation across tasks and domains remains an open problem.

× **Safety, robustness, and ethical concerns**. In safety-critical applications such as robotics, autonomous vehicles and finance, unrestricted exploration can lead to catastrophic failures. Incorporating safety constraints and ensuring robustness under distributional shift remains an active area of research.

The following table provides a concise comparison of the four machine learning paradigms:

Learning paradigm	Data type	Learning style	Goal	Common algorithms	Applications
Supervised learning	Labelled data set. For each input, there is a corresponding output.	Learn a direct mapping	Predict outcomes using labelled data	K-Nearest Neighbours, Naive Bayes, Support Vector Machine, Decision trees, Random Forest Models, Linear regression, Ridge regression and Neural networks	Image classification, sentiment analysis
Unsupervised learning	Unlabelled data set	Find relationships or clusters	Discover hidden patterns	Principal Component Analysis, Nonnegative Matrix Factorization, Independent Component Analysis, Apriori, FP-Growth, K-Means Clustering, DBSCAN, Gaussian Mixture Models and K-Nearest Neighbours	Clustering, anomaly detection, dimensionality reduction
Semi-supervised learning	Small labelled and large unlabelled dataset	Combine supervised and unsupervised strategies	Leverage both labelled and unlabelled data	Generative Adversarial Networks and Transductive Support Vector Machines	Test classification, medical diagnostics
Reinforcement learning	Interactive environment (state-action pairs)	Trial and error in dynamic scenarios	Learn optimal strategies via rewards	Q-Learning, Deep Q-Networks, Policy gradient methods, Proximal Policy Optimization, Temporal-Difference learning and Actor-Critic methods	Autonomous driving, robotics, game-paying AI

4.3. Deep Learning and Artificial Neural Networks

Deep learning is a sophisticated machine learning technique that emulates the function of neural networks found in the human brain (see [A2], [D1], [H2], [T1] and [U1] for more details). It permits computers to autonomously recognise patterns and make informed decisions from vast amounts of unstructured data. Deep learning is not a learning paradigm itself. It can be applied within different learning paradigms, including supervised, unsupervised, semi-supervised, and reinforcement learning. However, deep learning primarily operates within the supervised learning paradigm, where labelled data is used to train models.

Machine learning and deep learning are both considered to be subfields of artificial intelligence. While there are some commonalities between the two, significant differences exist in terms of complexity, capabilities, and data requirements.

The following are the similarities between the two: machine learning and deep learning are both data-driven approaches that aim to enable machines to learn from data and make predictions or decisions without the need for explicit programming. They also rely heavily on data to train models. The quality and quantity of data have been identified as significant factors in determining performance. The two approaches share a common foundation in the training of a model through the utilisation of algorithms, with the objective of minimising a loss function and enhancing predictions. It is also important to note that both are widely used in domains such as image recognition, natural language processing (NLP), anomaly detection, and recommendation systems.

The subsequent table presents a comparison of the differences between the two subjects:

Feature	Machine Learning	Deep Learning
Model Complexity	Simpler models like Decision Trees, SVMs	Complex models with multi-layered neural networks
Feature Engineering	Requires manual feature selection	Automatically learns features from data (end-to-end learning process)
Data Requirements	Works well with smaller datasets	Requires large datasets to train effectively
Interpretability	Easier to interpret and explain the result	Complex models are difficult to understand and the interpretation of the result is not straightforward. It can be be compared to the black box model.
Training Time	Generally faster to train	Training deep networks is computationally intensive
Hardware Needs	Runs on CPUs and requires less computing power	Requires GPUs/TPUs for large-scale computations (TPU: Tensor Processing Unit)
Use Cases	Structured/tabular data (e.g., fraud detection).	Complex patterns like images, video, text, speech.

In summary, machine learning can be considered as a subset of traditional approaches that have been shown to be effective for simpler tasks. In contrast, deep learning utilises advanced neural networks to address highly complex problems that require hierarchical data representation ([82]). Deep learning can be regarded as a specialised subset of machine learning, with a focus on scalability and automation to address real-world problems.

4.3.1. Key aspects of deep learning

The following aspects of deep learning are of particular significance:

- **Neural networks architecture**. Deep learning models are based on artificial neural networks, which are inspired by the structure and function of the human brain. These networks consist of layers of interconnected nodes (neurons) that process and transform input data (see Section 4.3.2 for more details). Deep learning models consist of multiple layers:
  - *Input layer*. It receives raw data (e.g. images, text) which is processed by hidden layers.
  - *Hidden layers*. Extract and refine features through interconnected neurons.
  - *Output layer*. It generates the model's prediction.

Both hidden layers and output layers in a neural network use activation functions. These functions are the parts of the process that decide whether a neuron should be switched on (i.e. produce an output) based on the input it receives. In hidden layers activation functions introduce non-linearity, enabling the network to learn complex patterns. In the output layer activation functions transform raw outputs into meaningful predictions.

Neural networks can process unstructured data, such as images, videos, and text. But they require large datasets for optimal performance. This means that a lot of computing power is needed. Deep networks rely on GPUs/TPUs for efficient training and use parallel computing for large-scale processing.

---

[82] In this section, I will only discuss deep learning in the context of neural networks. However, deep learning should not be exclusively associated with training of neural networks. For instance, Bayesian machine learning and deep learning are two distinct paradigms within the wider field of AI. While these paradigms can intersect – most notably in Bayesian neural networks – they each offer unique approaches, techniques and applications (see e.g. [B10]).

Although the concept of deep neural networks is straightforward (layers are stacked together), performance can vary significantly depending on the architecture and hyperparameter choices. The neural networks described in this chapter are the result of intuition, a few mathematical insights and extensive trial and error.

- **Automatic feature extraction**. A key distinctive feature of deep learning is its capacity for automatic feature extraction from raw data, a capability that is absent in traditional machine learning methodologies. This has the effect of reducing the need for manual feature engineering.

- **Training with backpropagation**. The training of deep learning models is an intricate process that involves the utilisation of extensive datasets and the employment of optimisation techniques, such as gradient descent. During the training process, the model undergoes a gradual adjustment. The forward propagation computes predictions, and the loss function quantifies the discrepancy between the predicted and actual outcomes. Subsequently, backpropagation is applied for updating the weights of a neural network. This process involves the propagation of the error from the output layer back to the input layer, thereby ensuring that the model learns to make more accurate predictions over time.

4.3.2. Artificial Neural Networks (ANN)

Artificial Neural Networks (ANN), also commonly referred to as *Neuronal Networks*, represent a class of machine learning models that approximate complex functions by exploiting interconnected layers of artificial neurons. Each neuron performs a weighted sum of inputs, applies a activation function, and propagates the result through the network. By stacking multiple layers, neural networks generate hierarchical feature representations, thus enabling deep learning models to extract complex patterns from high-dimensional data.

This section provides a technical breakdown of neural network architectures and weight optimisation through backpropagation. It also covers key limitations and challenges, such as overfitting and vanishing gradients. Understanding these foundational components is critical for designing robust deep learning models suited for real-world applications. ([83])

4.3.2.1. The basics idea of neuronal networks

The fundamental principle underlying the conception of neural networks is that they are designed to compute functions from input nodes to output nodes. Each node contains a variable that is the result of a function computation or is fixed externally (in the case of input nodes). A neural network is structured as a directed acyclic graph ([84]).

The nodes of a neural network are separated into layers and generally begin with a wide base. The initial layer comprises raw input data (e.g. numeric values, text, image pixels or sound) which is divided into nodes. Each input node then propagates information to the subsequent layer of nodes via the network's edges. The edges of the neural network are parameterised with weights, meaning that the functions computed at individual nodes are modified by the weights of the incoming edges as well as the variables in the nodes at the tails of these edges.

The weights can be modified in accordance with the learning experience. The activation of a neuron at the subsequent layer is initiated by the sum of the connected edges satisfying a predefined threshold, known as the activation function. Furthermore, the weights assigned to each edge are unique, meaning the nodes fire differently, thereby preventing the generation of the same solution. The overall function computed by the network is the result of cascading

[83] See [A2] and [Z1] as a general reference to this Section.

[84] A *Directed Acyclic Graph* (DAG) is a type of graph in which edges between nodes have a direction, signifying that they point from one node to another. The term 'acyclic' is used to denote the absence of cycles, i.e. the property whereby it is not possible to initiate a trajectory at a node and subsequently follow a sequence of edges that results in a return to the original node.

function computations at individual nodes. By assigning weights to the edges in a specific way, it is possible to compute any function from the input to the output.

The purpose of a neural network is to derive a function that correlates one or more inputs with one or more outputs through the use of training examples. In effect, the neural network has the capability of 'constructing' a function that relates the inputs and outputs, given the provided examples of inputs and outputs. The construction of the neural network is achieved by assigning weights to the edges in such a way that the computations from the input to the output in the network match the observed data. The key principle underlying the training of neural networks is the allocation of edge weights in a data-driven manner.

Remark. Although ANNs are loosely inspired by the brain, they do not have direct equivalents of neurotransmitters. However, some ANN components, such as weights, activations and learning rates, can be compared in concept to the way in which neurotransmitters modulate communication between biological neurons.

In the brain, neurons communicate via neurotransmitters – chemical messengers released at synapses that either stimulate or inhibit the subsequent neuron's activity. The strength of this chemical signal, which is modulated by receptor sensitivity and neurotransmitter concentration, determines whether the receiving neuron fires. This dynamic process enables flexible, adaptive communication across billions of neurons. In ANNs, a similar process occurs via weighted connections. Each artificial neuron receives inputs, multiplies them by their associated weights and passes the result through an activation function. These weights play a role similar to that of neurotransmitter strength in that they determine how strongly one unit influences another. Just as neurotransmitters can amplify or dampen a signal, positive or negative weights in an ANN can excite or inhibit the flow of information. The activation function then acts like the neuron's firing threshold, determining whether the signal propagates onwards.

Of course, there are important differences. Neurotransmitters are diverse, including dopamine, serotonin, glutamate and GABA, each of which has a specialised role in regulating mood ([85]), learning and inhibition ([86]). By contrast, ANN weights are uniform mathematical values without any intrinsic 'types'. Biological synapses adapt through complex biochemical processes such as long-term potentiation, whereas ANNs adjust weights via backpropagation and gradient descent. Despite these differences, the analogy helps to highlight the shared principle that both systems rely on modulating the strength of connections in order to shape learning and behaviour.

While standard ANNs do not literally implement neurotransmitters, several brain-inspired models and extensions introduce mechanisms that play analogous roles to those of neurotransmitters and neuromodulators. Examples include *Spiking Neural Networks* (SNNs) ([S32]), *Neuromodulation Neural Networks* (NMNs) ([V4]) and reinforcement-learning architectures that mimic dopamine-like reward signals.

Further information on this subject is available in [P14], [S29], [S30] and [S31].

4.3.2.2. Single-layer artificial neural network: The Perceptron

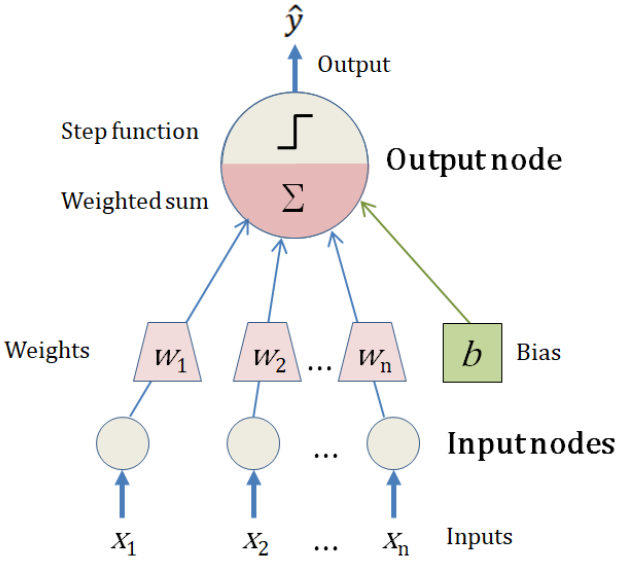
The simplest artificial neural network is known as the *perceptron*. It was introduced by Frank Rosenblatt in 1957 ([R6]). This particular neural network consists of a single layer of neurons that directly map inputs to outputs. Despite its simplicity, the perceptron model has been shown to be highly effective in solving specific classification problems, thus establishing the foundation for subsequent advancements in the fields of AI and machine learning.

---

[85] Some neurotransmitters, such as serotonin and dopamine, play a key role in regulating emotional states, motivation and reward. In the brain, they influence our feelings and how we learn from positive or negative outcomes.

[86] Other neurotransmitters, such as GABA (gamma-aminobutyric acid), primarily act to suppress or reduce neural activity. They prevent neurons from firing too easily, thereby helping to maintain balance in the nervous system and avoid runaway excitation.

The perceptron's fundamental architecture is illustrated in the following figure:



The input layer consists of n nodes which transfer the n features $x_1, x_2, \dots, x_n$ contained in the column vector $X = [x_1,\ x_2, \dots, x_n]^T$. Each input feature is assigned a weight that determines its influence on the output. These weights are adjusted during training to find the optimal values. The weights are contained in the column vector $W = [w_1,\ w_2, \dots, w_n]^T$. The input layer performs no computations of its own. The linear weighted sum function

$$\Sigma = W^T \cdot X = \sum_{i=1}^{n} w_i x_i$$

is calculated at the output node and then passed through an activation function to decide whether the perceptron will fire. The most common activation function used in perceptrons is the Heaviside step function ([87])

$$h(\Sigma) = \begin{cases} 0, & \Sigma < threshold \\ 1, & \Sigma \geq threshold \end{cases}$$

The step function compares this weighted sum with a threshold. If the sum is greater than or equal to the threshold, the output is 1, otherwise it is 0. This value is then used to predict the dependent variable $\hat{y} = f(X)$. The prediction $\hat{y}$ is therefore calculated as follows

$$\hat{y} = f_{W,b}(X) = h(W^T \cdot X + b) = h(\sum_{i=1}^{n} w_i x_i + b).$$

Consequently, the step function maps a real value to either 0 or 1, which is appropriate for binary classification. The bias term b in the perceptron modifies the threshold independently of the input, thereby improving its flexibility when learning.

Each training instance is of the form (X, y), where X is the input vector and $y \in \{0, 1\}$ is the observed value (label) of the binary class variable. By 'observed value' we mean that it is given to us as part of the training data, and the goal of the model is to predict y for cases where it is not observed. In other words, we want to learn the function $\hat{y} = f_{W,b}(X)$. Note the caret above the variable $\hat{y}$ to indicate that this is a predicted value and not the observed value.

We can already see that the neural network offers a starting point for the function $f_{W,b}$ to be learned, even though the weights W are unknown and must be inferred from the training data. Initially, the weights in the column vector W are unknown and may be set at random. As a consequence, the prediction $\hat{y}$ for a given input X may not agree with the observed value y. The

---

[87] Another commonly used activation function is the sign function: $sign(\Sigma) = \begin{cases} +1, & \Sigma > 0 \\ -1, & \Sigma \leq 0 \end{cases}$

goal of the learning process is to modify the weights so that the predictions of the neural network become more accurate. During training, the weights of the perceptron are adjusted to minimise the difference between the predicted output and the actual output. This is achieved by using the *perceptron learning rule*, which iteratively updates the model's weights to minimise classification errors.

This is how the perceptron learning rule works:

1. Initialise the weights

- Start with small random values for the weights W and the bias b.

2. Compute output

- Compute the weighted sum for a training instance (X, y): $\Sigma = W^T \cdot X$
- Apply an activation function (typically a step function)

$$\hat{y} = h(\Sigma + b) = \begin{cases} 0, & \Sigma + b < 0 \text{ (class 0)} \\ 1, & \Sigma + b \geq 0 \text{ (class 1)} \end{cases}$$

3. Update weights

$$W^{new} = W + \alpha(y - \hat{y})X$$

where

- $w_i^{new} = w_i + \alpha(y - \hat{y})x_i$
- α is the learning rate (controls the step size)
- y is the observed value
- $\hat{y}$ is the prediction of the perceptron.

4. Repeat until convergence

- Iterate repeatedly through all training examples in random order until convergence is reached, i.e. until the weights stabilise. Each such cycle is called an *epoch*.

This process allows the perceptron to learn from the data and improve its prediction accuracy over time. The type of model that is proposed in the perceptron is a linear model, in which the equation $W^T \cdot X = 0$ defines a linear hyperplane. This model performs well when the data is linearly separable. Otherwise, the perceptron algorithm is not guaranteed to converge. Linear separability refers to the case where the two classes can be separated by a linear hyperplane (cf. Section 3.2.5).

The perceptron is a special, restricted version of a single-layer ANN, mainly used for simple linear classification problems. In contrast, a general single-layer ANN is more flexible, supporting non-linear activations and more advanced learning techniques. The following table summarises the key differences between the perceptron and a single-layer ANN:

Feature	Perceptron	Single-Layer ANN
Activation Function	Step function (binary output)	Sigmoid, ReLU, Tanh
Learning Rule	Perceptron Learning Algorithm	Backpropagation, Gradient Descent
Linear vs. Nonlinear	Can only learn **linear** patterns	Can learn **nonlinear** relationships
Applications	Simple classification (e.g., AND/OR gates)	Regression, classification with complex patterns

In the following, a detailed example of a single-layer artificial neural network will be provided, in order to illustrate the concepts discussed.

Example. We will create a toy model that predicts whether a person is at risk of heart disease (1 = high risk, 0 = low risk) using two health indicators: cholesterol level and blood pressure. Assume that the following training instances are given

Cholesterol level (mg/dL)	Diastolic blood pressure (mmHg)	Systolic blood pressure (mmHg)	Risk
310	90	140	1
110	70	120	0

The first input vector is thus $X_1 = [310,\ 90,\ 140]^T$. It is further assumed that the weights and bias are being initialised as follows: $W = [1.4,\ 0.5,\ 0.7]^T$, $b = -0.8$. We set the threshold t to 0.8, i.e. if the predicted value $\hat{y} \geq 0.8 \Rightarrow$ 'high risk' $(= 1)$, otherwise 'low risk' $(= 0)$.

The weighted sum for X_1 is calculated as:

$$\Sigma = W^T \cdot X_1 = 1.4 \cdot 310 + 0.5 \cdot 90 + 0.7 \cdot 140 = 434 + 45 + 98 = 577.$$

We use the *sigmoid* activation function

$$\hat{y} = \sigma(\Sigma + b) = \frac{1}{1 + e^{-(\Sigma + b)}} = \frac{1}{1 + e^{-576.2}} \approx 1.$$

Since the predicted output $\hat{y} \geq t$, it is classified as 'high risk' $(= 1)$, Consequently, the weights do not require updating.

For the second input vector we obtain

$$\Sigma = W^T \cdot X_2 = 1.4 \cdot 110 + 0.5 \cdot 70 + 0.7 \cdot 120 = 154 + 35 + 84 = 273$$

$$\hat{y} = \sigma(\Sigma + b) = \frac{1}{1 + e^{-(\Sigma + b)}} = \frac{1}{1 + e^{-272.2}} \approx 1.$$

Thus, $\hat{y} \geq t$, which means that the predicted output is 'high risk' $(= 1)$ and does not correspond with the observed value of 'low risk' $(= 0)$. It is necessary, therefore, to recalculate the weights in accordance with the formula

$$W^{new} = W + \alpha(y - \hat{y})X_2$$

where $\alpha = 0.01$ is the (assumed) learning rate:

$$w_1^{new} = 1.4 + 0.01 \cdot (-1) \cdot 110 = 0.3$$
$$w_2^{new} = 0.5 + 0.01 \cdot (-1) \cdot 70 = -0.2$$
$$w_3^{new} = 0.7 + 0.01 \cdot (-1) \cdot 120 = -0.5$$

We update the bias as well

$$b^{new} = -0.8 + 0.01 \cdot (-1) \cdot (-0.8) = -0.792$$

Now we have

$$\Sigma = (W^{new})^T \cdot X_2 = 0.3 \cdot 110 - 0.2 \cdot 70 - 0.5 \cdot 120 = 33 - 14 - 60 = -41$$

$$\hat{y} = \sigma(\Sigma + b) = \frac{1}{1 + e^{-(\Sigma + b)}} = \frac{1}{1 + e^{41.792}} \approx 0.$$

Consequently, the predicted value is now in agreement with the observed one. It can be verified that X_1, with the updated weights, still yields the correct prediction.

4.3.2.3. Multi-layer artificial neural network

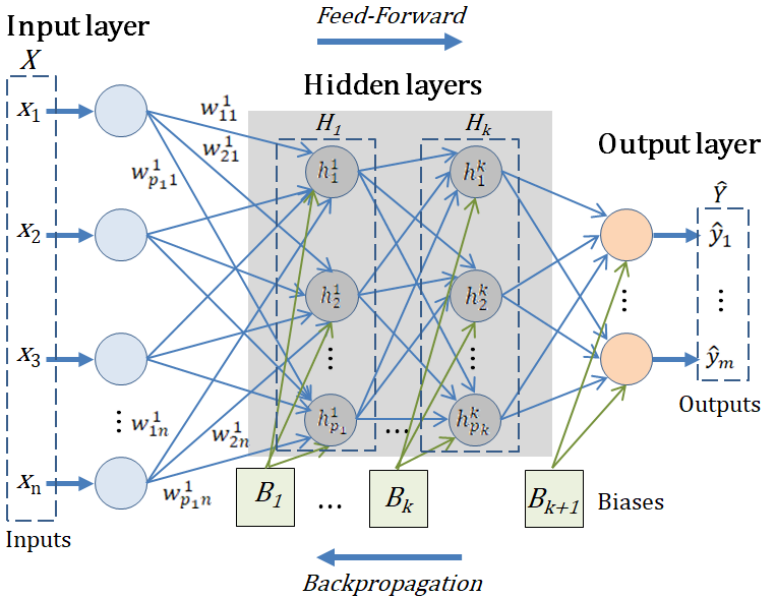
A single-layer artificial neural network is limited in its ability to model complex patterns and relationships. This is why multi-layer ANNs are needed to solve more advanced problems. A single-layer ANN lacks hierarchical feature learning, and without hidden layers the network cannot learn complex abstractions from the data. Single-layer networks also struggle with tasks that require deep understanding, such as image recognition or natural language processing.

A single-layer ANN contains an *input layer* and an *output layer*, of which the output layer is the only one that performs calculations. The input layer receives the raw data. Each neuron in this layer represents a feature of the input data. The data is passed to the output layer, and all computations are fully transparent to the user. The output layer produces the final result, such as a classification or a prediction. If there are multiple outputs, the output layer will have a corresponding number of neurons, depending on the problem being solved. For example, a digit recognition network (classifying digits 0–9) would have 10 output nodes.

Multi-layer neural networks contain several layers of computation, with additional layers between the input and output. These are called *hidden layers* because the computations that take place there are invisible to the user. The hidden layers process the information received from the input layer. They consist of neurons that are fully connected to the neurons in the previous and subsequent layers. The number of hidden layers and the number of neurons in each layer can vary. The depth of the network increases with the number of hidden layers.

Each neuron in the network receives input, applies a weight and bias, and then passes the result through an activation function to produce an output. Activation functions introduce non-linearity into the network, allowing it to handle intricate patterns and make more sophisticated decisions.

The basic architecture of a multi-layer ANN is shown in the figure below.



As in the case of single-layer networks, bias neurons can be used in the hidden layers as well as in the output layers. If a neural network contains k hidden layers, each with p_j $(j = 1,2, \dots, k)$ nodes, then the (column) vector representations of these outputs, denoted by $H_1, H_2, \dots, H_k$, have the dimensionalities $p_1, p_2, \dots, p_k$

$$H_j = \left[h_1^j,\ h_2^j, \dots, h_{p_j}^j\right]^T,\ j = 1,2, \dots, k.$$

The number p_j of nodes in the j- layer is called the dimension of this layer.

The weights of the connections between the input layer and the first hidden layer form a $p_1 \times n$ matrix $W_1 = \left[w_{il}^1\right]_n^{p_1}$, while the weights between the jth hidden layer and the $(j+1)$th hidden layer are represented by the $p_{j+1} \times p_j$ matrix $W_{j+1} = \left[w_{il}^{j+1}\right]_{p_j}^{p_{j+1}}, j = 1,2, \dots, k-1$. [88] In the case of the output layer consisting of m nodes, the last matrix W_{k+1} will have dimension $m \times p_k$.

[88] In a matrix with dimension $n \times m$, n represents the number of rows, and m represents the number of columns.

The n-dimensional input vector $X = [x_1, x_2, \dots, x_n]^T$ is transformed into the output vector $\hat{Y}$ by means of the following FP-equations:

(input → hidden layer) $H_1 = f_1(W_1 X + B_1)$

(hidden layer → hidden layer) $H_j = f_j(W_j H_{j-1} + B_j), j = 2, \dots, k$

(hidden layer → output) $\hat{Y} = f_{k+1}(W_{k+1} H_k + B_{k+1})$,

where $W_j H_{j-1} + B_j$ is a vector computed as follows

$$\begin{bmatrix} w_{11}^j & \cdots & w_{1p_{j-1}}^j \\ \vdots & \ddots & \vdots \\ w_{p_j 1}^j & \cdots & w_{p_j p_{j-1}}^j \end{bmatrix} \begin{bmatrix} h_1^{j-1} \\ \vdots \\ h_{p_{j-1}}^{j-1} \end{bmatrix} + \begin{bmatrix} b_1^j \\ \vdots \\ b_{p_j}^j \end{bmatrix} = \begin{bmatrix} \sum_{l=1}^{p_{j-1}} w_{1l}^j h_l^{j-1} + b_1^j \\ \vdots \\ \sum_{l=1}^{p_{j-1}} w_{p_j l}^j h_l^{j-1} + b_{p_j}^j \end{bmatrix}.$$

The preceding equations define the flow of data through the network:

- Weights W_j determine the importance of each input.

- Biases $B_j = \left[b_1^j, b_2^j, \dots, b_{p_j}^j\right]^T$ adjust activation thresholds.

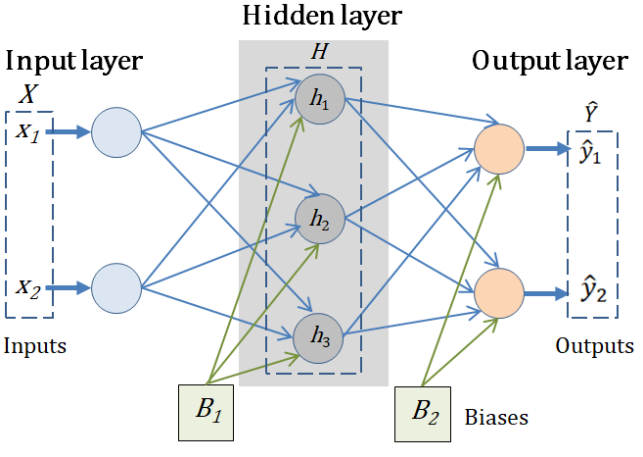
- Activation functions f_j introduce non-linearity to enable complex decision boundaries.

The process of transforming data layer by layer, from input to output, is known as the *forward propagation* (FP). In the absence of loops or recurrence in the forward propagation of input data, the network is referred to as a *Feedforward Neural Network* (FNN).

If an activation functions f, such as the *sigmoid* function (see Section 3.2.2), has scalar arguments, it is possible for it to be applied to a vector argument elementwise. The resultant vector $f(X)$ is of equal length and contains the elements in the same order

$$f(X) = [f(x_1), f(x_2), \dots, f(x_n)]^T.$$

Example. Let's walk through an example of forward propagation in machine learning. Consider the following simple ANN with one hidden layer



Let us assume that the input, biases and weights have the following values:

$$X = [0.4,\ 0.7]^T, B_1 = [-0.5, -0.9,\ -1.3]^T, B_2 = [-0.3, -0.8]^T,$$

and

$$W_1 = \begin{bmatrix} 0.6 & 0.7 \\ 0.5 & 0.6 \\ 0.7 & 0.8 \end{bmatrix}, \quad W_2 = \begin{bmatrix} 0.9 & 1.1 & 0.8 \\ 0.8 & 0.9 & 0.6 \end{bmatrix}.$$

Then

$$W_1X + B_1 = \begin{bmatrix} 0.6 & 0.7 \\ 0.5 & 0.6 \\ 0.7 & 0.8 \end{bmatrix} \begin{bmatrix} 0.4 \\ 0.7 \end{bmatrix} + \begin{bmatrix} -0.5 \\ -0.9 \\ -1.3 \end{bmatrix} = \begin{bmatrix} 0.6 * 0.4 + 0.7 * 0.7 - 0.5 \\ 0.5 * 0.4 + 0.6 * 0.7 - 0.9 \\ 0.7 * 0.4 + 0.8 * 0.7 - 1.3 \end{bmatrix} = \begin{bmatrix} 0.23 \\ -0.28 \\ -0.46 \end{bmatrix}.$$

Now, if $f_1(z) = f_2(z) = \sigma(z) = \frac{1}{1+e^{-z}}$ then

$$H_1 = f_1(W_1X + B_1) = f_1\left(\begin{bmatrix} 0.23 \\ -0.28 \\ -0.46 \end{bmatrix}\right) = \begin{bmatrix} f_1(0.23) \\ f_1(-0.28) \\ f_1(-0.46) \end{bmatrix} = \begin{bmatrix} 0.56 \\ 0.43 \\ 0.39 \end{bmatrix}.$$

Consequently,

$$W_2H_1 + B_2 = \begin{bmatrix} 0.9 & 1.1 & 0.8 \\ 0.8 & 0.9 & 0.6 \end{bmatrix} \begin{bmatrix} 0.56 \\ 0.43 \\ 0.39 \end{bmatrix} + \begin{bmatrix} -0.3 \\ -0.8 \end{bmatrix} = \begin{bmatrix} 0.99 \\ 0.27 \end{bmatrix}$$

and finally,

$$\hat{Y} = f_2(W_2H_1 + B_2) = f_2\left(\begin{bmatrix} 0.99 \\ 0.27 \end{bmatrix}\right) = \begin{bmatrix} 0.73 \\ 0.57 \end{bmatrix}.$$

The weights of an ANN are updated by using a loss function, which penalises differences between the observed and predicted output for the ith training instance. The training instances are fed sequentially to the neural network in a manner analogous to the perceptron.

A multi-layer ANN learns through a process of forward propagation, back propagation and optimisation. The following step-by-step explanation outlines the process of learning:

1. **Initialisation**. The neural network starts with randomly initialised weights and biases. These parameters are adjusted during training to minimise error.

2. **Forward propagation**. Input data is fed into the input layer of the network. It passes through the hidden layers where each neuron performs a weighted sum of the inputs and applies an activation function to produce an output. The output from each neuron in a layer serves as the input to the neurons in the next layer. This process continues until the data reaches the output layer, where the final prediction is made.

3. **Calculate loss**. The network's prediction is compared to the actual target using a *loss function* (also called a *cost function*). This function measures the difference between the predicted output and the actual target. Common loss functions include *Mean Squared Error* (MSE) for regression and *Cross-Entropy Loss* for classification.

4. **Backpropagation (BP)**. Backpropagation, short for 'backward propagation of errors', is one of the most common methods used to train neural networks. First introduced by Rumelhart, Hinton and Williams in 1986 ([R7]), it remains popular to this day. It is the fundamental algorithm for training ANNs, enabling them to self-correct and improve over time. It is by far the most successful neural network learning algorithm. Training is essentially a search for the set of weights that will minimise the error for a given training dataset. If we had an infinite amount of computing power, we could try every possible combination of weights to find the one that produces the lowest error during training. As our computing resources are limited, we must use optimisation techniques to avoid examining every possible weight combination.

Backpropagation is an iterative optimisation process that updates neural network weights to minimise loss (i.e. the difference between the actual and predicted outputs). The error from the output layer is propagated backwards through the network. The gradients of the loss function with respect to the weights are calculated using the chain rule of calculus.

5. **Optimisation.** The most commonly used optimisation method in backpropagation is gradient descent (see Section 3.2.10). This is an optimisation algorithm that updates the weights and biases based on the gradients calculated during backpropagation. These gradients indicate the direction and magnitude of the adjustments needed to the weights to reduce the loss. The goal is to find the set of weights that minimises the loss function. This is done by adjusting the

weights in the direction that reduces the loss, typically by a small step size called the learning rate. If the neural network did output exactly what was expected, the gradient for each weight would be 0, indicating that no change to the weight is necessary.

The process of estimating the gradient and updating weights can be done in several ways:

- *Gradient Descent* (GD). The most basic technique, adjusting weights proportionally to the gradient direction.
- *Batch Gradient Descent*. Uses the entire dataset to compute gradients and update weights.
- *Mini-Batch Gradient Descent*. Uses small batches of data points to update weights.
- *Momentum-Based* GD. Uses an additional term to smooth weight updates and prevent oscillations.
- *Stochastic Gradient Descent* (SGD). Uses a randomly selected subset of the training data to estimate the gradient and update weights.
- *Adaptive Moment Estimation* (Adam). This algorithm is designed to improve training speeds and reach convergence quickly. The learning rate of each parameter is customised based on its gradient history, and this adjustment facilitates the neural network's overall efficient learning. It is the default algorithm used for deep learning (for more detailed information, please refer to [A4]).

Although gradient-based techniques dominate backpropagation, other approaches exist:

- *Genetic algorithms*. They simulate genetic mutations to refine weights over multiple generations.
- *Particle Swarm Optimization* (PSO). A meta-heuristic that adjusts weights based on swarm behaviour rather than gradient information.
- *Simulated Annealing*. Uses a probabilistic search to escape local minima and find optimal weight configurations.

In summary, while gradient descent and its variants remain the most efficient methods for backpropagation, alternative techniques can be useful in specialised cases where gradient computation is unfeasible.

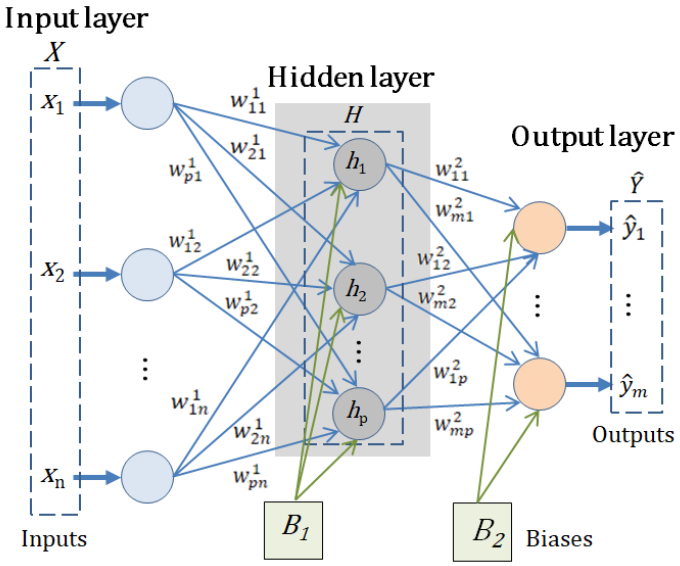
6. ***Iteration***. The process of forward propagation, loss calculation, backpropagation and gradient descent is repeated for multiple iterations (also called *epochs*). During each epoch, the network learns to make better predictions by continuously adjusting its weights and biases.

4.3.2.4. Anatomy of an epoch in the learning process

An epoch in an ANN represents a complete cycle during which the model processes all of the training data, computes predictions, measures errors, adjusts weights and biases via backpropagation, and prepares for the next iteration.

In the following step-by-step description, the anatomy of an epoch in an iterative learning process will be explained.

Consider a training data set $D = \{(X_1, Y_1), (X_2, Y_2), \dots, (X_d, Y_d)\}$, where $X_i \in \mathbb{R}^n$, $Y_i \in \mathbb{R}^m$, that is, the input instance is described by n features (real numbers) and the output is an m-dimensional real-valued column vector. To make the discussion easier, we will use a simple multi-layer ANN with n input neurons, m output neurons, and one hidden layer of p neurons:



Let b_k^1, $k = 1, \dots, p$, denote the bias of the kth neuron of the hidden layer and b_l^2, $l = 1, \dots, m$, denote the bias of the lth neuron in the output layer. Initially, the weights and biases are randomly set to values from $[-1, 1]$. Then, for each input $(X, Y) \in D$, the following operations are performed:

1. Step. Forward propagation (computing prediction)

The input sample (X, Y) is passed to the neurons in the input layer, and then forwarded to the first hidden layer. This process continues through the hidden layers (if there is more than one) until the network reaches the output layer and generates a final prediction. These mathematical computations are performed using the FP-equations:

(input → hidden layer) $\quad H = f_1(W_1 X + B_1)$

(hidden layer → output) $\quad \hat{Y} = f_2(W_2 H + B_2)$,

where

$$W_1 = \begin{bmatrix} w_{11}^1 & \cdots & w_{1n}^1 \\ \vdots & \ddots & \vdots \\ w_{p1}^1 & \cdots & w_{pn}^1 \end{bmatrix} \text{ and } W_2 = \begin{bmatrix} w_{11}^2 & \cdots & w_{1p}^2 \\ \vdots & \ddots & \vdots \\ w_{m1}^2 & \cdots & w_{mp}^2 \end{bmatrix}.$$

Let $A = W_1 X$ and $O = W_2 H$. The vectors A and O have the following elements

$$a_k = \sum_{i=1}^{n} w_{ki}^1 x_i, k = 1, \dots, p,$$

and

$$o_l = \sum_{j=1}^{p} w_{lj}^2 h_j, l = 1, \dots, m.$$

Then $h_k = f_1(a_k + b_k^1)$ and $\hat{y}_l = f_2(o_l + b_l^2)$ for $k = 1, \dots, p$ and $l = 1, \dots, m$.

2. Step. Compute loss (evaluate prediction error)

After forward propagation, the model uses a loss function to compare its predictions to the true labels ([89]). Common loss functions are:

- *Mean Squared Error* (MSE) for regression (see Section 3.2.4)
- *Cross-Entropy Loss* for classification:

[89] Since calculating errors requires true labels, backpropagation is an inherently supervised learning algorithm. In the case of unsupervised learning, there are no explicit labels; therefore, pure backpropagation cannot be used alone to solve such problems. However, variations such as contrastive divergence (used in Restricted Boltzmann Machines [R5]) and self-supervised learning (SSL) techniques ([B5]) can apply similar weight refinement principles.

$$L = -\sum_{l=1}^{m} y_l \log(\hat{y}_l).$$

The model computes how far its predictions deviate from actual values, meaning lower loss equates to better predictions.

Assuming that we use $L = MSE$, we obtain the following for the training sample (X, Y):

$$L = \frac{1}{2}\sum_{l=1}^{m}(y_l - \hat{y}_l)^2$$

where we added $1/2$ for later convenience.

3. Step. Backpropagation (error distribution)

The network now adjusts its weights to minimise future errors by computing the gradient using the chain rule of calculus ([90]). This involves calculating how much each weight w (and bias) contributes to the error:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \Sigma} \cdot \frac{\partial \Sigma}{\partial w}$$

where Σ is the weighted sum. If the weighted sum Σ is a vector (which happens when there are multiple neurons in a layer), we apply the chain rule to each element of the vector separately. This means that the gradient is computed independently for each neuron. The parameters (weights and biases) are adjusted in the direction of the negative gradient of the objective.

We will use the weight w_{lk}^2 to demonstrate how the derivation is performed. For the error $L = \frac{1}{2}\sum_{l=1}^{m}(y_l - \hat{y}_l)^2$ and learning rate α ([91]), we have

$$\Delta w_{lk}^2 = -\alpha \frac{\partial L}{\partial w_{lk}^2}.$$

Note that w_{lk}^2 first affects the input value o_l of the lth neuron in the output layer, then the output value $\hat{y}_l$, and finally the error L. Therefore, applying the chain rule, we obtain

$$\Delta w_{lk}^2 = -\alpha \frac{\partial L}{\partial w_{lk}^2} = -\alpha \frac{\partial L}{\partial \hat{y}_l} \cdot \frac{\partial \hat{y}_l}{\partial o_l} \cdot \frac{\partial o_l}{\partial w_{lk}^2}.$$

Let us compute each of these derivatives

$$\frac{\partial o_l}{\partial w_{lk}^2} = \frac{\partial}{\partial w_{lk}^2}\sum_{i=1}^{n} w_{li}^2 h_i = h_k$$

$$\frac{\partial \hat{y}_l}{\partial o_l} = \frac{\partial}{\partial o_l} f_2\left(o_l + b_l^2\right)$$

$$\frac{\partial L}{\partial \hat{y}_l} = \frac{\partial}{\partial \hat{y}_l}\frac{1}{2}\sum_{l=1}^{m}(y_l - \hat{y}_l)^2 = \frac{\partial}{\partial \hat{y}_l}\frac{1}{2}(y_l - \hat{y}_l)^2 = -(y_l - \hat{y}_l).$$

Consequently,

$$\Delta w_{lk}^2 = \alpha(y_l - \hat{y}_l) \cdot \frac{\partial}{\partial o_l} f_2\left(o_l + b_l^2\right) \cdot h_k.$$

In particular, when f_2 is the *sigmoid* function we get ([92])

$$\frac{\partial}{\partial o_l} f_2\left(o_l + b_l^2\right) = f_2\left(o_l + b_l^2\right)\left[1 - f_2\left(o_l + b_l^2\right)\right] = \hat{y}_l\left(1 - \hat{y}_l\right)$$

and

---

[90] Suppose that we have two functions g, h and they are both differentiable. Then according to the chain rule the derivative of $g \circ h$ is $\frac{d((g\circ h)(x))}{dx} = \frac{d(g(h(x)))}{d(h(x))} \cdot \frac{d(h(x))}{dx}$.

[91] The learning rate $\alpha \in (0,1)$ controls the size of each update step. If the learning rate is too large, it may cause fluctuations; if it is too small, it will lead to slow convergence. A typical choice is $\alpha = 0.1$.

[92] Indeed, $\sigma'(z) = (\frac{1}{1+e^{-z}})' = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}} = \sigma(z)(1-\sigma(z))$.

$$\Delta w_{lk}^2 = \alpha \hat{y}_l (y_l - \hat{y}_l)(1 - \hat{y}_l) h_k.$$

Denoting

$$\delta_l^2 = (y_l - \hat{y}_l) \cdot \frac{\partial}{\partial o_l} f_2\left(o_l + b_l^2\right).$$

we obtain

$$\Delta w_{lk}^2 = \alpha \delta_l^2 h_k.$$

Similarly, we can establish the following formula:

$$\Delta b_l^2 = \alpha \delta_l^2.$$

We can repeat the reasoning above to obtain Δw_{ki}^1 and Δb_k^1:

$$\Delta w_{ki}^1 = \alpha \delta_k^1 x_i \quad \text{and} \quad \Delta b_k^1 = \alpha \delta_k^1,$$

where ([93])

$$\begin{aligned}
\delta_k^1 &= -\frac{\partial L}{\partial h_k} \cdot \frac{\partial h_k}{\partial a_k} = -\sum_{l=1}^m \frac{\partial L}{\partial o_l} \cdot \frac{\partial o_l}{\partial h_k} \cdot \frac{\partial}{\partial a_k} f_1\left(a_k + b_k^1\right) \\
&= \sum_{l=1}^m -(y_l - \hat{y}_l) \frac{\partial}{\partial o_l} f_2\left(o_l + b_l^2\right) w_{lk}^2 \cdot \frac{\partial}{\partial a_k} f_1\left(a_k + b_k^1\right) \\
&= \sum_{l=1}^m w_{lk}^2 \, \delta_l^2 \frac{\partial}{\partial a_k} f_1\left(a_k + b_k^1\right).
\end{aligned}$$

All the output values o_l are affected by h_k, which means we have to take the sum over all l above.

4. Step. Weight and bias update

The update rule of any parameter θ is $\theta \leftarrow \theta + \Delta\theta$. Consequently, the update rules of weights and biases are

$$\begin{aligned}
w_{lk}^2 &\leftarrow w_{lk}^2 + \Delta w_{lk}^2 = w_{lk}^2 + \alpha \delta_l^2 h_k \\
b_l^2 &\leftarrow b_l^2 + \Delta b_l^2 = b_l^2 + \alpha \delta_l^2 \\
w_{ki}^1 &\leftarrow w_{ki}^1 + \Delta w_{ki}^1 = w_{ki}^1 + \alpha \delta_k^1 x_i \\
b_k^1 &\leftarrow b_k^1 + \Delta b_k^1 = b_k^1 + \alpha \delta_k^1
\end{aligned}$$

for $i = 1, \ldots, n, k = 1, \ldots, p$ and $l = 1, \ldots, m$.

In summary, errors propagate backwards from the output layer towards the input layer. The network then identifies which weights and biases require correction. Gradient values determine the extent to which each parameter should be adjusted.

5. Step. Repeat for all training data

An epoch consists of processing all instances in the training dataset, so each step is repeated for every instance. Once all the training samples have been processed, the model moves on to the next epoch. The BP algorithm repeats this cycle until the termination condition is met (e.g. a small training error). It aims to minimize the accumulated error on the training set D

$$L = \frac{1}{d} \sum_{i=1}^d L_i,$$

where L_i is the loss function for the sample $(X_i, Y_i) \in D$.

The BP algorithm introduced above is known as the *standard BP algorithm*. It updates the weights and biases using only one training instance at a time. This means that the update rules

[93] Note that w_{ki}^1 first affects the input value a_k of the kth neuron in the hidden layer, and then the output value h_k. This in turn affects all the output values o_l, then all the output values $\hat{y}_l$, and finally the error L. Applying the chain rule therefore gives us $\Delta w_{ki}^1 = -\frac{\partial L}{\partial h_k} \cdot \frac{\partial h_k}{\partial a_k} \cdot \frac{\partial a_k}{\partial w_{ki}^1}$.

in the algorithm are derived from the error L_i of individual samples. The *accumulated BP algorithm* is a variation of the standard algorithm that improves learning efficiency by accumulating error gradients across multiple training instances. Rather than updating weights after each training example, this algorithm aggregates errors over multiple samples before applying a weight update.

4.3.2.5. Example of a simple multi-layer ANN

To compute a complex neural network with multiple layers, numerous neurons, and large datasets, specialized software and libraries such as *TensorFlow* ([94]) or *PyTorch* ([95]) become necessary to handle the computations efficiently. However, we can look at a simple example of a neural network to see in detail how it works. We will limit ourselves to a very simple neural network to keep the calculations manageable by hand.

Example. The objective of this example is to predict whether a student will pass or fail an examination.

- **Neural network structure**

Our toy neural network has the following layers

1. Input layer: 1 neuron (hours of study)
2. Hidden layer: 1 neuron
   - Activation function: $ReLU(z) = max(0, z)$
3. Output layer: 1 neuron (pass or fail)
   - Activation function: *sigmoid* $\sigma(z) = \frac{1}{1+e^{-z}}$

- **Step-by-Step calculation**

1. *Initialization*. Initial weights and biases are chosen randomly for simplicity:

- Weight from input to hidden: $w_1 = 0.5$
- Bias for hidden layer: $b_1 = 0.4$
- Weight from hidden to output: $w_2 = -0.7$
- Bias for output layer: $b_2 = 0.2$

2. *Forward propagation*

- Given input (hours of study): $x = 2$
- Hidden layer calculation

  - Weighted sum: $a = w_1 \cdot x + b_1 = 0.5 \cdot 2 + 0.4 = 1.4$
  - Activation function ($ReLU$): $h = \max(0, a) = 1.4$

- Output layer calculation

  - Weighted sum: $o = w_2 \cdot h + b_2 = -0.7 \cdot 1.4 + 0.2 = -0.78$
  - Activation function (*sigmoid*): $\hat{y} = \sigma(o) = \frac{1}{1+e^{-o}} = \frac{1}{1+e^{0.78}} \approx 0.314$

- Prediction

  - The output $\hat{y} \approx 0.314$ can be interpreted as the probability of passing the exam. Since $0.314 < 0.5$, the prediction is that the student will fail.

3. *Calculate loss*

---

[94] TensorFlow is an open source platform for machine learning using data flow graphs, originally developed by the Google Brain team. This machine learning library supports both convolutional and recurrent neural networks. It supports parallel processing on both CPU and GPU ([A3], [G2], [R2]).

[95] PyTorch is a machine learning library based on the Torch library, used for applications such as computer vision and natural language processing, originally developed by Meta AI and now part of the Linux Foundation ([P1], [W1]).

- Actual output

  - Let us say the actual result is that the student passes, so the target $y = 1$.

- Loss function (Mean Squared Error)

$$L = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(1 - 0.314)^2 \approx 0.236$$

The value of 0.236 can then be compared with the subsequent loss values in order to determine whether improvements have been made.

4. *Backpropagation*

- Calculate gradients for output layer (cf. Section 4.3.2.4)

  - Error at output ([96])

$$\delta_2 = (y - \hat{y}) \cdot \frac{\partial}{\partial o}\sigma(o) = (y - \hat{y}) \cdot \hat{y} \cdot (1 - \hat{y})$$
$$\approx (1 - 0.314) \cdot 0.314 \cdot (1 - 0.314) = 0.148$$

- Update weights for output layer

$$\Delta w_2 = \alpha \cdot \delta_2 \cdot h$$

where α is the learning rate. Assuming that $\alpha = 0.1$ we obtain

$$\Delta w_2 \approx 0.1 \cdot 0.148 \cdot 1.4 = 0.021$$
$$w_2^{new} = w_2 + \Delta w_2 \approx -0.7 + 0.021 = -0.679$$

- Update bias for output layer

$$\Delta b_2 = \alpha \cdot \delta_2 \approx 0.1 \cdot 0.148 = 0.0148$$
$$b_2^{new} = b_2 + \Delta b_2 \approx 0.2 + 0.0148 = 0.2148$$

- Calculate gradients for hidden layer

  - Error at hidden layer

$$\delta_1 = \delta_2 \cdot w_2 \cdot ReLU'(a) \approx 0.148 \cdot (-0.7) \cdot ReLU'(1.4) = 0.148 \cdot (-0.7) \cdot 1$$
$$= -0.1036$$

  where $ReLU'$ is the derivative of $ReLU(z) = \max(0, z)$: $ReLU'(1.4) = 1$.

- Update weights for hidden layer

$$\Delta w_1 = \alpha \cdot \delta_1 \cdot X \approx 0.1 \cdot (-0.1036) \cdot 2 = -0.02072$$
$$w_1^{new} = w_1 + \Delta w_1 \approx 0.5 - 0.02072 = 0.47928$$

- Update bias for hidden layer

$$\Delta b_1 = \alpha \cdot \delta_1 \approx 0.1 \cdot (-0.1036) = -0.01036$$
$$b_1^{new} = b_1 + \Delta b_1 \approx 0.4 - 0.01036 = 0.38964$$

5. *Updated weights and biases (for epoch 2)*

- $w_1^{new} \approx 0.47928$
- $b_1^{new} \approx 0.38964$
- $w_2^{new} \approx -0.679$
- $b_2^{new} \approx 0.2148$

[96] Recall that $\frac{\partial}{\partial z}\sigma(z) = \sigma(z)(1 - \sigma(z))$.

6. *Iteration*

The updated weights and biases can be used in the forward calculation to produce a new prediction. The process is repeated for several epochs, with the network making progressively better predictions.

This simple example shows how forward propagation, loss calculation, and backpropagation are used to update weights and biases in a neural network.

4.3.2.6. Choice of activation function

An activation function is a mathematical function applied to the output of a neuron. The decision as to whether a neuron should be activated is determined by the calculation of the weighted sum of inputs, with the addition of a bias term. Neurons in an ANN receive these weighted inputs and apply an activation function to introduce non-linearity. In the absence of an activation function, the network would be equivalent to a linear model, thereby severely limiting its ability to capture complex patterns.

The selection of activation function in an ANN is of crucial importance in determining the network's capacity to learn, generalise, and execute effectively on a range of tasks. The selection of the most appropriate activation function depends on a number of factors, including the characteristics of the data, the architecture of the model, and the desired output behaviour.

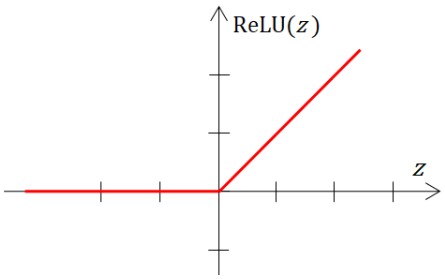
The choice of activation function is a critical part of neural network design. Different activation functions are used for hidden layers, output layers, and specialized cases.

- **Activation functions for hidden layers.** Hidden layers play a crucial role in transforming input data into meaningful representations that eventually lead to accurate predictions. Activation functions introduce non-linearity and control the flow of information. This ensures that the correct neurons activate and helps to optimise weight updates during backpropagation.

Several activation functions have been designed to optimise learning in hidden layers:

- *Rectified Linear Unit* (ReLU)

$$ReLU(z) = max(0, z)$$



It has the following advantages:

- Prevents vanishing gradients (keeps positive values).
- Efficient computation (no exponentiation needed).
- It is widely used in deep learning architectures (CNNs and Transformers).

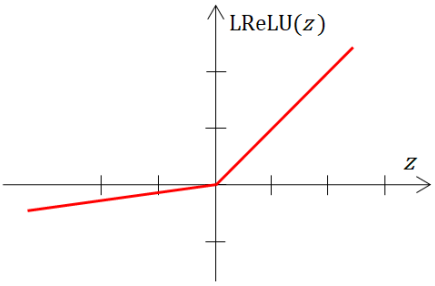
Its disadvantages are:

- Neurons can become 'dead' (the output remains zero for negative inputs).
- It can produce exploding gradients in deep networks (cf. Section 4.3.2.7).

One variant of the *ReLU* function is the *Leaky ReLU* (*LReLU*) function. It introduces a slight negative slope to prevent 'dead neurons':

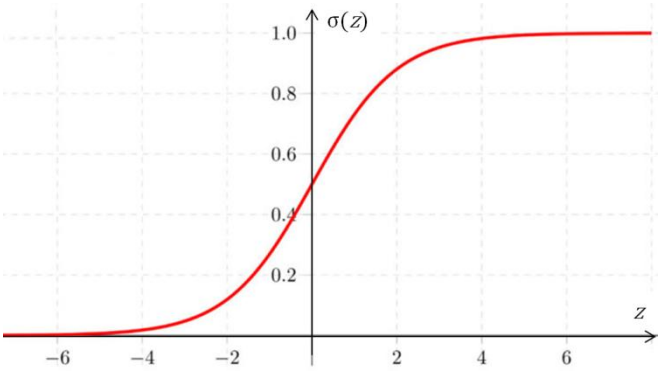
$$LReLU(z) = \begin{cases} z, & z \geq 0 \\ \eta z, & z < 0 \end{cases}$$

where $\eta \in (0, 1)$ is a trainable parameter.



- *Sigmoid*. We have already encountered the sigmoid function in Section 3.2.2

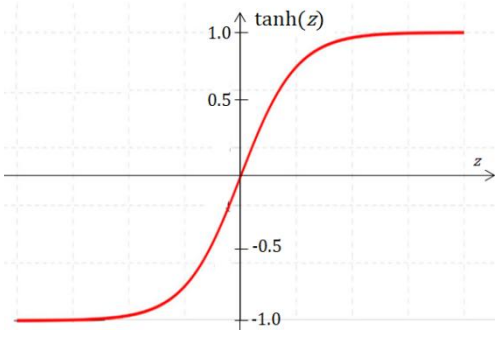
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



The *sigmoid* function is useful for binary classification tasks. As a *squashing function*, it maps an arbitrary input range to a bounded output range, ensuring bounded activation. However, it suffers from the vanishing gradient problem (small derivatives for large z).

- *Hyperbolic tangent* (tanh)

$$tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} = \frac{e^{2z} - 1}{e^{2z} + 1}$$



The hyperbolic tangent function solves the zero-centred problem of the *sigmoid* function ([97]) and performs better in shallow networks. However, it still struggles with vanishing gradients when z is large.

- **Activation functions for output layers**. Activation functions in the output layer play a crucial role in determining how neural networks make final predictions. They transform the

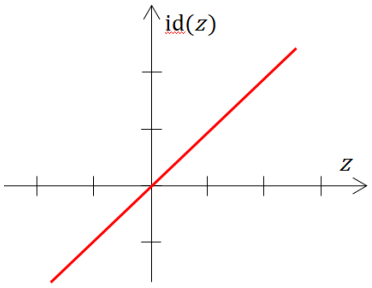
[97] The sigmoid activation function outputs values in the range (0, 1), so its outputs are not zero-centred. This results in the inputs to the downstream layer being strictly positive on average, which biases the signs of the gradients during backpropagation..

network's output into the desired format (e.g. probabilities for classification or real numbers for regression) and impose constraints on the output values to ensure they align with the problem domain. Unlike hidden layers, which introduce non-linearity for feature extraction, the activation functions of the output layer guarantee that the final values are relevant for making decisions.

Apart from *sigmoid* and *tanh*, the most common activation functions for output layers are:

- *Identity*

$$id(z) = z$$



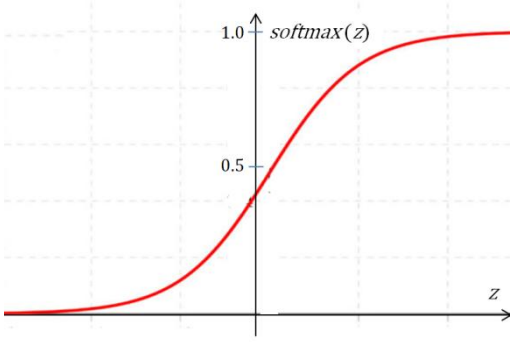
This function is ideal for regression problems. It produces unrestricted real-valued predictions. It is used when the output can be any numerical value, for example, in house price prediction or stock market forecasting. However, as the identity function introduces no non-linearity, it cannot capture complex feature transformations.

- *Softmax*

This is a common activation function used in neural networks for multi-classification problems, where the final layer produces a set of values, each of which corresponds to a different class. Before *softmax* is applied, these values can be any real numbers and may not provide any meaningful information directly. The *softmax* function converts these values into probabilities, indicating the likelihood of each class being correct. Specifically, the function for the ith output is defined as follows:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{m} e^{z_j}}$$

where z_i is the output of the previous layer of the network for the ith class, and m is the total number of classes.



While *sigmoid* and *softmax* have similar shapes, they serve different purposes and behave differently in terms of interpretability and use cases. The *sigmoid* function is used for binary classification and works independently for each neuron, meaning that it does not consider the outputs of other neurons. *Softmax*, on the other hand, transforms a vector of values into a probability distribution. It is used for multi-class classification, considering all the neurons in a layer and assigning relative probabilities across multiple classes.

- **Choosing the optimal activation function**. Choosing the optimal activation function is crucial for the performance of a neural network, as it directly affects learning dynamics, gradient flow, generalisation ability and computational efficiency. The following factors should be considered when selecting the most appropriate activation function:

- Task type (e.g. regression, classification, autoencoder ([98]), attention mechanisms ([99]))
- Gradient behaviour (avoiding vanishing or exploding gradients)
- Computational efficiency
- Interpretability of outputs
- Network depth (shallow vs. deep networks).

The table below summarises which activation function to use for which type of problem:

Problem Type	Recommended Activation Function	Where to Use
Regression	Identity	Output Layer
Binary Classification	Sigmoid	Output Layer
Multi-Class Classification	Softmax	Output Layer
Deep Networks	ReLU / Leaky ReLU	Hidden Layers
Recurrent Networks (RNNs, LSTMs)	Tanh / Sigmoid	Hidden Layers
Generative Models (GANs, VAEs)	Leaky ReLU / Tanh	Hidden Layers
Attention Mechanisms	Softmax	Specialized Layers

4.3.2.7. Choice of loss function

A loss function, also referred to as an *error* or *cost function* ([100]), plays a key role in machine learning. It measures the difference between the outputs predicted by an algorithm and the actual target values. As we have seen above, during training, a learning algorithm such as backpropagation uses the gradient of the loss function to adjust the model's parameters and minimise the loss. This effectively improves the model's performance on the dataset.

- **Common loss functions include**

- *Mean Square Error* (MSE)

We have already encountered this function in previous sections (see e.g. Section 3.2.4):

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2.$$

The MSE (also referred to as *L2 Loss*) penalises large errors more than small ones. It is most effective for predicting continuous values.

- *Mean Absolute Error* (MAE)*/L1 Loss*

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y}_i|.$$

---

[98] An autoencoder is a type of artificial neural network that learns to represent data by compressing and then reconstructing it. It is commonly used for dimensionality reduction, anomaly detection and the creation of generative models ([B6]).

[99] Attention mechanisms are fundamental concepts in deep learning, particularly in Natural Language Processing (NLP), computer vision, and sequence modelling. They enable neural networks to concentrate on the most relevant aspects of the input data when making predictions, thereby mimicking the way humans focus on specific details (see [B7] and Section 5.2.2).

[100] Although the terms 'loss function' and 'cost function' are often used interchangeably, they have distinct meanings. The loss function measures the accuracy of a single prediction made by a machine learning. The cost function, also known as the *objective function*, represents the average loss across a complete training set containing multiple training examples. It quantifies the model's performance on the entire dataset (for more information on loss functions, please refer to [A5]).

This function is less sensitive to outliers than the MSE. It is useful for problems where all deviations are equally important.

- *Huber Loss/ Smooth Mean Absolute Error* $(SMAE)$

$$SMAE = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & |y - \hat{y}| \leq \delta \\ \delta(|y - \hat{y}| - \frac{1}{2}\delta), & otherwise \end{cases}$$

where $\delta > 0$ is a parameter that determines the threshold for switching between the quadratic and linear components of the loss function. Huber Loss combines the advantageous characteristics of MSE and MAE into a single loss function. Like MAE, Huber Loss is less sensitive to outliers, but like MSE, it also penalises minor errors within the data sample. It is ideal for datasets containing noisy labels or outliers.

- *Binary Cross-Entropy* (BCE)

$$BCE = -[y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})]$$

where y is the true binary label (0 or 1) and $\hat{y}$ is the predicted probability of the positive class. BCE is used for predicting two categories (e.g., 0 or 1). It works well with *sigmoid* activation in the output layer.

- *Categorical Cross-Entropy* (CCE)

$$CCE = -\sum_i y_i \cdot log\,(\hat{y}_i).$$

This loss function is used for multi-class classification problems, such as classifying digits from 0 to 9.

- *Contrastive Loss* (CL)

Given two feature vectors X_1 and X_2, the contrastive loss function is:

$$CL = (1 - y) \cdot \frac{1}{2} D^2 + y \cdot \frac{1}{2} max(0, m - D)^2$$

where

- y is a binary label (0 for similar pairs, 1 for dissimilar pairs)
- D is the distance between embeddings X_1 and X_2 (e.g., Euclidean distance)
- m is a margin that defines the separation threshold between dissimilar samples.

Contrastive Loss is a loss function that is commonly used in metric learning for tasks such as face recognition, image similarity and text matching. It helps neural networks to learn meaningful embeddings by ensuring that similar items are close together in the learned feature space and dissimilar ones are far apart.

In this context, embeddings refer to vector representations of data – usually words, images, or other structured data – mapped into a continuous, low-dimensional vector space, where geometric proximity reflects semantic similarity (see Section 5.1.3.2 for more details). Unlike traditional classification losses (e.g. cross-entropy), in contrastive loss we do not explicitly compare predictions to truth labels. Instead, contrastive loss operates in the embedding space, using distance metrics between feature representations. In other words, contrastive loss does not measure classification errors directly, but rather evaluates the relationship between pairs of data points. In this case, the 'prediction' is the computed distance between two embeddings: $= \| X_1 - X_2 \|$. The model 'predicts' whether two samples are similar or different based on their embedding distance. In this sense, the prediction is implicit, embedded in the model's learned distance function rather than in a traditional classification score.

- **Choosing the optimal loss function**. Selecting the optimal loss function is essential for training neural networks because it directly affects the dynamics of learning, convergence, and generalisation. The appropriate loss function varies depending on the following factors:

- Task type (e.g. regression, classification or generative models).
- Robustness to outliers (some loss functions handle extreme values better).
- Computational efficiency (some loss functions are more expensive to compute).
- Interpretability, i.e. how meaningful are the error values?
- Gradient behaviour (avoiding vanishing/exploding gradients).

The table below provides a summary of the most suitable activation function for each problem type:

Loss Function	Best For	Advantages	Disadvantages
Regression			
Mean Squared Error (MSE)	Standard regression	Penalizes large errors more	Sensitive to outliers
Mean Absolute Error (MAE)	Robust regression	Less sensitive to outliers	May lead to slow convergence
Huber Loss	Data with outliers	Balances MSE & MAE	Needs tuning of delta parameter
Classification			
Binary Cross-Entropy	Binary classification	Works with Sigmoid activation	Can cause unstable training
Categorical Cross-Entropy	Multi-class classification	Works with Softmax activation	Requires one-hot encoding (see Section 5.1.3.1)
Specialised taks			
Contrastive Loss	Similarity learning	Learns meaningful embeddings	Requires careful pair selection

The key point to remember is that the nature of the output nodes, the activation function and the loss function all depend on the application in question.

4.3.2.8. Choice of optimisation algorithm

Selecting an optimisation algorithm is a critical design decision when training artificial neural networks. The optimiser governs how weights are updated during the learning process, thereby influencing the speed of convergence, stability and generalisation. This section outlines the current state of the art, practical selection criteria and the conceptual relationship between optimisation and machine learning.

- **Common optimisation algorithms**. There are several hundred popular optimisation algorithms, and several dozen algorithms to choose from in popular scientific code libraries. This can make it difficult to know which ones to consider for a given optimisation problem ([B20]).

Modern deep learning relies heavily on first-order, gradient-based methods ([101]) due to their ability to handle high-dimensional parameter spaces and large datasets. In particular, stochastic gradient descent (SGD) and its adaptive variants remain the workhorses of deep learning.

- *SGD and SGD with momentum*. SGD is probably the most popular optimisation algorithm. Unlike regular batch gradient descent, SGD follows the estimated gradient. It has proven to be much faster than batch gradient descent and almost as effective. Momentum addresses

[101] First-order, gradient-based methods are a family of optimisation algorithms that update model's parameters using only first-order derivative information – that is, the gradient of the loss function with respect to each parameter.

some of the problems associated with SGD. It was designed to accelerate SGD and make it more robust, even when the gradient is noisy and the parameter space is curvy.

- *Adam and AdamW*. These algorithms compute per-parameter adaptive learning rates using first- and second-moment estimates. AdamW decouples weight decay from the gradient update to improve generalisation.
- *AdaGrad and RMSProp*. *Adaptive Gradient* (AdaGrad) scales each parameter's learning rate inversely proportional to the square root of the sum of its past squared gradients. *Root Mean Square Propagation* (RMSProp) is an improvement on AdaGrad that uses an exponentially decaying average of past squared gradients instead of a cumulative sum. This prevents the learning rate from shrinking too much.
- *LAMB and LARS*. *Layer-wise Adaptive Moments optimizer for Batch training* (LAMB) and *Layer-wise Adaptive Rate Scaling* (LARS) work in a similar way to SGD or Adam. What sets them apart is their ability to adapt step sizes per layer, ensuring stable training when using very large batch sizes – sometimes involving tens of thousands of examples in parallel. LAMB combines Adam's adaptive moment estimates with layer-wise scaling, making it particularly effective for large models such as large language models. LARS rescales the learning rate for each layer proportionally to the ratio of the weight norm to the gradient norm.

Most second order methods are based on Newton's Method and due to expensive computations and saddle points in high dimensional space did not become popular in ML ([102]). However there are promising developments like *Saddle Free Newton* (SFN) method that might bring a new era of optimization. SFN is a second-order optimisation algorithm that addresses the issue of saddle points in high-dimensional machine learning loss landscapes. Essentially, SFN is like giving Newton's method 'curvature-corrected glasses' – it retains the beneficial scaling from curvature, but reverses the deceptive signals from saddle-point directions, causing the algorithm to move away from traps rather than towards them.

- **Choosing an optimisation algorithm**. Selecting the right optimiser depends on your model, data, and hardware:

- *Model size and batch scale*. Small to medium models typically perform well with AdamW or SGD with momentum. Very large batches (common in language model pretraining) often require LAMB or LARS to maintain convergence.
- *Convergence speed versus generalisation*. Adaptive methods converge rapidly, but can sometimes overfit. A hybrid strategy can perform better. One approach is to start training with AdamW and then switch to SGD with momentum for the final fine-tuning stage.
- *Learning-rate sensitivity*. If you require robust defaults, Adam and RMSProp tolerate a wider range of initial rates. For precise control, SGD requires schedules that are designed with great care (e.g. cosine annealing or warm restarts ([103])).
- *Computational and memory budget*. First-order methods (SGD and Adam) use little memory and are highly parallelisable. Second-order methods (L-BFGS and K-FAC ([104])) reduce the number of iterations, but require extra storage for curvature estimates.

---

[102] In deep learning, the loss surface is highly non-convex and filled with saddle points. The issue with classical Newton's method is that it uses the Hessian matrix (a matrix of second derivatives of the loss function) to adjust parameter updates. However, near saddle points, the Hessian matrix has both positive and negative eigenvalues. This means that standard Newton steps can move towards the saddle point instead of away from it. Furthermore, computing and inverting the Hessian exactly is computationally expensive in large models.

[103] Cosine annealing and warm restarts are learning-rate scheduling techniques designed to improve training efficiency and prevent models from becoming trapped in local minima. Rather than reducing the learning rate in fixed steps, cosine annealing allows it to follow a smooth cosine curve from a maximum value to a minimum value over a set number of iterations or epochs. Warm restarts reset the learning rate to a high value, initiating a new cosine decay cycle. This gives the optimiser a "fresh start" to explore new regions of the loss landscape.

In summary, the choice of optimisation method depends on the specific problem, the architecture of the machine learning model, the training objectives and the available computational resources. Research in this area is ongoing, with the aim of improving convergence rates, stability, and the ability to handle large datasets and complex models.

- **Machine learning versus optimisation**. Although closely related, machine learning and optimisation serve different purposes. Optimisation falls within the domain of mathematics. Mathematical problems are well-defined. Optimisation is the mathematical toolkit used to adjust model parameters in order to minimise (or maximise) the chosen loss or objective.

Machine learning, on the other hand, involves designing models and loss functions that can identify patterns in data, as well as evaluating their ability to generalise to new examples. Machine learning falls within the domain of engineering. Engineering problems are often not mathematically well-defined. ML involves creating a mathematical model that roughly describes the behaviour of the system under study. This model can then be subjected to mathematical analysis using standard techniques such as optimisation theory.

Although optimisation provides a way to minimise the loss function for machine learning, the goals of optimisation and ML are fundamentally different. The former is primarily concerned with minimising an objective, while the latter is concerned with finding a suitable model given a finite amount of data. For example, training error and generalisation error typically differ. Since the objective function of the optimisation algorithm is usually a loss function based on the training dataset, the aim of optimisation is to minimise training error. However, the objective of machine learning (or, more broadly, statistical inference) is to minimise the generalisation error ([Z3]).

In conclusion, the ML problem is not just about optimising a known function; it is about finding the parameters of a model that minimise an unknown expected risk, guided by finite data and conscious bias. By evaluating training and validation loss, we can estimate how well our model will perform on unseen data. Training loss indicates how well the model fits the data it was trained on, whereas validation loss provides an indication of how it will perform on new examples.

4.3.2.9. Limitations and challenges of ANNs

Artificial neural networks are powerful machine learning models, but they have several limitations and raise several challenges that can affect their effectiveness, efficiency and interpretability. While many of these issues affect machine learning models in general (see Section 4.5), some are particularly evident in ANNs because of their structure and learning mechanisms.

Let us take a closer look at these limitations and challenges:

× **Vanishing and exploding gradients**. Vanishing and exploding gradients are major issues in deep neural networks, particularly during backpropagation. These problems can severely impact a model's ability to learn effectively, resulting in slow or unstable training. Ideally, gradients should guide weight updates towards minimising the loss function. However, when networks become very deep, the behaviour of the gradients can become erratic, leading to vanishing or exploding issues.

A *vanishing gradient* problem occurs when the gradients become extremely small as they propagate backwards through the layers. This means that the early layers (those closer to the input) receive almost no gradient updates, which prevents effective learning. This slows

---

[104] Both L-BFGS and K-FAC are second-order optimisation methods, meaning they exploit curvature information from the Hessian to perform more accurate parameter updates than first-order methods such as SGD or Adam. The trick is that they do this approximately because computing and inverting the full Hessian for modern neural networks is intractable.

convergence or can lead to complete failure to learn. Deep layers become redundant because they do not receive meaningful updates.

We encounter an *exploding gradient* problem when the gradients become excessively large, resulting in unstable weight updates. In effect, the network's weights become too large, causing numerical instability and divergence. Consequently, loss functions fluctuate wildly, preventing effective convergence.

Mitigating these issues requires the proper selection of activation functions, weight initialisation and gradient control techniques.

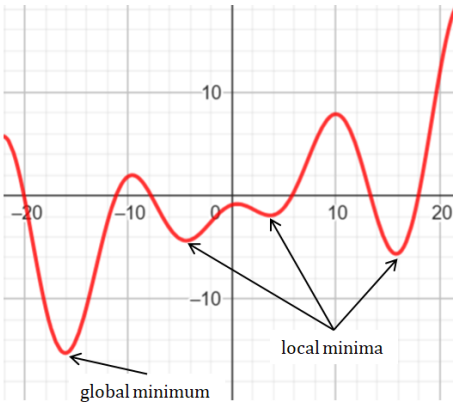
× **Problem of local minima**. This is one of the fundamental challenges when training artificial neural networks. It affects the way in which optimisation algorithms, such as gradient descent, navigate the complex loss landscape in order to find the best possible solution.

The training error (i.e. the loss L) of a neural network depends on its weights W_i and biases B_i. From this viewpoint, the training of neural networks can be seen as a process of parameter optimisation. This means that the aim is to find the set of parameters in the parameter space that minimises L. There are two types of optimality: the local minimum and the global minimum. A loss function L has a *local minimum* at (W_0, B_0) if there exists a constant $\varepsilon > 0$ such that

$$L(W_0, B_0) \leq L(W, B) \quad \text{for every} \quad (W, B) \in \{(W, B) : \| (W_0, B_0) - (W, B) \| \leq \varepsilon\}.$$

We say that L has the *global minimum* at (W_0, B_0) if $L(W_0, B_0) \leq L(W, B)$ for all (W, B) in the parameter space. Intuitively, a local minimum solution refers to a point in the parameter space with an error smaller than those of the surrounding points. On the other hand, if L possesses the global minimum solution at (W_0, B_0), then L has no smaller value at any other point in the parameter space.

It should be noted that there may be multiple local minima, but nevertheless the global minimum is unique, although it can be attained in a multitude of points. It is clear that the global minimum must be a local minimum, but not vice versa. The figure below shows four local minima, but only one of them is the global minimum.



The objective of the optimisation algorithm is to identify the global minimum of the loss function. However, the algorithm may occasionally become trapped in a local minimum. This, in turn, may result in the model falling short of optimal accuracy. The underlying cause of the local minima problem in ANNs is that deep networks possess millions of parameters, thus giving rise to complex loss surfaces with multiple minima. Furthermore, in contrast to simple linear models, ANNs possess highly non-convex loss functions, resulting in multiple valleys and peaks. We therefore usually settle for finding a value of the loss that is very low, but not necessarily minimal in any formal sense.

In real-world applications, the following strategies are frequently employed to move from a local minimum to the global one:

- The use of adaptive optimisation algorithms, such as Adam, helps to escape from local minima by means of the dynamic adjustment of learning rates.

- Gradually reducing the learning rate assists the model in converging towards the global minimum, thus avoiding the premature trapping of the model in a local minimum during the learning process.

- The utilisation of different sets of parameters to initiate multiple neural networks, with the subsequent selection of the network exhibiting the smallest error. Since the search begins from different initial points, multiple local minima can be obtained. The smallest of these minima is a more accurate estimation of the global one.

× **Overfitting**. ANNs, in particular deep networks, possess millions of parameters, resulting in a considerable overfitting risk when compared to traditional machine learning models, such as linear regression and K-nearest neighbours. Overfitting is one of the biggest challenges in ANNs.

Let us recall the definition of overfitting. This occurs when a neural network provides excellent prediction performance on the training data on which it is built, but performs poorly on unseen test instances. This phenomenon can be attributed to the tendency of the learning process to memorise arbitrary artefacts from the training data, which often prove to be ineffective when generalising to the test data. A potential indication of overfitting can be seen when a model is trained on different data sets, yet the same test instance yields very different results. This might be a sign that the training process is focusing on the nuances of the specific training data set, rather than learning patterns that generalise to unseen test instances.

The capacity of a model to produce useful predictions for instances with which it has no prior experience is termed *generalisation*. Generalisation is a useful practical property, and is therefore the most important aim in all machine learning applications. It is evident that in instances where training examples have already been assigned labels, there is no pragmatic value in predicting such examples once more.

The following methods are considered to be the key approaches for avoiding overfitting in a neural network:

- *Penalty-based regularisation*. Penalty-based regularisation represents the most prevalent technique employed by neural networks to mitigate the occurrence of overfitting. The concept of regularisation is to impose penalties (e.g. on large weight values) or other constraints on the parameters of a model, thereby preferring simpler models over more complex ones.
- *Early stopping*. In the early stopping of the iterative optimisation method, the process is terminated prematurely without reaching a convergent solution on the training data. The determination of the stopping point is made using a segment of the training data that has not been utilised in the construction of the model. The process is terminated when the error on the held-out data begins to rise.
- *Continuation methods*. These methods employ simple training models as a preliminary step, subsequently introducing more complex models. Starting with the optimum point of the simpler model provides a good initialization for a more intricate model. Deep learning models sometimes struggle with highly complex tasks at the start. Continuation methods improve stability by letting networks learn simple tasks before harder ones.
- *Improving training data*. The quality and variety of the training have a significant impact on the generalisation capabilities of an ANN. One method of achieving quality enhancement is to create new training samples through transformations (in computer vision, for instance, it can be done via image rotation or flipping and scaling). Another approach is the expansion of the dataset through either data collection or synthetic data generation.
- *Optimising hyperparameters*. The selection of appropriate model settings is essential for ensuring the efficiency of training. This process involves, for instance, a reduction in the learning rate and an adjustment to the batch size. It is apparent that the model's high

learning rates result in its rapid adaptation, which consequently leads to the overfitting of the training data. The use of smaller batches has been demonstrated to enhance randomness and mitigate the risk of overfitting.

× **Underfitting**. Underfitting constitutes a significant challenge in machine learning, including ANNs. This phenomenon occurs when a model is unable to identify the underlying patterns in the data, resulting in suboptimal performance on both the training and test datasets.

One potential cause of underfitting is the simplicity of the model. This can arise if the neural network is too shallow, i.e. if it has a small number of layers and neurons, resulting in an inadequate capacity to learn complex features. This phenomenon is a common occurrence when linear models are employed to address nonlinear problems. A further possible explanation for this phenomenon may be found in the inadequate hyperparameter settings. To illustrate this point, consider the scenario in which the learning rate is excessively high or the training is terminated prematurely. In such cases, the model is unable to acquire deep feature representations, resulting in suboptimal performance.

The solution to the problem of underfitting is to increase the complexity of the model, improve the duration of the training process, refine the hyperparameters, and ensure the quality of the dataset is adequate.

× **Adversarial vulnerability**. One of the most significant threats currently facing machine learning systems is that of adversarial attacks. These attacks exploit the vulnerabilities and limitations inherent in machine learning models, particularly deep neural networks. Adversarial attacks are a form of cyber-attacks that involve the manipulation of input data or the model itself with the aim of causing the AI system to produce incorrect or undesired outcomes. This has the potential to result in erroneous classification in security-sensitive applications such as facial recognition or fraud detection.

The implementation of continuous monitoring of AI systems for unexpected behaviour or outputs has been proposed as a method to help in the early detection of adversarial attacks. Furthermore, the implementation of advanced AI security measures is recommended, with the goal of identifying and neutralising adversarial inputs before they can compromise the system. Notable techniques include resilient machine learning models, which exhibit reduced sensitivity to adversarial manipulation, and anomaly detection systems that facilitate the identification of atypical patterns or inputs.

× **Hyperparameter tuning complexity**. Hyperparameter tuning represents a crucial yet complex aspect of training ANNs. It affects model performance, generalisation, and computational efficiency. Let us recall that hyperparameters represent configuration settings that govern the learning behaviour of an ANN; however, these parameters are not learned during the training process. They have to be set before training begins, and subsequently refined to ensure optimal performance. Examples of hyperparameters include the learning rate, the number of layers and neurons, and the optimiser choice.

Although hyperparameter tuning greatly impacts the performance of an ANN, selecting the right values can be challenging due to the complexity of the search space, training instability and computational cost. For instance, using too many neurons or layers can lead to overfitting, whereas too few can result in underfitting. As is well known ([Z1]), a feedforward neural network consisting of a single hidden layer with a sufficient number of neurons can approximate continuous functions of any complexity with arbitrary accuracy. However, there is currently no systematic method for determining the optimal number of neurons in the hidden layer, and trial and error is typically employed in practice.

Remark. There are meta AI approaches designed to automate hyperparameter tuning, which is often referred to as *hyperparameter optimisation* (HPO) or AutoML. These methods treat the hyperparameter search process as a learning problem and some use meta-models (including large language models) to guide it ([G26], [T17]). From the conceptual view, hyperparameters are 'outer loop' variables. Model parameters are learned using gradient descent, whereas

hyperparameters are chosen via a meta-process. Meta AI models aim to learn the outer loop by building surrogate models (e.g. Bayesian optimisation), using evolutionary algorithms or pretrained LLMs to suggest good starting points. They create a hierarchy:

- Inner loop: the model learns the weights.
- Outer loop: the meta model learns the hyperparameters.
- Sometimes, even higher loops exist (meta-meta learning).

Meta AI models automate the hyperparameter tuning process, replacing manual trial and error with a data-driven search. There are many Meta AI tools available for hyperparameter optimisation that save time and improve performance by eliminating the need for manual process. The most notable ones are Optuna (developed by the Japanese AI company Preferred Networks), Hyperopt (open-source), and Microsoft's NNI (Neural Network Intelligence). In the enterprise sector, Google Vizier (Vertex AI Vizier) is a powerful, industrial-scale service for large-scale optimisation. Together, these tools form the backbone of modern AutoML workflows, enabling the systematic exploration of hyperparameter spaces that would be impractical to tune manually.

However, these meta models also have their own hyperparameters, which must be chosen or tuned. Often, these are fewer and less sensitive than those of the base model. This is sometimes called the 'hyper-hyperparameter' problem. In practice, researchers often set these meta hyperparameters to reasonable defaults based on prior studies or empirical heuristics rather than tuning them extensively because the goal is to reduce the overall search cost. At a theoretical level, though, the recursion never ends: every optimiser has knobs, and one can always imagine a higher-level optimiser to tune them.

× **Parameter problem**. In deep learning, the *parameter problem* refers to the challenges arising from the large number of trainable parameters in modern neural networks ([105]). These parameters, known as weights and biases, are what the model learns during training. However, as models become more complex, several issues arise.

Deep learning models can possess a vast number of parameters. For example, GPT-3 has 175,000,000,000 parameters, whereas PaLM 1 (Pathways Language Model), a large language model developed by Google AI, has 530,000,000,000. This can lead to overfitting and optimisation complexity, since more parameters mean a larger search space for gradient descent, making the loss function landscape harder to navigate due to the presence of many local minima and saddle points. Furthermore, a high number of parameters requires more memory, longer training times and more powerful hardware.

In summary, the parameter problem is central to the scalability of deep learning. As models such as GPT-4 grow in size, it becomes essential to understand and manage parameters to enable efficient training and reduce computational costs.

× **High data requirements**. ANNs require substantial amounts of labelled, high-quality data in order to learn effectively. This phenomenon is especially evident in deep neural networks, which possess numerous layers and a multitude of adjustable parameters. In the absence of sufficient data, there is a risk of overfitting.

Inadequately labelled or noisy data can compromise the performance of a model. Inconsistent data can lead to biased learning, which can in turn affect predictions. For instance, medical AI models frequently encounter challenges when confronted with inconsistent healthcare records. However, the process of collecting sufficient labelled data, the subsequent cleansing and preprocessing of it, requires significant effort prior to the start of training. But this is both time-consuming and expensive.

---

[105] The number of parameters in a neural network is a function of the hyperparameters that define the model's architecture (number of layers, number of units per layer, kernel/filter size in CNNs, etc.).

Furthermore, the collection of real-world data gives rise to concerns regarding privacy, particularly in sensitive domains such as healthcare and the storage of personal information. Regulations such as the *General Data Protection Regulation* (GDPR) and the *California Consumer Privacy Act* (CCPA) impose restrictions on the manner in which data can be collected and processed ([106]).

× **High computational cost**. The enormous flexibility that deep learning models offer comes with a substantial computational cost. Training of deep ANNs requires not only significant computing power, but also frequently demands graphics processing units (GPUs), tensor processing units (TPUs), or other specialised hardware.

The first reason for enormous computational cost is applicable universally to all statistical models. In order to achieve an enhancement in performance by a factor of K, it is necessary to utilise at least K^2 additional data points during the training of the model. The second cause of the computational cost is attributable to overparameterisation, which results in a total computational cost for improvement of at least K^4. For instance, a 10-fold improvement would require at least a 10,000-fold increase in computation. However, it should be noted that this is a theoretical estimate. In practice, the actual requirements scale with at least the ninth power. It is evident that this ninth power implies that in order to achieve a 50% reduction in error rate, it is necessary to increase computational resources by more than 500 times ([T6]).

For instance, in recent years, there has been a considerable increase in computational demands associated with reducing errors in image classification. The estimation of the computational cost-performance curve of this task allows for the projection of the necessary computational resources required to achieve future performance enhancement. To illustrate this point, consider the example of obtaining an error rate of 5 percent. This would require 10^{28} floating-point operations ([T6]).

Researchers at the University of Massachusetts Amherst estimated the economic cost and carbon emissions implied by this computational demand. The following is the response: the financial cost of training such a model has been estimated at US $100 billion ($= 100 \cdot 10^9$), with the resulting carbon emissions amounting to those produced by New York City in a single month ([107]). Furthermore, an estimation of the computational burden resulting from a 1 percent error rate reveals results that are considerably worse ([T6]).

Empirical scaling laws in AI suggest that, all other things being equal – specifically, the quality of training data – the allocation of computing power must be equally divided between the increase in model size and the increase in data quantity to achieve optimal training. Consequently, as budgets for training models continue to increase, the size of both datasets and models is increasing in proportion. On a global scale, the financial expense associated with training the most sophisticated models has increased two- or threefold over the past eight years ([F2]).

Researchers at OpenAI designed and trained a deep-learning language system called GPT-3, at a cost of over $4 million per full training run (this figure only covers computing costs;

---

[106] GDPR, implemented in the EU, applies to any organization processing the personal data of EU residents, regardless of its location. CCPA is a US state law that applies to businesses conducting commercial activities in California or those that collect the personal information of residents within the state.

[107] Model training costs are increasing by a factor of around 2.4 every year. At present, 70% of the world's computing power for AI is held by the United States, 80% of it by American hyperscalers. Europe accounts for just 4% of global capacity, and suffers from industrial energy costs 1.5 to 3 times higher than in the US.
The world's leading AI start-ups are mostly based in the US, with 61% of global funding going to US companies, 17% to Chinese companies, and only 6% to EU companies. The development of infrastructure in the EU is hindered by a multitude of administrative procedures and legal appeals, as well as delays in connecting data centres to the electrical grid. For example, it takes at least five years to set up a data centre in France. ([F2])

engineering salaries, data preparation and infrastructure overheads are not included) ([108]). They made a mistake when implementing the system but did not fix it, explaining that, due to the cost of training, it was not feasible to retrain the model. Even businesses outside the tech industry are now beginning to avoid the high computational cost of deep learning ([109]). For example, a major European supermarket chain recently abandoned a deep-learning-based system that had significantly improved its ability to predict which products would be purchased. Company executives ended the project because they judged that the cost of training and running the system would be too high ([F2]).

4.3.3. Deep Learning architectures and models

A deep learning architecture serves as the blueprint for a neural network. It specifies the network's structure, defining the types of layers (e.g. convolutional, recurrent, or attention mechanisms), their arrangement, and how they connect with each other. It also specifies the overall flow of data from input to output. The diverse structural designs of neural networks enable models to learn complex patterns and representations. These architectures are specialised for different tasks, such as image recognition, natural language processing (NLP), reinforcement learning, and generative modelling.

On the other hand, a deep learning model is a specific implementation of an architecture that includes actual parameters, such as weights and biases. In practice, when people refer to a 'model', they are usually talking about the trained or deployable network that takes in data and produces predictions. Deep learning models are essentially multi-layered artificial neural networks that can automatically learn to recognise patterns in data.

In principle, a model with a greater number of parameters should possess a larger capacity and, consequently, be better able to handle complex learning tasks. However, in practical terms, complex models frequently require a significant amount of training time and are vulnerable to overfitting. The advent of cloud computing and big data has marked a paradigm shift in this regard. The enhanced computing capabilities have enabled more efficient training, while the big data has led to a reduction in the risk of overfitting. Recent trends in the field include the emergence of transformer within NLP systems and the exploration of hybrid architectures that integrate neural networks with symbolic reasoning.

In the following sections we will discuss the various deep learning architectures that have been utilised for AI applications. The initial focus will be on the simplest architectures, with subsequent progression to more complex ones. Understanding these architectures helps with selecting the right model for a given application and provides insights into how complex patterns and representations can be learned from data. While each architecture builds on the basic principles of neural networks – non-linearity, hierarchical abstraction and the use of large amounts of data – they differ in their approach to addressing specific challenges.

4.3.4. Feedforward Neural Networks (FNNs)

The concept of feedforward ANNs has been explored with great detail in Section 4.3.2. In this network, the information moves unidirectionally from the input nodes, through the hidden nodes (if any), to the output nodes. The network is characterised by the absence of cycles or loops. FNNs are particularly well suited to straightforward problems, such as digit recognition, fundamental classification, and function approximation.

In conclusion, FNNs can be regarded as a foundational concept in the field of neural networks and deep learning. The approach they have developed for modelling data and making

---

[108] OpenAI only carried out one full training run of the 175-billion-parameter GPT-3 model before its release. The overall cost of training GPT-3, including the many smaller pilot runs and the single full run, is generally estimated to have been in the low tens of millions of US dollars (with the most accurate estimates suggesting $10–20 million), rather than the $4–5 million often cited for one full run. By comparison, the cost of training GPT-4 increased to tens of millions, and the cost of training GPT-5 has been reported to be around half a billion dollars per full run ([G27], [O3]).

[109] For further information on this issue, readers are referred to Section 4.5.

predictions is both straightforward and accessible, and they have provided a foundation for more advanced neural network architectures that are now being used in a range of modern artificial intelligence applications.

Remark. People often use the terms 'multi-layer FNN' and ' multi-layer perceptron (MLP)' interchangeably because the canonical MLP is a fully connected, multi-layer FFN. However, MLPs are a special case of FNNs, which are a broader category. While every MLP is a multi-layer FFN, not every multi-layer FFN is an MLP. If it uses convolutions, attention mechanisms, sparse connections or other non-dense layers, it does not fall within the strict definition of an MLP.

In summary, an ANN is the most general category, covering many other architectures beyond simple feedforward stacks. An FNN is an ANN without cycles. A multi-layer FNN has multiple hidden layers and is the same as a classic multi-layer perceptron (MLP).

4.3.5. Convolutional Neural Networks (CNNs)

A CNN is a robust architecture of a specialized type of neural network designed to process and analyze grid-like matrix datasets like images, videos, or even sequences (e.g., audio). CNNs are particularly good at identifying patterns such as edges, shapes, or textures in images, making them extremely useful for computer vision tasks. They have revolutionized the way we handle visual data.

This section focuses on the use of CNNs for classification problems in image processing. However, these methods can also be applied to other tasks, including unsupervised feature learning, object detection and localisation. CNNs typically learn hierarchical features in different layers; earlier layers learn basic shapes, while later layers learn more complex ones.

CNNs modify multi-layer FNNs by introducing *convolutions*. First, let us look at what 'convolution' means. Mathematically, in one dimension, the *convolution* of two (real) functions, $x \to f(x)$ and $x \to h(x)$, is defined as follows

$$x \to (f * h)(x) = \int_{-\infty}^{\infty} f(s)h(-s + x)ds$$

where s is a dummy variable of integration. This operation may be considered as the area of overlap between the function f and the spatially reversed version of the function h ([110]). In other words, convolution blends two functions to produce a third function that expresses how the shape of one is modified by the other.

The function h represents the input, while f refers to a *filter*. The input h slides over the filter f, performing dot-product calculations. The variable x controls the movement of the input function relative to the filter function.

This naming convention originates from signal processing (where convolution was originally used to analyse waves) and deep learning. In these fields, h is known as the *input* because it represents the original data to be processed, such as image pixels, time-series values or word embeddings, while f is known as the *filter* because it extracts specific features from the input. The result is a feature map, highlighting essential details from the input. Different filters extract different features, leading to deeper learning insights.

Convolution is the fundamental process in CNNs, allowing them to automatically identify meaningful features in raw input data. However, CNNs use discrete convolution, which is a discrete approximation of continuous mathematical convolution.

The following types of convolution are used:

- **1D convolution**. This is used for text, speech and time-series data (e.g. sentiment analysis).

---

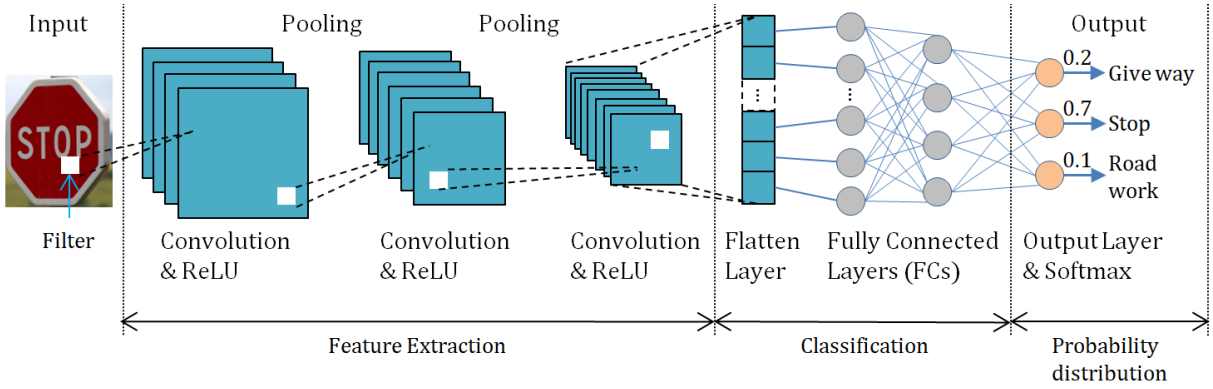
[110] The function h is reflected about the y-axis and shifted. Why is h reflected? The simplest explanation is that this is how convolution is defined. If we did not reflect h, the operation would be called *cross-correlation*.

- **2D convolution (standard CNN convolution)**. This is used for image processing and computer vision.

- **3D convolution**. This is used for videos and volumetric data where depth (the third dimension) is important. For example, MRI scans use 3D convolutions to analyse multiple slices of medical images.

4.3.5.1. The structure of a CNN

CNNs consist of multiple layers, including the input, convolutional, pooling, flatten, and fully connected layers. The following figure illustrates the architecture of CNNs.



Let us take a closer look at each of these layers.

- **Input layer.** The input layer of a CNN receives structured data in various formats, depending on the application, but most commonly images. The data formats used for CNN inputs are as follows:

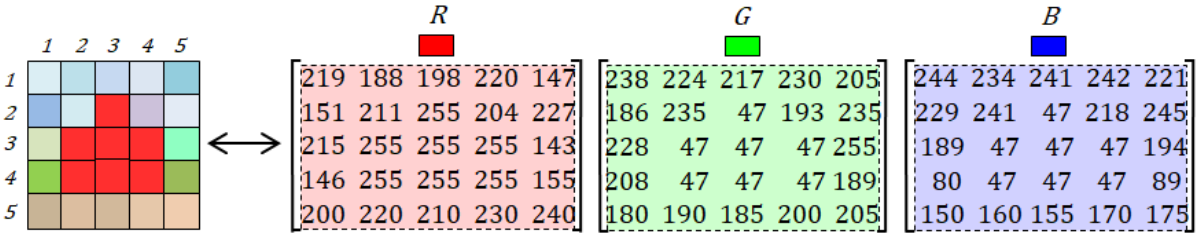
  - *Images*. This is the standard use case, in which CNNs extract spatial features from pixels.
  - *Videos*. A sequence of image frames that are processed either frame-by-frame or using 3D convolutions for temporal analysis.
  - *Text data*. CNNs can process text using word embeddings (e.g. character-level CNNs for natural language processing (NLP)).
  - *Time-series data*. Applied to sequential signals (e.g. financial market data).
  - *Graph data*. Used in *Graph Convolutional Networks (GCNs)* to process node-based relationships.
  - *Medical signals.* Used to detect patterns in ECG, MRI and EEG scans for diagnostic purposes.

We shall now discuss CNNs in the context of image processing. A two-dimensional image is an example of grid-structured data. This type of data exhibits spatial dependencies, as the colour values of adjacent pixels in an image are often similar. Adding a third dimension captures these different colours, creating a three-dimensional input volume. Therefore, the features in a CNN have dependencies based on spatial distances.

Digital images are essentially grids made up of tiny units called *pixels*. Each pixel represents the smallest part of an image and contains information about its colour and intensity. A pixel typically consists of three values that correspond to the red, green and blue (RGB) colour channels ([111]). These values range from 0–255 and determine the colour and intensity of the pixel. The primary colours are mixed in various proportions to generate a diverse array of colours. The specific levels of red, green and blue in a pixel determine its resulting colour.

---

[111] Other formats include e.g. HSL, HSV and CMYK, each of which stores and represents images differently ([J2]). In this article, we focus on RGB images.



As can be seen in the example above, the input image on the left has a size of just 5×5 pixels. Therefore, it consists of 25 pixels in total. In reality, such a small picture would be difficult to see, but it is a convenient size for our purposes here. The image has a grid that can be used to access each pixel. On the right, we can see that a CNN stores this particular image in three 5×5 matrices: $R = [r_{ij}]$, $G = [g_{ij}]$ and $B = [b_{ij}]$. If we access a particular pixel, such as one at location (3, 2), we receive the pixel values $[r_{32}, g_{32}, b_{32}] = [255, 47, 47]$.

In general, an RGB image input consists of three separate slices (one for each colour channel). Each slice is an $h \times w$ matrix, where h is the height of the image (i.e. the number of pixels vertically, or the number of rows of the matrix) and w is the width of the image (i.e. the number of pixels horizontally, or the number of columns of the matrix). When stacked together, the full input becomes an $h \times w \times 3$ structure. The representation of the input layer in this three-dimensional structure is natural because two dimensions are devoted to spatial relationships and a third dimension is devoted to the independent properties along channels.

The typical sizes used in CNNs are:

- *Small image datasets:* 32×32×3, e.g. CIFAR-10 and CIFAR-100 ([112]).
- *Standard image processing:* 224×224×3, as used in ImageNet-trained models such as ResNet.
- *High-resolution applications:* 512×512×3 or larger, used in medical and satellite images.
- *Custom sizes* that can be resized dynamically based on the model architecture.

Note that many CNN architectures resize high-resolution images (e.g. 4,000 × 3,000 pixels) to 224×224 pixels in order to match the dimensions required by the model. An alternative approach is to divide large images into smaller patches, or tiles, and process each one separately.

In summary, an RGB image input is structured as three stacked matrices (slices), one for each colour channel. CNNs process all three colour channels simultaneously to extract relevant patterns from the full colour spectrum.

- **Convolutional layers**. These layers are the heart of CNNs. Convolutional layers apply filters to input data in order to extract features. In the first layer, the input is an image and the output is a feature map. In mathematical convolution, it is the input function that shifts relative to the filter function via the variable x. However, in CNNs, the convolution operation is often described in reverse, i.e. where each filter slides over the image, capturing specific patterns like edges, corners, or textures.

---

[112] The CIFAR-10 dataset, created by the Canadian Institute for Advanced Research, is a labelled set of images commonly used to train machine learning and computer vision algorithms. One of the most widely used datasets for machine learning research, it contains 60,000 colour images, each of a 32x32 resolution, in 10 different classes. These classes represent aeroplanes, cars, birds, cats, deer, dogs, frogs, horses, ships and trucks. There are 6,000 images in each category. CIFAR-10 is a labelled subset of the 80 Million 'Tiny Images' dataset, which was published in 2009. When the dataset was created, students were paid to label all the images.
The CIFAR-100 dataset contains 100 classes, which are grouped into 20 superclasses. For example, CIFAR-10 has a single 'automobile' class, while CIFAR-100 differentiates between pickup trucks, buses, motorcycles and bicycles. ([W3])

Mathematically, the convolution operation is a scalar product between the filter and a small region of the image. In a digital image, pixels are arranged in a grid of rows and columns, so they are naturally indexed using integer coordinates. Given a filter matrix $[w_{ij}]$ of the size $m \times n$ and an image matrix $[x_{ij}]$ of the size $h \times b$ $(h \geq m,\ b \geq n)$, the convolution operation computes the matrix $[y_{ij}] = [w_{ij}] * [x_{ij}]$:

$$y_{ij} = \sum_{k=1}^{m} \sum_{l=1}^{n} w_{m-k+1,n-l+1} x_{i+k-1,j+l-1}.$$

The result y_{ij} is a scalar (hence why it is called the scalar product). This operation is carried out for every possible position in the input, sliding the filter across the image, to create the full feature map $[y_{ij}]$. Note that matrix $[w_{ij}]$ has been reflected twice: once over the x-axis and once over the y-axis.

Let X be an $h \times h$ matrix. If F is an $m \times m$ matrix $(h \geq m)$, then the size of the convolution matrix $F * X$ is $k = h - m + 1$.

Example. Let us compute the convolution $Y = F * X$ for

$$X = \begin{bmatrix} 10 & 20 & 30 \\ 15 & 25 & 35 \\ 20 & 30 & 40 \end{bmatrix},\ F = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}.$$

Since $n = m$, the matrix $F * X$ is 1-dimensional, i.e. it is a scalar. Notice that

$$F_{flipped} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}.$$

The convolution $Y = [y_{11}] = F * X$ is the element-wise multiplication of X and $F_{flipped}$, followed by the subsequent addition of the terms

$$\begin{aligned} y_{11} = {} & (-1) \cdot 10 + 0 \cdot 20 + 1 \cdot 30 + \\ & (-1) \cdot 15 + 0 \cdot 25 + 1 \cdot 35 + \\ & (-1) \cdot 20 + 0 \cdot 30 + 1 \cdot 40 \\ = {} & 60. \end{aligned}$$

Remark. In practical implementations of CNNs, the operation most commonly used is actually *cross-correlation*, even though the term *convolution* is widely used. It can be confusing, so you will have to be careful when reading the literature. The definition of cross-correlation is as follows:

$$y_{ij} = \sum_{k=1}^{m} \sum_{l=1}^{n} w_{kl} x_{i+k-1,j+l-1}.$$

Notice the difference: Convolution flips the kernel both around its horizontal and vertical axis, but cross-correlation does not. Why is the term 'Convolutional' Neural Networks' still used? Originally, convolution was more widely understood in signal processing, where true convolution (with kernel flipping) is essential, and this term was adopted in deep learning. However, since CNNs automatically learn optimal filter weights, the flipping step becomes unnecessary and cross-correlation works just as well for detecting patterns. This is why frameworks such as *TensorFlow* and *PyTorch* implement cross-correlation instead. Nevertheless, they still call it 'convolution' for consistency with historical terminology. We will adhere to this nomenclature and retain the term 'convolution', despite the use of cross-correlation.

Now, let us take a look at the following key terms of convolutional layers: channel, filter, stride padding, the convolution operation and activation.

- *Channel*. The term *channel* in a CNN has different meanings depending on which layer we are referring to. In the input layer, channels represent different types of raw data. For images, these channels correspond to colour components. Thus, an *RGB* image input is represented as an $h \times w \times 3$ structure (height, width and three colour channels).

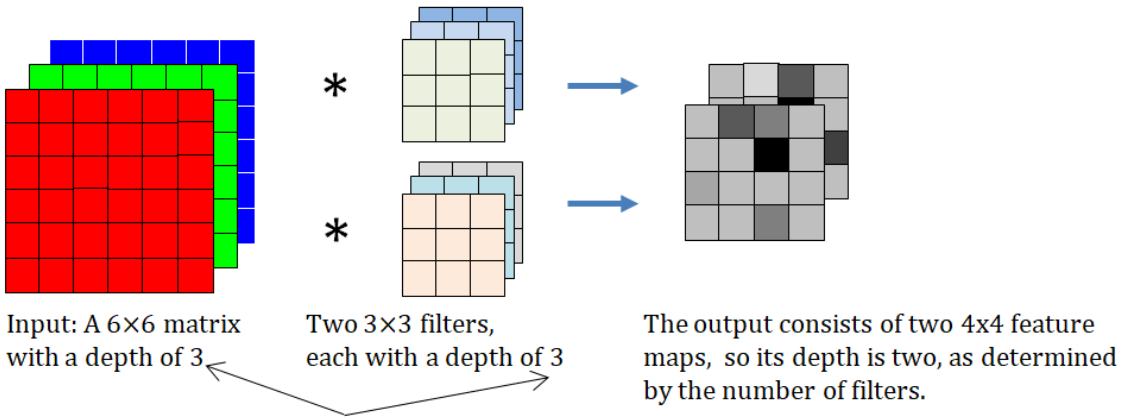
In hidden convolutional layers, channels correspond to feature maps rather than raw input data. Each filter in a convolutional layer produces a new feature map, meaning that the number of channels increases as the network becomes deeper. For instance, early layers detect edges, middle layers detect textures, and deep layers detect higher-level patterns extracted from local image regions.

Let us assume that the input to the s^{th} layer is $h_s \times b_s \times d_s$ in size. Here, h_s refers to height, b_s to width (or breadth) and d_s to depth. In almost all image-centric applications, the values of h_s and b_s are the same. In the context of a single layer, 'depth' refers to the number of channels in that layer. Therefore, $d_1 = 3$ refers to the number of primary colour channels, whereas for $s > 1$, d_s refers to the number of feature or activation maps.

- *Filter.* Filters are also organised into sets of three-dimensional structural units. A filter is usually composed of square matrices whose dimensions are typically much smaller than those of the layer to which the filter is applied. For consistency in feature extraction, all filters in a given CNN layer (say, layer s) have the same height and width f_s. Each filter consists of multiple square matrices (slices or kernels), one for each channel of the input ([113]). The size of these matrices is always uniform within that layer. Each filter detects a specific feature in every position of the input.

All feature maps from the previous layer must be processed by the filters of the subsequent layer. This means that the depth of each filter in layer s ($s > 1$) is always equal to the depth d_{s-1} of the previous layer. This ensures that the convolution operation spans the entire previous layer's feature space, allowing information to be fully processed. For instance, if the input has three channels (RGB), each filter must be three-dimensional in order to interact with all colour channels. Similarly, if a hidden layer has 64 feature maps, each filter applied to that layer must also have a depth of 64 to process all the extracted features.

The number of filters in layer s is arbitrary and determines the number of feature maps in the output, and thus the depth d_{s+1} of the next layer $s + 1$.



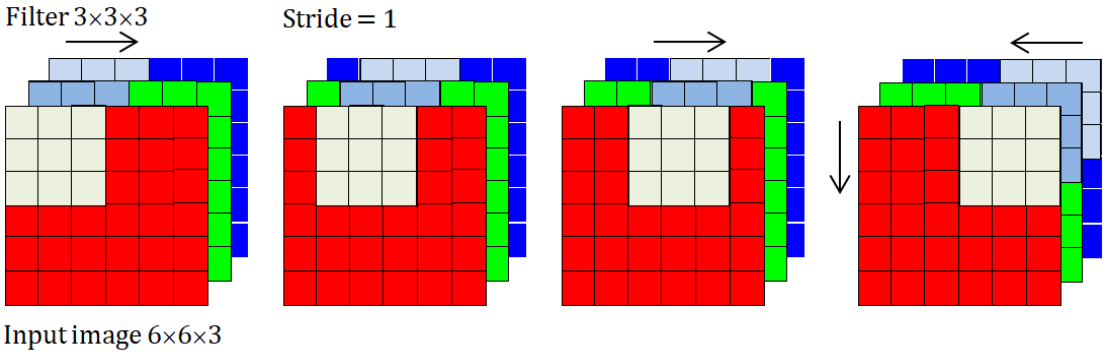
Common filter sizes include:

- 3 × 3: most frequently used in modern CNN architectures.
- 5 × 5: captures slightly larger patterns.

[113] In the context of CNNs, the terms 'filter' and 'kernel' are often used interchangeably, but they have slightly different meanings. A kernel (or 'slice') is a single 2D matrix of weights used in convolution operations. In contrast, a filter is a 3D structure formed by multiple kernels spanning all input channels. Therefore, kernels operate on individual channels in convolution, while filters are the complete sets of kernels that ensure CNNs process multi-channel inputs correctly.

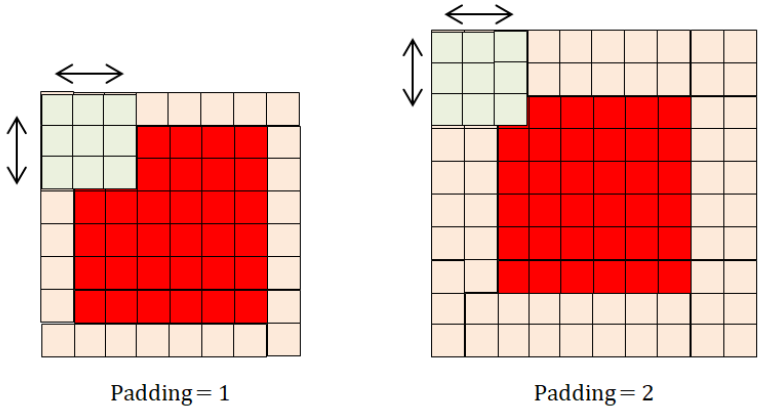
- 7×7: used in deeper models for broad feature detection.

- *Stride.* Stride is the step size by which a filter moves across the input image during convolution. This directly affects the size of the output feature map and controls how much the input shrinks at each layer. For example, a stride of 1 means the filter moves one pixel at a time, resulting in a high-resolution feature map. A stride of 2 means the filter moves two pixels at a time, reducing the size of the feature map faster.



Increasing the stride enables the filter to cover a larger area of the input image, making it useful for capturing global features. In contrast, reducing the stride enables finer, more localised details to be captured. Furthermore, increasing the stride helps to prevent overfitting, though it reduces computational efficiency by decreasing the spatial dimensions of the feature map.

- *Padding.* Padding refers to adding extra pixels around the edge of the input image before convolution is applied. This helps to control the size of the output feature maps and plays a key role in improving feature extraction.



Padding is mainly applied when the edges of an image contain useful information that you want to capture. The amount of padding can be increased up to the size of the kernel that is used. The most common type is zero padding, whereby the added pixels are set to zero.

- *Convolution.* Now, let us give a formal definition of the convolution operation from the s^{th} layer to the $(s+1)^{\text{th}}$, layer. The r^{th} filter $(r = 1,2, \dots, d_{s+1})$ in the s^{th} layer is given by the following matrix vector, i.e., a vector whose elements are matrices:

$$\mathrm{W}^{(r,s)} = [W_1^{(r,s)}, W_2^{(r,s)}, \dots, W_{d_s}^{(r,s)}],$$

where $W_c^{(r,s)} = \left[w_{abc}^{(r,s)}\right]$, $a, b = 1, 2, \dots, f_s$, is the c^{th} filter's slice (being an $f_s \times f_s$ matrix). The indices $a, b = 1, 2, \dots, f_s$ and $c = 1, 2, \dots, d_s$ determine the positions along the height, width, and depth of the filter. For example, $w_{214}^{(r,s)}$ is the element in the 4^{th} slice at location $(2, 1)$.

The feature maps in the s^{th} layer are represented by the following channel structure:

$$\boldsymbol{X}^s = [X_1^s,\ X_2^s, \dots, X_{d_s}^s],$$

where $X_c^s = \left[x_{ijc}^s\right]$, $i = 1, 2, \dots, h_s$, $j = 1, 2, \dots, b_s$, is the c^{th} channel (being an $h_s \times b_s$ matrix). Thus, $\boldsymbol{X}^s$ is a matrix vector whose elements are channel matrices. In particular, $\boldsymbol{X}^1$ is the input layer, consisting of the R, G, B channels, i.e. $\boldsymbol{X}^1 = [R, G, B]$.

The convolutional operations

$$X_r^{s+1} = \boldsymbol{W}^{(r,s)} * \boldsymbol{X}^s, r = 1,2, \dots, d_{s+1},$$

compute d_{s+1} channels of the entire feature vector $\boldsymbol{X}^{s+1}$ of the layer $s + 1$

$$\boldsymbol{X}^{s+1} = [X_1^{s+1},\ X_2^{s+1}, \dots, X_{d_{s+1}}^{s+1}].$$

The r^{th} matrix $X_r^{s+1} = \left[x_{ijr}^{s+1}\right]$ has the following elements ([114])

$$x_{ijr}^{s+1} = \sum_{c=1}^{d_s} \sum_{k=1}^{f_s} \sum_{l=1}^{f_s} w_{klc}^{(r,s)} x_{i+k-1,j+l-1,c}^s,$$

where $i = 1, 2, \dots, h_s - f_s + 1$ and $j = 1, 2, \dots, b_s - f_s + 1$. This means that the size of matrix X_r^{s+1} is equal to $(h_s - f_s + 1) \times (b_s - f_s + 1)$.

More generally, if X is an $h \times b$ matrix and W is an $f \times f$ filter, then the output matrix $W * X$ has the following dimensions

$$h_{out} = (h - f + 2p)/s + 1,\ b_{out} = (b - f + 2p)/s + 1,$$

where p is padding and s stride ([115]).

Note that the matrix X_r^{s+1} can be expressed as the element-wise sum of the convolutions $W_c^{(r,s)} * X_c^s$

$$X_r^{s+1} = \sum_{c=1}^{d_s} W_c^{(r,s)} * X_c^s,$$

resulting in one single feature map (channel) of the layer $s + 1$ for the r^{th} filter.

As with all neural networks, biases can also be added to forward operations. Each filter in a layer is associated with its own bias; for example, the r^{th} filter in the s^{th} layer has a bias of $b^{(r,s)}$. When a convolution is performed using the r^{th} filter in the s^{th} layer, the value of $b^{(r,s)}$ is added to the dot product.

$$x_{ijr}^{s+1} = \sum_{c=1}^{d_s} \sum_{k=1}^{f_s} \sum_{l=1}^{f_s} w_{klc}^{(r,s)} x_{i+k-1,j+l-1,c}^s + b^{(r,s)}.$$

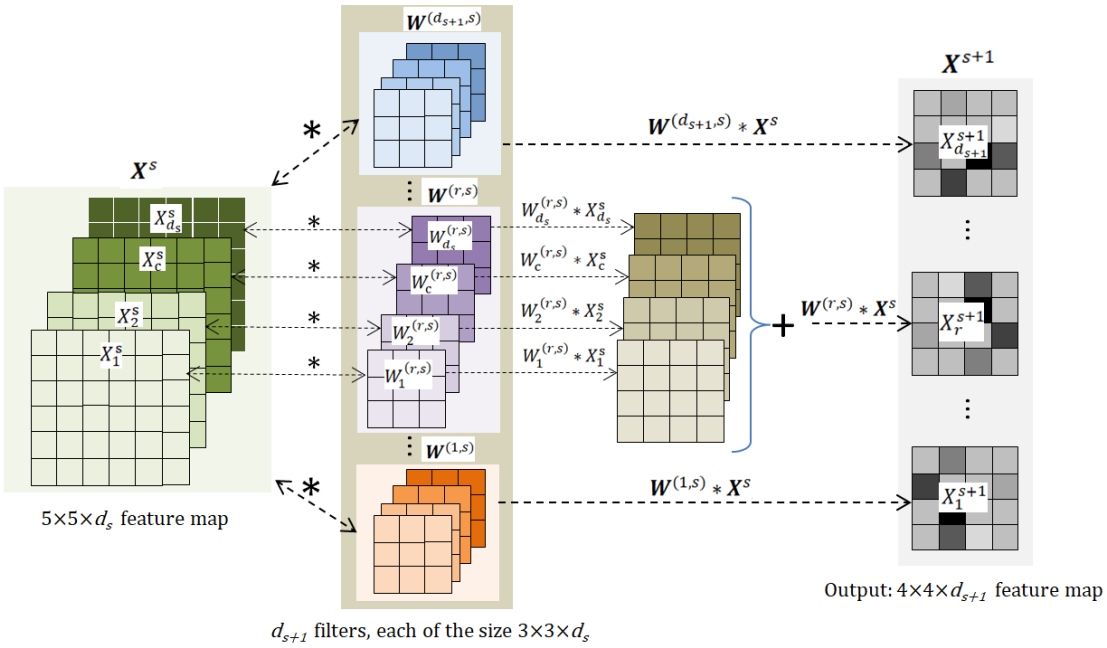
Bias plays an important role in convolutional layers by helping the model to adjust activation values. For example, if a convolutional filter detects edges, the bias ensures that subtle contrasts are not overlooked.

Although the above algorithm may appear complex in terms of notation, the underlying convolutional operation is essentially a straightforward dot product applied across the entire filter volume, repeated for all valid spatial positions (i, j) and filters (indexed by r). The following figure illustrates how the output is computed.

---

[114] Please note that this is actually a cross-correlation.

[115] Observe that neither h_{out} nor b_{out} is guaranteed to be a positive integer. In practice, however, deep learning frameworks either enforce the correct constraints or automatically adjust the padding/stride to ensure valid output sizes.



- *Activation*. In most standard CNN architectures, an activation function f is applied immediately after a convolution layer and before the pooling operation ([116]). This ensures that CNN learns non-linear features before reducing the dimensions. Thus, in a typical CNN, the input to the $(s+1)^{\text{th}}$ convolutional layer is not simply the vector of the raw feature maps $\boldsymbol{X}^{s+1} = [X_1^{s+1}, X_2^{s+1}, \dots, X_{d_{s+1}}^{s+1}]$, but rather the vector of the *activation maps* obtained by applying the activation function f to each feature map:

$$f(\boldsymbol{X}^{s+1}) = [f(X_1^{s+1}), f(X_2^{s+1}), \dots, f(X_{d_{s+1}}^{s+1})],$$

where $f(X_r^{s+1}) = f([x_{ijr}^{s+1}]) = [f(x_{ijr}^{s+1})], r = 1,2,\dots,d_{s+1}$. Therefore, applying an activation function does not change the dimensions, since it is simply a one-to-one mapping of features to activation values.

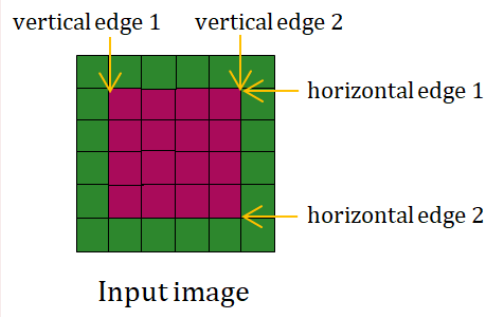
The most commonly used activation function in CNNs is *ReLU* (see Section 4.3.2.6). CNNs must capture complex patterns, and *ReLU* ensures that the outputs are not simply linear combinations of the inputs. Furthermore, unlike *sigmoid* or *tanh* functions, *ReLU* does not squash values into a small range, enabling strong gradients to be maintained during backpropagation. *ReLU* performs a simple element-wise maximum operation, making it much faster to compute than exponential functions such as *sigmoid* or *tanh*.

A *ReLU* typically follows a convolution operation, and the *ReLU* layer is often not explicitly shown in pictorial illustrations of most convolutional neural network architectures ([117]).

Example. Let us consider the real-world scenario of detecting edges in an RGB image. We will apply convolution using filters designed to highlight edges. The input image and its corresponding *RGB* structure $\boldsymbol{X} = [R, G, B]$ are as follows:

[116] Although some models experiment with reversing the order, the standard approach remains: Convolution → Activation → Pooling.

[117] Some frameworks treat activation functions as separate layers when defining a model. Examples of such frameworks include *TensorFlow* and *Keras*.



$$R=\begin{bmatrix}83&83&83&83&83&83\\83&169&169&169&169&83\\83&169&169&169&169&83\\83&169&169&169&169&83\\83&169&169&169&169&83\\83&83&83&83&83&83\end{bmatrix}\quad G=\begin{bmatrix}157&157&157&157&157&157\\157&11&11&11&11&157\\157&11&11&11&11&157\\157&11&11&11&11&157\\157&11&11&11&11&157\\157&157&157&157&157&157\end{bmatrix}\quad B=\begin{bmatrix}97&97&97&97&97&97\\97&90&90&90&90&97\\97&90&90&90&90&97\\97&90&90&90&90&97\\97&90&90&90&90&97\\97&97&97&97&97&97\end{bmatrix}.$$

To detect vertical edges 1 and 2, we use the filters [118] $\boldsymbol{W}^{v1}=[W_R^{v1},W_G^{v1},W_B^{v1}]$ and $\boldsymbol{W}^{v2}=[W_R^{v2},W_G^{v2},W_B^{v2}]$, where

$$W_R^{v1}=W_G^{v1}=W_B^{v1}=\begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix}$$

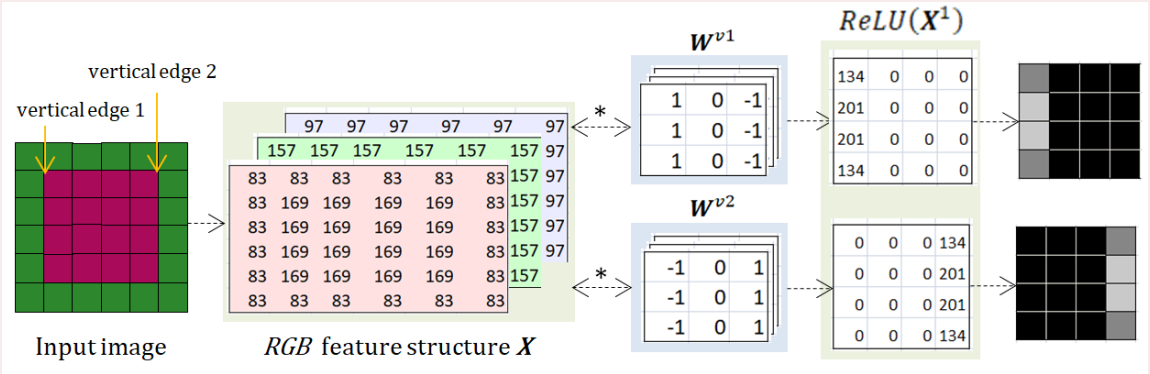
and

$$W_R^{v2}=W_G^{v2}=W_B^{v2}=\begin{bmatrix}-1&0&1\\-1&0&1\\-1&0&1\end{bmatrix}.$$

First, we compute the convolutions (we set the biases to 0) and then we apply the $ReLU$ function to them

$$ReLU(\boldsymbol{W}^{v1}*\boldsymbol{X})=\begin{bmatrix}134&0&0&0\\201&0&0&0\\201&0&0&0\\134&0&0&0\end{bmatrix},\quad ReLU(\boldsymbol{W}^{v2}*\boldsymbol{X})=\begin{bmatrix}0&0&0&134\\0&0&0&201\\0&0&0&201\\0&0&0&134\end{bmatrix}.$$

After using these edge detection filters, the resulting feature maps contain numerical values representing the locations and intensities of edges in the image. We can now utilise grayscale conversion to generate grey images from $ReLU(\boldsymbol{W}^{v1}*\boldsymbol{X})$ and $ReLU(\boldsymbol{W}^{v2}*\boldsymbol{X})$



[118] The question the reader may have is how to select the right filter for extracting a specific feature, such as an edge. In a CNN, the values of an edge detection filter (or any other filter) are not programmed in; they are learned automatically during training. At the beginning, CNN filters are initialised with random values. These random values do not detect edges initially, but as the model is trained, the filters evolve to recognise patterns such as edges, textures and objects. CNNs use many filters, each of which learns different types of features. Filter values are updated by backpropagation with gradient descent.

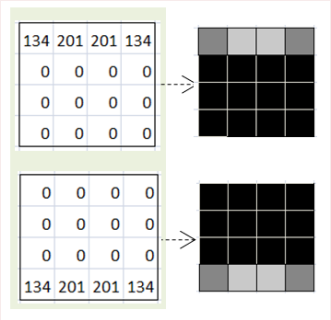
The lighter regions in the grey images correspond to the detected vertical edges 1 and 2 in the input image.

Similarly, we can use the filters $\boldsymbol{W}^{h1}$ and $\boldsymbol{W}^{h2}$ with slides

$$W_R^{h1} = W_G^{h1} = W_B^{h1} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix},$$

$$W_R^{h2} = W_G^{h2} = W_B^{h2} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}.$$

to detect horizontal edges 1 and 2:



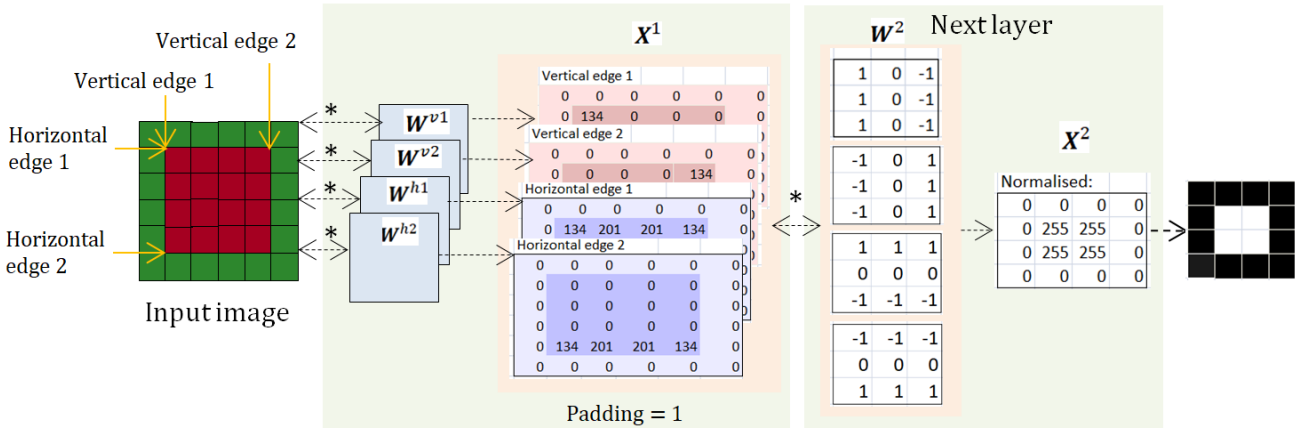
Remark. In a standard CNN, when a single filter is applied to an RGB image, the resulting feature map effectively loses direct colour information because the filter sums across the RGB channels. However, CNNs can still retain colour-related features through specific mechanisms. One method is to use multiple filters. CNNs typically apply multiple filters per layer, rather than just one. Some of these filters naturally evolve to focus on specific colour patterns, enabling the network to extract colour-based textures as well as grayscale edges. Deeper layers then combine information from these filters to reconstruct complex colour-based features. For instance, while a single filter may lose colour specificity, another filter in the same layer may learn a texture unique to red objects, thus preserving that distinction. If explicit colour recognition is required, alternative methods such as channel separation or colour space conversion can be employed.

- *Hierarchical feature engineering*. In CNNs, hierarchical feature engineering refers to the structured approach through which features are progressively extracted at different levels, ranging from low-level details to high-level abstract representations.

In other words, CNNs learn features in a hierarchical manner. This means that the initial layers identify basic features, such as edges, corners and simple textures. The middle layers then build complex shapes from these basic features. Finally, the deepest layers recognise entire objects by identifying high-level patterns. These patterns can be object parts, such as eyes, wheels and doors, or full object representations, such as cars, dogs and buildings. The number of filters increases with each layer, enabling the CNN to detect more detailed patterns. The more filters there are, the more aspects of an image the CNN can detect. As the layers are stacked, they build on previous features, making the classification process more robust.

This hierarchical process enables CNNs to automatically construct complex features from raw input when performing tasks such as image classification, object detection and facial recognition.

The figure below illustrates how filters identify edges and join them together to form a rectangle.



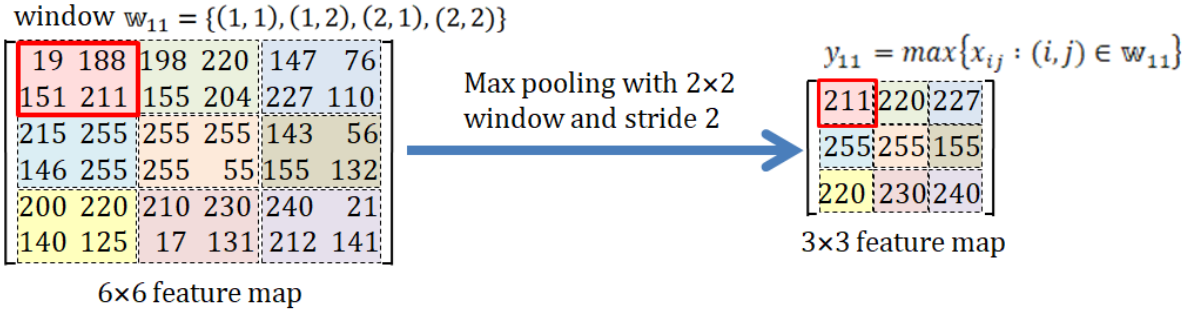
All values v greater than 255 in the $\boldsymbol{X}^2$ output have been normalised according to the following formula

$$v_{norm} = \frac{v - v_{min}}{v_{max} - v_{min}} \cdot 255.$$

- **Pooling layers**. In CNN architectures, it is common practice to insert a pooling layer periodically between successive convolutional layers. This is a type of down sampling operation used to reduce the spatial dimensions (height and width) of feature maps while preserving the most important information. Pooling makes the network more efficient by reducing computation and improving generalisation, as it makes the model more invariant to minor variations such as translations or distortions.

Unlike filters, the pooling operation works on small grid regions of size $p_s \times p_s$ in each layer and produces another layer with the same depth. For each square region of size $p_s \times p_s$ in each of the d_s activation maps, the maximum value is returned. This approach is referred to as *max pooling*. Another technique is called *average pooling*, involves returning the average value in a region.

Pooling typically uses a 2×2 or 3×3 window $\mathbb{w}$ that moves across the activation map with a stride of (usually) 2. This produces a downsampled version of the map that retains only the most important details. Using a stride $s_s = 2$ creates a new layer of size $[(h_s - p_s)/s_s + 1] \times [(b_s - p_s)/s_s + 1] \times d_s$, which significantly reduces the spatial dimensions of each activation map.



Modern architectures such as ResNet and DenseNet ([119]) typically omit traditional pooling after each convolutional layer. Instead, these models use strided convolutions (increasing the stride from 1 to 2) to achieve down sampling while retaining more information.

Another type of pooling is *Global Average Pooling* (GAP). GAP is used instead of multiple pooling layers at the end of a CNN, prior to the final classification layer. GAP reduces each

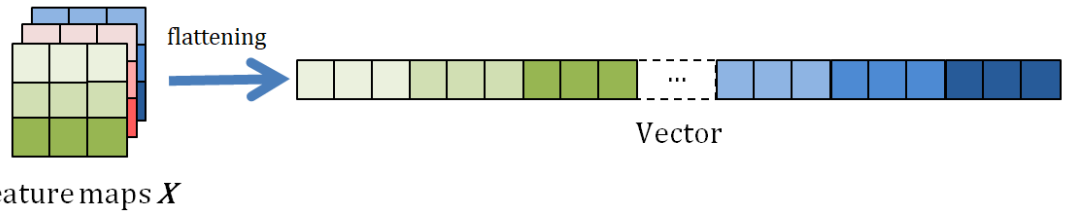
[119] *Densely Connected Convolutional Network* (DenseNet) is a type of CNN architecture that connects all layers to each other ([H6]). The design of DenseNet is based on the simple principle that by concatenating the feature maps of all previous layers, a dense block enables each layer to access the features of all preceding layers. In classic CNNs, each layer only has access to the features of the layer immediately preceding it.

feature map to a single value per channel, thereby improving generalisation and reducing the number of parameters.

- **Flatten layer**. The *Flatten Layer* converts multi-dimensional feature maps into a one-dimensional matrix, i.e. a vector. This transformation enables the output of convolutional layers, which are three-dimensional structures, to be used as input for fully connected layers, where each neuron receives a single input. Without flattening, the fully connected layer would not be able to process the feature map, since it requires a flat input vector rather than a multi-dimensional array.

Let us remark that the pooling and flattening layers do not have an activation function.

The mapping of a feature map structure to a vector in the flatten layer is done in a specific order. This ensures that the spatial relationships in the feature map structure are preserved. The standard order typically used is row-major order (also known as C-style ordering), meaning that pixels are extracted from left to right across rows. Once a row has been fully added, the next row is processed. This process is repeated for each feature map before moving on to the next one in depth.



Although flattening alters the shape of the data, it does not affect the information itself.

Note that the order of the channels in the feature matrix vector $\boldsymbol{X}$ is not arbitrary. Rather, it is based on the order in which the filter's slices are applied during the convolution process. This order is maintained across layers to ensure consistent backpropagation and feature extraction.

- **Fully connected layers (FCs)**. After the convolutional and pooling layers, the extracted features are flattened and passed through fully connected layers. These layers serve as the final stage for performing classification or regression tasks upon the processed features.

Fully connected layers, also known as *dense layers*, consist of neurons that are all connected to each other. Unlike convolutional layers, which preserve spatial relationships, FCs treat input features as a flat vector. One can think of FCs as feedforward neural networks (FNNs) that are attached to the end of a CNN pipeline for the purpose of making the final decision. The difference between standalone FNNs and FC layers in CNNs is that FNNs process raw input directly, whereas FC layers in CNNs process pre-learned feature representations, making classification more effective.

- **Output layer**. The output layer is the final layer of a CNN and is responsible for generating the final predictions. Depending on the type of task, it typically consists of one or more neurons. For classification tasks, there is one neuron per possible class (e.g. 10 neurons for recognising digits 0–9). For regression, there is either a single neuron or multiple neurons representing continuous values (e.g. object coordinates). The output layer applies a final activation function to ensure that the predictions are formatted correctly.

At first glance, the output layer in a CNN appears to work similarly to the output layer in an ANN, since both are responsible for producing final predictions. However, while ANNs process raw feature vectors, CNNs process structured feature maps from convolutional layers. CNNs use spatial feature extraction (convolutions and pooling) before reaching the output layer. Furthermore, CNNs typically use the *softmax* function for multi-class image classification, whereas ANNs may use different output functions depending on the application. In essence, CNNs and ANNs share the same output layer logic, but CNNs have specialised processing layers before reaching it.

- **Parameters.** In CNNs, parameters refer to the learnable values that the network adjusts during training to optimise feature detection and classification. In addition to the parameters of fully connected layers, which operate in a similar manner to those in a standard FNN (see Section 4.3.2), CNNs possess further learnable parameters within their convolution layers. These are of two primary types:
  - *Weights.* These are the values within the filters (kernels) that determine how features are extracted. Each filter has a set of weights that are applied to the input image during convolution. The network learns these weights through backpropagation, adjusting them to detect patterns.
  - *Biases.* As we know, a bias is a constant value added to the output of each neuron before an activation function is applied. This helps to shift feature activations, ensuring that CNNs can learn better representations. Each feature map has a corresponding bias parameter that is adjusted during training.
- **Hyperparameters.** When you consider the hierarchical process of recognising images, you may wonder how this is possible given that the filters are initialised with random numbers and then trained automatically. How can the network achieve this particular order of increased feature complexity when the filters are generated autonomously? This is precisely where the hyperparameters come into play.

Although CNN filters are learned automatically, the network's structure is intentionally designed by engineers prior to training. This is achieved by setting hyperparameters that define the architecture of a CNN. Hyperparameters are manually set before the network's learning process starts and are not learned during training. These settings play a crucial role in controlling how the CNN extracts features, learns patterns and optimises performance.

The hyperparameters that control the entire CNN framework are:

- *Model depth:* number of convolutional and fully connected layers.
- *Filter settings:* size, stride and number of filters per layer.
- *Pooling strategy:* type (max/average), size and frequency of pooling layers.
- *Fully connected layers:* number of neurons and layers before the output.
- *Activation functions:* choice of *ReLU*, *sigmoid*, *softmax*, etc.
- *Regularisation:* dropout rate ([120]), batch normalisation and weight decay.
- *Training strategy:* learning rate, optimiser selection, batch size and epochs.

Hyperparameters impact CNN performance (poorly chosen hyperparameters can lead to overfitting or underfitting), optimise training speed, and improve feature extraction. Filter size, stride and padding affect the ability of a CNN to recognise patterns. Hyperparameter tuning is done manually or using techniques such as grid search or Bayesian optimisation.

4.3.5.2. Training process of a CNN

CNNs are trained using supervised learning, in which labelled data is used to adjust the model and enable it to make accurate predictions. At first glance, FNNs and CNNs appear to be trained using the same standard procedures involving forward propagation, loss computation, backpropagation and weight updates, usually via gradient descent or its variants. However, there are several key differences in the details of the process due to their underlying architectures. While the backpropagation process in the fully connected layers of a CNN is fundamentally the same as in an FNN (see Section 4.3.2 for details), the backpropagation process in CNN convolutional layers requires additional steps.

---

[120] The dropout rate refers to the proportion of neurons that are randomly ignored during training in order to reduce overfitting. This simple yet powerful technique helps models generalise better to unseen data.

The training process of a CNN is described below, with a special focus on how backpropagation works through convolutional layers. The training process consists of three main steps:

1. Forward propagation (feature extraction and prediction)

The network's weights (filters) and biases are usually initialized with small random values. An input (e.g., an image) is fed into the network and passes through convolutional, pooling and fully connected layers. Each convolutional layer applies filters to extract hierarchical features such as edges, textures and shapes. The fully connected layers then map these features onto class probabilities using activation functions such as *softmax*. Based on the current parameters, the network produces an output (prediction).

2. Loss computation (error calculation)

The predicted output is compared to the true label using a loss function (e.g. Categorical Cross-Entropy (CCE) loss for classification – see Section 4.3.2.6). Loss quantifies the difference between the predicted and actual labels. Higher loss indicates that the network has made poor predictions and requires weight adjustments.

3. Backpropagation and parameter update

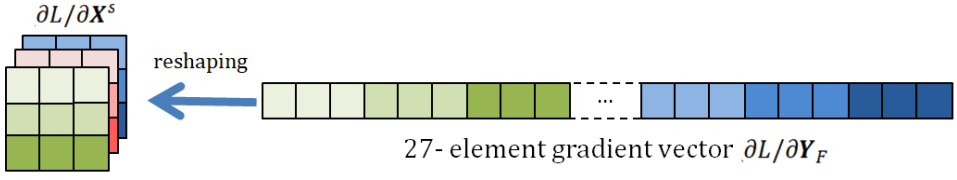
The network uses backpropagation to compute the gradients of the loss with respect to its parameters. This involves applying the chain rule to every layer – including the convolutional layers – to determine how to update the weights and biases in order to minimise the loss. Backpropagation calculates the total gradient for each filter weight and bias by adding together the contributions from all positions. Then, parameter updates are performed using optimisation algorithms (e.g. gradient descent or Adam), which subtract a portion of the gradient (scaled by the learning rate) from the current parameter values. In convolutional layers, weight sharing means that each filter's weight is updated using information from many different spatial locations. This ensures that the learned features effectively capture patterns across the entire input.

- **Backpropagation through FC layers**. The first step involves propagating errors backwards from the output layer through the fully connected layers to the flatten layer using the chain rule. This process of backpropagation is essentially the same as that in FNNs (see Section 4.3.2.8).

- **Backpropagation through a flatten layer**. In backpropagation, the gradients with respect to the output Y_F of the flatten layer (a vector) are computed by an FC layer. Since flattening is merely a reshaping operation, its derivative is the identity mapping (with the exception of reordering). This means that the gradient from the FC layer, $\partial L/\partial \boldsymbol{X}^s$, is reshaped back to the original dimensions of the input structure. For example, if F denotes the flattening operation, $\boldsymbol{X}^s$ the input (i.e. the output of the last convolutional/pooling layer) and $Y_F = F(\boldsymbol{X}^s)$ then the gradient with respect to $\boldsymbol{X}^s$ is given by

$$\frac{\partial L}{\partial \boldsymbol{X}^s} = reshape\left(\frac{\partial L}{\partial Y_F}\right).$$

For example, if the output $\boldsymbol{X}^s$ from the convolution and pooling layers is a (3x3)x3 structure, flattening it yields a 27-element vector. The FC also returns a 27-element vector gradient. This gradient is then reshaped back into a (3x3)x3 structure so that each component of the flattened gradient aligns with the corresponding spatial position in $\boldsymbol{X}^s$.



- **Backpropagation through a pooling layer**. Let us consider backpropagation in the context of max pooling. During backpropagation, a gradient is computed for each element Y_{klc} of the output $Y_c = [y_{klc}]$ produced by the pooling layer. Let us denote the gradient from the subsequent layers

with respect to the pooled output y_{klc} as $\partial L/\partial y_{klc}$. For each pooling window, the pooling operation previously selected the maximum value from a input feature map. Mathematically, if $X_c = [x_{ijc}]$ is a input feature map, then for a particular window $\mathbb{w}_{kl}$ we have (see Section 4.3.5.1)

$$y_{klc} = max\{x_{ijc} : (i,j) \in \mathbb{w}_{kl}\}.$$

Then the gradient with respect to each element x_{ijc} in that window is

$$\frac{\partial L}{\partial x_{ijc}} = \begin{cases} \partial L/\partial y_{klc}, & if\, x_{ijc} = y_{klc}\ (the\ \max element) \\ 0, & otherwise \end{cases}.$$

If there are ties, meaning that more than one element in the window equals the maximum, the gradient can be split equally between them or handled according to a predetermined rule.

In the case of max pooling, only the input element that was the maximum value within the window during the forward pass receives a non-zero gradient from the upstream layer. The other elements in that window do not contribute to y_{klc} (as they were 'surpassed' by the maximum value), and therefore receive a gradient of zero.

During the forward pass, many ML frameworks record the index of the maximum value for each pooling window, often called 'argmax indices'. During backpropagation, these indices are used to populate an output gradient structure with the same dimensions as the input. The gradient is inserted at the position of the maximum value, with zeros filling in the remaining positions. Note that the pooling layer reduces the spatial dimensions during the forward pass. During the backward pass, therefore, the gradient with respect to each reduced (pooled) output must be 'expanded' (or redistributed) back to the corresponding locations in the original, larger input structure.

- **Backpropagation through convolution layers**. Consider a simple case with one input channel $X = [x_{ij}]$ and one filter with only one slice: $W = [w_{kl}], k, l = 1, 2, \dots, f$. Then for a specific spatial location (i, j). the output $Y = [y_{ij}]$ is computed as (see Section 4.3.5.1)

$$y_{ij} = \sum_{k=1}^{f} \sum_{l=1}^{f} w_{kl} x_{i+k-1, j+l-1} + b.$$

During the backward pass, assume that the upstream gradient $\delta Y = \partial L/\partial Y$ has been computed for the output of the convolutional layer. Now, our goal is to compute:

- The gradient of the loss with respect to the filter weights $\partial L/\partial W$.

For each weight w_{kl} the gradient is computed by 'sliding' over the input X and correlating it with the error term from δY:

$$\frac{\partial L}{\partial w_{kl}} = \sum_{i,j} x_{i+k-1, j+l-1} \delta y_{ij},$$

where δy_{ij} is the error gradient from next layer (backpropagated) at position (i, j).

- The gradient with respect to the bias $\partial L/\partial b$.

Bias b is associated with the filter W and its gradient is the sum of the upstream gradients over the spatial dimensions:

$$\frac{\partial L}{\partial b} = \sum_{i,j} \delta y_{ij}.$$

- The gradient with respect to the input $\partial L/\partial X$.

The gradient with respect to the input is derived by 'distributing' the upstream error back through the convolution operation. For a particular element of the input x_{pq}, we accumulate contributions from all the filters whose receptive fields include x_{pq}. In the case of a single filter, we have

$$\frac{\partial L}{\partial X_{pq}} = \sum_{i,j} \delta y_{ij}\, w_{p-i, q-j}.$$

Otherwise, we would sum over all filters. The reason is that each filter is used multiple times across the input, so we need to accumulate gradients from every location where the filter was applied.

If the convolution uses strides greater than 1 or padding, these parameters affect how the gradients are mapped back to the input. The dimensions of δY must be appropriately 'unpacked' to match the dimensions of X. If the results of the convolution are passed through an activation function (e.g. $ReLU$), the gradient must also propagate through the derivative of that function.

- **Update of parameters**. After computing the gradients, standard optimisation methods are used to update the parameters. The most basic update rule is provided by gradient descent. For a parameter θ (which could be a filter weight w or a bias b), the update rule is

$$\theta \leftarrow \theta - \alpha \frac{\partial L}{\partial \theta},$$

where α is is the learning rate (a hyperparameter controlling the step size).

In standard backpropagation for CNNs, all gradients are first computed during the backward pass using the current (unchanged) parameters. Once the entire backward pass is complete, the optimiser uses these computed gradients to update the parameters simultaneously. These parameters are the filter weights and biases. This is the final step in the training process, whereby updating the parameters reduces the loss. This simultaneous update ensures consistency – every gradient was computed based on the same set of parameter values.

In summary, training a CNN involves two stages: a forward pass, which computes features via convolution and nonlinear activation, and a backward pass, which applies the chain rule to propagate error gradients. The CNN then uses these gradients to update its parameters, learning to extract increasingly abstract features from the data and thereby minimising the loss over time.

4.3.5.3. Limitations and challenges of CNNs

CNNs have many limitations and challenges in common with other deep architectures, such as MLPs, RNNs and Transformers. These limitations are rooted in their depth, nonlinear learning dynamics and data-driven nature. For example, they require massive labelled datasets, are prone to overfitting and generalisation gaps, and require very high computational resources. These issues were discussed in Section 4.3.2.9.

However, although CNNs have impressive capabilities, they are subject to several inherent architectural challenges that can significantly impact their performance and reliability. These challenges arise from the design and operational foundations of CNNs, presenting intricate obstacles for scholars and practitioners alike ([L4]).

Let us take a closer look at some of the key limitations of CNNs ([L4], [M9], [M10]).

× Spatial inductive bias and stationarity assumption. CNNs owe much of their success in image-processing tasks to their built-in assumptions about image structure. Two core principles – spatial inductive bias and locality assumptions – determine how CNNs process and learn from grid-structured data. Inductive bias means assuming that a certain type of spatial structure is present in the data ([M9]).

In CNNs, the spatial inductive bias is manifested through translation-equivariant filters and the stationarity assumption. As filters are shared across all spatial locations, a feature detected in one part of the image will be recognised elsewhere with the same pattern. The stationarity assumption means that CNNs do not expect the statistics of local patterns (such as edges and textures) to change across the image. This enables a small set of filters to generalise more widely, rather than learning separate weights for each position.

These biases speed up training and enhance generalisation on natural images, where local features tend to be repeated across different regions. However, the downside is that a CNN

struggles to recognise identical patterns in different contexts and is ineffective with non-stationary data, such as medical scans with region-specific anomalies.

× **Locality assumption**. Locality assumptions restrict the view of each neuron to a small area of the input, known as its receptive field. The early layers only process small neighbourhoods (e.g. 3×3 or 5×5 pixels), capturing basic features such as edges and corners. Therefore, there is no direct global context. A convolutional layer can only 'see' a small neighbourhood. Capturing long-range dependencies requires either stacking many layers or using dilated convolutions, both of which complicate the architecture and training process. To enlarge the effective receptive field, CNNs must be made deeper. This can lead to vanishing/exploding gradients, more parameters, longer training times and greater memory and computing requirements.

× **Interpretability**. Early convolutional filters (such as those detecting edges and textures) are somewhat interpretable, but deeper layers encode highly abstract patterns that cannot be easily visualised. Explaining the decision-making process through dozens or hundreds of feature maps remains an open research problem.

× **Vulnerability to adversarial attacks**. Adversarial attacks involve making tiny, often barely noticeable, changes to images that cause convolutional neural networks to make incorrect predictions. These perturbations exploit the model's sensitivity in high-dimensional input spaces, causing it to become unreliable even when human observers see no change in the image. This vulnerability poses significant challenges for safety-critical applications such as autonomous driving, medical imaging and security systems.

For instance, an image recognition system may correctly identify a stop sign 99 % of the time, yet a carefully designed modification (e.g. a small sticker on the sign) could cause it to misclassify the same image as a speed limit sign.

Attackers continue to exploit newer architectures, such as Vision Transformers, which reveals that no vision model is inherently immune.

4.3.6. Recurrent Neural Networks (RNNs)

The key to the success of any neural architecture lies in designing the network's structure with a semantic understanding of the relevant domain. CNNs largely adhere to this principle, employing sparse connections and a high degree of parameter sharing in a domain-specific manner. In a particular layer, the value of a feature is connected only to a local spatial region in the previous layer, and a consistent set of shared parameters is used across the full spatial footprint of the image. This type of architecture can be viewed as domain-aware regularisation.

A significant level of domain-aware regularisation is also present in *Recurrent Neural Networks* (RNNs), which share parameters from different temporal periods based on the assumption that temporal dependencies remain constant over time. Unlike CNNs, which are based on an understanding of spatial relationships, RNNs are designed to understand temporal relationships. ([A2])

Up until now, our focus has been primarily on fixed-length data. For instance, when we examined ANNs in Section 4.3.2, we assumed that each feature vector $X = [x_i]$ contained a fixed number of components, with each numerical feature x_i corresponding to a specific attribute. In Section 4.3.5, we moved on to image data, where the inputs consist of the pixel values at each coordinate in an image. There, CNNs were necessary in order to handle the hierarchical structure. However, our data was still of fixed size. Our objective was to develop a model that could analyse a single image and provide a single prediction. But how should we approach a sequence of images, such as a video?

RNNs are a class of artificial neural network that has become more popular in recent years. Unlike feedforward networks, they have recurrent connections. The major benefit of these connections is that they enable the network to refer to previous states, allowing it to process

arbitrary sequences of input. They are widely used in tasks involving time-series, text, audio, or any data where order and context matter. ([121])

The following are the key features of RNNs:

- Sequence processing. In contrast to standard neural networks, RNNs take into account not only the present input, but also previous inputs. This renders them well-suited to sequential tasks. The network architecture incorporates loops to facilitate the persistence of information.

- Hidden state. The concept of a 'hidden state' is central to how RNNs work. The network maintains this state, which acts as a memory, capturing context from previous time steps. At each time step, the hidden state is updated based on the current input and the previous hidden state.

- Shared weights. The same set of weights is applied to each time step, thereby ensuring the efficiency of the model for variable-length sequences.

4.3.6.1. Architecture of a traditional RNN

The basic structure of an RNN consists of the following components ([U2]):

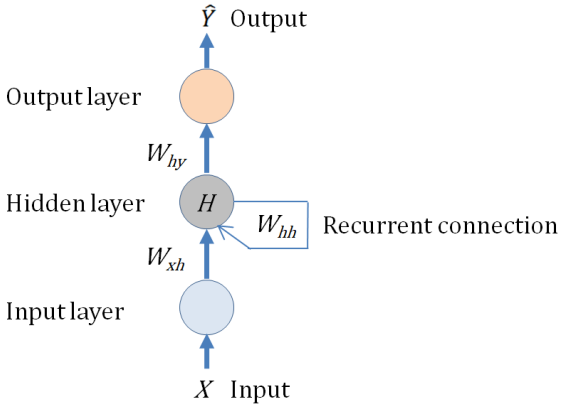
- **Input layer**. This layer receives sequential data. For example, in natural language processing tasks, each element of the sequence could represent a word or character.

- **Hidden layer**. The hidden layer is the core component of an RNN. It maintains an internal, or hidden, state that evolves as the network processes each element of the sequence. This hidden state captures information from previous time steps, enabling the network to preserve context and model dependencies within the sequence.

- **Recurrent connection**. This is a key feature of RNNs. It consists of a set of weights that loop back to the hidden layer from the previous time step. This loop allows the RNN to maintain and update its hidden state as it processes each element of the sequence. Consequently, the recurrent connection provides the network with the ability to remember and utilise information from the past.

- **Output layer**. This layer produces predictions based on information from the hidden state. The number of neurons in this layer varies depending on the specific task. In a language model, for instance, it may be a softmax layer that predicts the next word in a sequence.

The figure below illustrates the recurrent unit, which is a fundamental building block of RNNs. This unit maintains a hidden state – a type of memory that is updated at each time step, based on the current input and the previous hidden state.

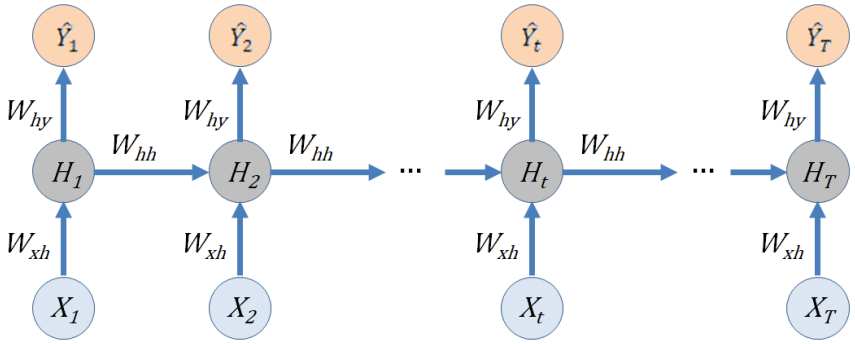


[121] Recurrent neural networks are Turing-complete ([S7]). This means that, with sufficient data and computing power, a recurrent neural network can simulate any algorithm. However, the Turing completeness is not very useful in practice, as achieving this goal requires an unrealistic amount of data and computational resources. ([A2])

In an RNN, every 'time-step' input X_t is treated as a vector, even if its length is 1. If the input sequence contains word tokens, each token is usually converted into a one-hot vector in $\mathbb{R}^v$ (where v is the vocabulary size). This vector is then multiplied by an embedding matrix ([122]) to ensure that X_t is in $\mathbb{R}^n, n \ll v$. When a sequential input $\boldsymbol{X} = [X_1, X_2, \ldots, X_t, \ldots X_T]$ of length T is provided, the recurrent unit takes two vectors at each time step t: X_t – the current input and H_{t-1} – the hidden (memory) state from the previous step. It then produces an updated hidden state H_t and often an output $\hat{Y}_t$. In many applications, such as language modelling, the output is not produced at each time t, but only at the last time stamp at the end of the sentence.

Please note that in RNNs, the time-step index t does not necessarily represent the passage of time in the real world. Although we refer to it as 'time', it actually indicates position within the sequence.

The next figure shows an unfolded version of a recurrent unit. This is obtained by unrolling the network structure for the entire input sequence at different, discrete times.



It is clear that this differs from a typical multi-level ANN, which applies different parameters at each level. An RNN uses the same parameters (W_{xh}, W_{hh} and W_{hy}) and performs the same computation at each time instance t on different inputs from the same sequence. The weight matrix W_{xh} governs the connection from the input layer X to the hidden layer H. W_{hh} represents the weight matrix associated with connections within the hidden layer. W_{hy} is the weight matrix controlling the connection from the hidden layer to the output layer. Sharing these parameters enables the RNN to capture temporal dependencies and process sequential data efficiently by retaining information from previous inputs in its current hidden state.

The figure above illustrates a case in which there is both an input and an output at each time instance t. In practice, either the input or the output units may be missing at a given time. Which inputs and outputs are missing would depend on the specific application. Generally, any subset of inputs or outputs can be absent in a given application. The main types of RNNs include ([D2]):

- **One-to-one.** One-to-one RNN models have a single input and output. Also known as a 'vanilla' neural network, this architecture is used in traditional neural networks and for general machine learning tasks such as image classification.

- **One-to-many (missing inputs).** One-to-many RNNs have a single input and multiple outputs. This makes them ideal for image captioning and music generation, as they use a single input (such as a keyword) to produce multiple outputs (such as a sentence).

- **Many-to-one (missing outputs).** This model has multiple inputs and a single output. It is commonly applied to tasks such as sentiment analysis or text classification.

---

[122] One-hot encoding is a technique used to represent categorical data as a numerical vector. Each unique category is represented by a binary column, where a value of 1 indicates the presence of the category and a value of 0 indicates its absence. Word embedding techniques such as *Word2Vec* convert words into low-dimensional, dense vectors – see Section 5.1.3 for more details.

- **Many-to-many.** Many-to-many models map multiple inputs to multiple outputs. Machine translation and *Named Entity Recognition* (NER) ([123]) are based on many-to-many RNNs, which can structure several words or sentences into many different outputs (such as a new language or various categorisations).

In the following, we will assume that all inputs and outputs are present. However, it is easy to adapt this approach to cases where some inputs or outputs are missing, simply by removing the relevant terms from the forward propagation equations.

4.3.6.2. Training process of an RNN

The training process for a simple RNN is described below. It consists of the following main steps:

1. Setup and initialisation

First, we define the model architecture by choosing:

- Input size n (dimensionality of each X_t)
- Hidden size d (the number of hidden units)
- Output size m
- Learning rate α
- Number of epochs
- Optimiser, etc.

Let's assume that we have a training set $\{(\boldsymbol{X}^i, \boldsymbol{Y}^i) : i = 1, 2, \dots, m\}$ consisting of sequences

$$\boldsymbol{X}^i = [X_1^i, X_2^i, \dots, X_{T_i}^i] \text{ and } \boldsymbol{Y}^i = [Y_1^i, Y_2^i, \dots, Y_{T_i}^i],$$

where $X_t^i \in \mathbb{R}^n$ and $Y_t^i \in \mathbb{R}^m$, $t = 1,2, \dots, T_i$. The weight matrices $W_{xh} \in \mathbb{R}^{d \times n}$, $W_{hh} \in \mathbb{R}^{d \times d}$ and $W_{hy} \in \mathbb{R}^{m \times d}$, as well as biases $B_h \in \mathbb{R}^d$ and $B_y \in \mathbb{R}^m$, are usually initialised with small random values (e.g. using the Xavier or He method [K8]). Moreover, we set the initial hidden states H_0^i to the null vector 0.

2. Forward pass (unrolling in time)

For each sequence $\boldsymbol{X}^i$, we 'unroll' the RNN for its length T_i by creating T_i copies of the same cell, each of which shares the same weights. This means that, for each timestamp t, the following computations are performed:

$$H_t^i = f\left(W_{xh} \cdot X_t^i + W_{hh} \cdot H_{t-1}^i + B_h\right)$$
$$\hat{Y}_t^i = g(W_{hy} \cdot H_t^i + B_y),$$

where f is typically *tanh* or *ReLU*, and g depends on the task (*softmax* for classification and *identity* for regression).

3. Loss computation

The loss function $\boldsymbol{L}$ of all time steps is defined based on the loss at every time step as follows:

$$\boldsymbol{L}(\hat{\boldsymbol{Y}}^i, \boldsymbol{Y}^i) = \sum_{t=1}^{T} L(\hat{Y}_t^i, Y_t^i),$$

where L is, for example, the cross-entropy loss function for classification or the mean-squared error loss function for regression.

4. **Backpropagation through time (BPTT)**

---

[123] *Named Entity Recognition* (NER) is a subfield of Natural Language Processing (NLP) which involves the automatic identification and categorisation of 'named entities' in text, such as people, organisations, locations, etc.

RNNs use a variant of backpropagation tailored for sequential data, called *Backpropagation Through Time* (BPTT). Gradients are calculated for the loss function with respect to each parameter by unrolling the RNN over all time steps. The unrolled RNN is essentially an FNN with the property that the same parameters (weights and biases) are repeated throughout the unrolled network, appearing at each time step. Then, just as in any feedforward neural network, we can apply the chain rule, backpropagating gradients through the unrolled net ([Z3]).

Starting with the derivative $\partial \boldsymbol{L}/\partial \widehat{\boldsymbol{Y}}^i$ of the loss function with respect to the outputs, the straightforward backpropagation as described in Section 4.3.2 can be used to compute the gradients ([124])

$$\frac{\partial \boldsymbol{L}}{W_{xh}}, \frac{\partial \boldsymbol{L}}{W_{hh}}, \frac{\partial \boldsymbol{L}}{W_{hy}}, \frac{\partial \boldsymbol{L}}{B_h}, \frac{\partial \boldsymbol{L}}{B_y}$$

in different layers of the unrolled temporal network. However, when computing gradients with respect to the weights W_{xh}, W_{hh} and W_{hy}, the sharing of these weight matrices across different temporal layers must also be considered. At first, it may seem difficult to calculate the loss gradient with shared weights due to the complexity of the loss function. Fortunately, there is a useful technique that can simplify the calculation.

The solution lies in introducing the temporal variables, W_{xh}^t, W_{hh}^t and W_{hy}^t ([A2]). These are defined as copies of the corresponding matrices, but where each one is only used at time step t. We can then use $\partial/\partial W^t$ to represent the contribution of the weights at time step t to the gradient. The gradient of the edge weights in the backward direction is then computed on the unrolled network without regarding the fact that the weights are shared across different time layers. In other words, it is assumed that the weights W_{xh}^t, W_{hh}^t and W_{hy}^t are different for each time stamp. Consequently, conventional backpropagation can be used to compute the following derivatives

$$\frac{\partial \boldsymbol{L}}{W_{xh}} = \sum_{t=1}^{T} \frac{\partial L}{\partial W_{xh}^t}$$
$$\frac{\partial \boldsymbol{L}}{W_{hh}} = \sum_{t=1}^{T} \frac{\partial L}{\partial W_{hh}^t}$$
$$\frac{\partial \boldsymbol{L}}{W_{hy}} = \sum_{t=1}^{T} \frac{\partial L}{\partial W_{hy}^t}.$$

To gain a better understanding of this process, we will look at how the gradient is computed with respect to the matrix W_{hh}. Let us calculate the gradient $\frac{\partial L}{\partial W_{hh}^\tau}$ at a specific time step τ and for a given index i. Using the chain rule, we obtain ([D3]) ([125])

$$\frac{\partial L}{\partial W_{hh}^\tau} = \frac{\partial L}{\partial \hat{Y}_\tau^i} \cdot \frac{\partial \hat{Y}_\tau^i}{\partial H_\tau^i} \cdot \frac{\partial H_\tau^i}{\partial W_{hh}^\tau}.$$

However, since we are dealing with recurrent neural networks, H_τ^i depends on the hidden states at all previous time steps. Once again, we must sum the contributions of each timestep to find the total gradient, yielding the following equation

---

[124] In multivariate calculus, derivatives are extended to encompass vector and matrix inputs, generalising beyond simple scalar functions. In this section, we consider the derivatives of the following functions: (1) $f: \mathbb{R}^m \to \mathbb{R} \Rightarrow \frac{\partial f}{\partial X} = \left[\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_m}\right]^T$, (2) $f: \mathbb{R}^d \to \mathbb{R}^m \Rightarrow \frac{\partial f}{\partial X} = \left[\frac{\partial f_i}{\partial x_j}\right]_{j=1,\dots,d}^{i=1,\dots,m}$, i.e. $\frac{\partial f}{\partial X}$ is an $m \times d$ matrix, (3) $f: \mathbb{R}^{k \times l} \to \mathbb{R}^p \Rightarrow \frac{\partial f}{\partial X} = \left[\frac{\partial f_1}{\partial X}, \dots, \frac{\partial f_p}{\partial X}\right]$, where $f_1, \dots, f_p$ are (scalar) components of f. Consequently, the derivative $\frac{\partial f}{\partial X}$ is a $p \times k \times l$ structure, i.e. a stack of p matrices, each of size $k \times l$. Therefore, $\frac{\partial f}{\partial X}$ is a 3D tensor.

[125] More precisely, $\frac{\partial L}{\partial W_{hh}^\tau} = \left(\frac{\partial L}{\partial \hat{Y}_\tau^i}\right)^T \cdot \frac{\partial \hat{Y}_\tau^i}{\partial H_\tau^i} \cdot \frac{\partial H_\tau^i}{\partial W_{hh}^\tau}$. The product $\left(\frac{\partial L}{\partial \hat{Y}_\tau^i}\right)^T \cdot \frac{\partial \hat{Y}_\tau^i}{\partial H_\tau^i}$ is a d-dimensional vector. Multiplying this by the $d \times d \times d$ tensor $\frac{\partial H_\tau^i}{\partial W_{hh}^\tau}$ yields the $d \times d$ matrix $\frac{\partial L}{\partial W_{hh}^\tau}$ (see [P16] for more details on tensors).

$$\frac{\partial L}{\partial W_{hh}^{\tau}} = \sum_{t=1}^{\tau} \frac{\partial L}{\partial \hat{Y}_{\tau}^{i}} \cdot \frac{\partial \hat{Y}_{\tau}^{i}}{\partial H_{\tau}^{i}} \cdot \frac{\partial H_{\tau}^{i}}{\partial H_{t}^{i}} \cdot \frac{\partial H_{t}^{i}}{\partial W_{hh}^{\tau}}.$$

Unfortunately, this is not the end of the story. When computing $\frac{\partial H_t^i}{\partial W_{hh}^{\tau}}$, we still need to consider that H_{τ}^{i} depends on previous time steps. Therefore, our final equation representing the gradient $\frac{\partial L}{\partial W_{hh}^{\tau}}$ is as follows ([126])

$$\frac{\partial L}{\partial W_{hh}^{\tau}} = \sum_{t=1}^{\tau} \frac{\partial L}{\partial \hat{Y}_{\tau}^{i}} \cdot \frac{\partial \hat{Y}_{\tau}^{i}}{\partial H_{\tau}^{i}} \cdot \left(\prod_{\rho=t}^{\tau-1} \frac{\partial H_{\rho}^{i}}{\partial H_{\rho-1}^{i}}\right) \cdot \frac{\partial H_{\tau}^{i}}{\partial H_{t}^{i}} \cdot \frac{\partial H_{t}^{i}}{\partial W_{hh}^{\tau}}.$$

Please note that we have used matrix calculus notation, whereby the derivative of a function with respect to a matrix is defined as a matrix of element-wise derivatives.

One computational issue with BPTT is that the underlying sequences may be very long, resulting in a large number of network layers. This can lead to problems with computation, convergence and memory usage. One way to address this problem is to use truncated backpropagation through time. In this approach, state values are computed correctly during forward propagation, but backpropagation updates are only performed over short segments of the sequence ([A2]).

5. Parameter update

After computing the gradients, standard optimisation methods are used to update the parameters. For instance, the gradient descent method provides the following update rule for a parameter θ (which could be a weight or a bias):

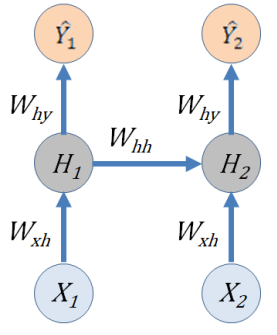
$$\theta \leftarrow \theta - \alpha \frac{\partial \boldsymbol{L}}{\partial \theta},$$

where α is is the learning rate.

6. Iteration

Steps 2–5 are repeated for many passes (epochs) through the dataset, with sequences being shuffled and batches being formed as appropriate.

Example. This example provides a complete illustration of the process of training a simple RNN on a single sequence for one epoch. We use mean-squared error loss (*MSE*) and straightforward stochastic gradient descent (*SGD*) updates. Our toy model has the following structure



- *Setup and initialisation*. We choose a minimal setting:
  - Input size $n = 1$ (dimensionality of each X_t)
  - Hidden size $d = 1$ (the number of hidden units)
  - Output size $m = 1$
  - Learning rate $\alpha = 0.1$

[126] The $\prod$ symbol means 'product' and by definition $\prod_{i=1}^{n} x_i = x_1 \cdot x_2 \cdot \ldots \cdot x_n$.

- Activation: *tanh* in the hidden layer and *identity* at the output
- Loss function: mean squared error (*MSE*).

We have a single training instance $\{(X, Y)\}$ consisting of sequences of length $T = 2$

$$\boldsymbol{X} = [X_1, X_2] \text{ and } \boldsymbol{Y} = [Y_1, Y_2],$$

where $X_1 = 0.5, X_2 = -0.5$ and $Y_1 = 0, Y_2 = 1$ (all one-dimensional, i.e. scalars). The weight matrices W_{xh}, W_{hh} and W_{hy} are one-dimensional, with initial values $W_{xh} = 0.7$, $W_{hh} = 0.5$ and $W_{hy} = 1.2$. Furthermore, the biases are initialised as follows: $B_h = 0$ and $B_y = 0.1$.

- *Forward pass (unrolling in time)*. We set the initial hidden state to $H_0 = 0$ and compute:
  - time $t = 1$

$$o_1 = W_{xh} \cdot X_1 + W_{hh} \cdot H_0 + B_h = 0.7 \cdot 0.5 + 0.5 \cdot 0 + 0 = 0.35$$
$$H_1 = tanh(o_1) = tanh(0.35) \approx 0.336$$
$$\hat{Y}_1 = id(W_{hy} \cdot H_1 + B_y) = 1.2 \cdot 0.336 + 0.1 = 0.5032.$$

Recall that $tanh(z) = (e^z - e^{-z})/(e^z + e^{-z})$ (see Section 4.3.2.6)

  - time $t = 2$

$$o_2 = W_{xh} \cdot X_2 + W_{hh} \cdot H_1 + B_h = 0.7 \cdot (-0.5) + 0.5 \cdot 0.336 + 0 = -0.182$$
$$H_2 = tanh(o_2) = tanh(-0.182) \approx -0.18$$
$$\hat{Y}_2 = id(W_{hy} \cdot H_2 + B_y) = 1.2 \cdot (-0.18) + 0.1 = -0.116.$$

- *Loss computation*. We use the mean of squared errors

$$L = \frac{1}{2}\left[\left(\hat{Y}_1 - Y_1\right)^2 + \left(\hat{Y}_2 - Y_2\right)^2\right] = 0.5 \cdot [(0.5032 - 0)^2 + (-0.116 - 1)^2] \approx 0.7491.$$

- *Backpropagation through time*. First, let us compute the gradients of the output layer. Since $\delta_t = \partial L/\partial \hat{Y}_t = \hat{Y}_t - Y_t$ $(t = 1, 2)$ we obtain

$$\delta_1 = 0.5032, \ \delta_2 = -1.116.$$

Hence ([127])

$$\begin{aligned}\frac{\partial L}{\partial W_{hy}} &= \frac{1}{2}\frac{\partial}{\partial W_{hy}}\left(\hat{Y}_1 - Y_1\right)^2 + \frac{1}{2}\frac{\partial}{\partial W_{hy}}\left(\hat{Y}_2 - Y_2\right)^2 \\ &= \left(\hat{Y}_1 - Y_1\right)\frac{\partial}{\partial W_{hy}}\hat{Y}_1 + \left(\hat{Y}_2 - Y_2\right)\frac{\partial}{\partial W_{hy}}\hat{Y}_2 \\ &= \left(\hat{Y}_1 - Y_1\right)H_1 + \left(\hat{Y}_2 - Y_2\right)H_2 \\ &= \delta_1 H_1 + \delta_2 H_2 \\ &= 0.5032 \cdot 0.336 + (-1.116) \cdot (-0.18) \approx 0.37\end{aligned}$$

and

$$\begin{aligned}\frac{\partial L}{\partial B_y} &= \frac{1}{2}\frac{\partial}{\partial B_y}\left(\hat{Y}_1 - Y_1\right)^2 + \frac{1}{2}\frac{\partial}{\partial B_y}\left(\hat{Y}_2 - Y_2\right)^2 \\ &= \left(\hat{Y}_1 - Y_1\right)\frac{\partial}{\partial B_y}\hat{Y}_1 + \left(\hat{Y}_2 - Y_2\right)\frac{\partial}{\partial B_y}\hat{Y}_2 \\ &= \left(\hat{Y}_1 - Y_1\right) + \left(\hat{Y}_2 - Y_2\right) \\ &= \delta_1 + \delta_2 \\ &= 0.5032 - 1.116 = -0.6128.\end{aligned}$$

Now we will compute the gradients of the hidden layer. Recall that ([D4])

---

[127] Writing $\partial L/\partial W_{hy} \approx 0.37$ is slightly imprecise. What we mean is that, for specific parameter values, the value of $\partial L/\partial W_{hy}$ is approximately 0.37: $\frac{\partial L}{\partial W_{hy}}|_{current\ point} \approx 0.37$. However, $|_{current\ point}$ is usually omitted under the convention that the derivative is always evaluated at the network's current state.

$$tanh'(z) = 1 - [tanh\,(z)]^2.$$

We have

$$\begin{aligned}
\frac{\partial L}{\partial H_2} &= \frac{1}{2}\frac{\partial}{\partial H_2}\left(\hat{Y}_1 - Y_1\right)^2 + \frac{1}{2}\frac{\partial}{\partial H_2}\left(\hat{Y}_2 - Y_2\right)^2 \\
&= 0 + \left(\hat{Y}_2 - Y_2\right)\frac{\partial}{\partial H_2}(W_{hy}\cdot H_2 + B_y) \\
&= \delta_2 \cdot W_{hy} \\
&= -1.116 \cdot 1.2 = -1.3392, \\
\frac{\partial L}{\partial o_2} &= \frac{1}{2}\frac{\partial}{\partial o_2}\left(\hat{Y}_1 - Y_1\right)^2 + \frac{1}{2}\frac{\partial}{\partial o_2}\left(\hat{Y}_2 - Y_2\right)^2 \\
&= 0 + \left(\hat{Y}_2 - Y_2\right)\frac{\partial}{\partial o_2}(W_{hy}\cdot H_2 + B_y) \\
&= \delta_2 \frac{\partial}{\partial o_2}(W_{hy}\cdot H_2) \\
&= \delta_2 \cdot W_{hy}\cdot \frac{\partial}{\partial o_2} tanh(\mathrm{o}_2)) \\
&= \frac{\partial L}{\partial H_2}\cdot [1 - (tanh\,(\mathrm{o}_2))^2] \\
&= \frac{\partial L}{\partial H_2}\cdot [1 - (H_2)^2] \\
&= -1.3392 \cdot [1 - (-0.18)^2] \approx -1.295, \\
\frac{\partial o_2}{\partial H_1} &= \frac{\partial}{\partial H_1}(W_{xh}\cdot X_2 + W_{hh}\cdot H_1 + B_h) \\
&= W_{hh} = 0.5.
\end{aligned}$$

For $t = 2$ we have

$$\begin{aligned}
\frac{\partial L}{\partial W_{xh}^{(2)}} &= \frac{\partial L}{\partial o_2}\cdot\frac{\partial o_2}{\partial W_{xh}} \\
&= \frac{\partial L}{\partial o_2}\cdot\frac{\partial}{\partial W_{xh}}(W_{xh}\cdot X_2 + W_{hh}\cdot H_1 + B_h) \\
&= \frac{\partial L}{\partial o_2}\cdot X_2 \\
&= -1.295\cdot(-0.5) = 0.6475, \\
\frac{\partial L}{\partial W_{hh}^{(2)}} &= \frac{\partial L}{\partial o_2}\cdot\frac{\partial o_2}{\partial W_{hh}} \\
&= \frac{\partial L}{\partial o_2}\cdot\frac{\partial}{\partial W_{hh}}(W_{xh}\cdot X_2 + W_{hh}\cdot H_1 + B_h) \\
&= \frac{\partial L}{\partial o_2}\cdot H_1 \\
&= -1.295\cdot 0.336 = -0.4351, \\
\frac{\partial L}{\partial B_h^{(2)}} &= \frac{\partial L}{\partial o_2}\cdot\frac{\partial o_2}{\partial B_h} \\
&= \frac{\partial L}{\partial o_2}\cdot\frac{\partial}{\partial B_h}(W_{xh}\cdot X_2 + W_{hh}\cdot H_1 + B_h) \\
&= \frac{\partial L}{\partial o_2}\cdot 1 \\
&= -1.295.
\end{aligned}$$

Now, let us consider the case when $t = 1$. We want to compute the total gradient of the loss with respect to h_1. Let us consider all the ways in which h_1 affects the loss L. There are two distinct paths. The first goes directly through the output at $t = 1$: $H_1 \to \hat{Y}_1 \to L$. This gives rise to

$$\begin{aligned}
\frac{\partial L^{(output)}}{\partial H_1} &= \frac{1}{2}\frac{\partial}{\partial H_1}\left(\hat{Y}_1 - Y_1\right)^2 + \frac{1}{2}\frac{\partial}{\partial H_1}\left(\hat{Y}_2 - Y_2\right)^2 \\
&= \left(\hat{Y}_1 - Y_1\right)\frac{\partial}{\partial H_1}(W_{hy}\cdot H_1 + B_y) + 0
\end{aligned}$$

$$= \delta_1 \cdot W_{hy}$$
$$= 0.5032 \cdot 1.2 = 0.6038.$$

The second path is indirect, involving a recurrence into time step 2: $H_1 \rightarrow \mathrm{o}_2 \rightarrow \hat{Y}_2 \rightarrow L$. This contributes another gradient back into H_1, as given by the recurrence

$$\Delta H_1^{(rec)} = \frac{\partial L}{\partial o_2} \cdot \frac{\partial o_2}{\partial H_1} = -1.295 \cdot 0.5 = -0.6475.$$

As the total loss depends on both $\hat{Y}_1$ and $\hat{Y}_2$, where $\hat{Y}_1$ and $\hat{Y}_2$ are functions of H_1, the total derivative is the sum of the contributions from both paths

$$\frac{\partial L}{\partial H_1} = \frac{\partial L^{(output)}}{\partial H_1} + \Delta H_1^{(rec)} = 0.6038 - 0.6475 = -0.0437.$$

Consequently

$$\frac{\partial L}{\partial o_1} = \frac{\partial L}{\partial H_1} \cdot \frac{\partial H_1}{\partial o_1} = -0.0437 \cdot \frac{\partial}{\partial o_1} tanh(o_1)$$
$$= -0.0437 \cdot [1 - (tanh\ (\mathrm{o}_1))^2]$$
$$= -0.0437 \cdot [1 - 0.336^2] \approx -0.0388$$

and

$$\frac{\partial L}{\partial W_{hh}^{(1)}} = \frac{\partial L}{\partial o_1} \cdot \frac{\partial o_1}{\partial W_{hh}}$$
$$= \frac{\partial L}{\partial o_1} \cdot \frac{\partial}{\partial W_{hh}} (W_{xh} \cdot X_1 + W_{hh} \cdot H_0 + B_h)$$
$$= \frac{\partial L}{\partial o_1} \cdot H_0$$
$$= -0.0388 \cdot 0 = 0,$$
$$\frac{\partial L}{\partial W_{xh}^{(1)}} = \frac{\partial L}{\partial o_1} \cdot \frac{\partial o_1}{\partial W_{xh}}$$
$$= \frac{\partial L}{\partial o_1} \cdot \frac{\partial}{\partial W_{xh}} (W_{xh} \cdot X_1 + W_{hh} \cdot H_0 + B_h)$$
$$= \frac{\partial L}{\partial o_1} \cdot X_1$$
$$= -0.0388 \cdot 0.5 = -0.0194,$$
$$\frac{\partial L}{\partial B_h^{(1)}} = \frac{\partial L}{\partial o_1} \cdot \frac{\partial o_1}{\partial B_h}$$
$$= \frac{\partial L}{\partial o_1} \cdot \frac{\partial}{\partial B_h} (W_{xh} \cdot X_1 + W_{hh} \cdot H_0 + B_h)$$
$$= \frac{\partial L}{\partial o_2} \cdot 1$$
$$= -0.0388.$$

Now, gradients are to be summed over t:

$$\frac{\partial L}{\partial W_{xh}} = \frac{\partial L}{\partial W_{xh}^{(1)}} + \frac{\partial L}{\partial W_{xh}^{(2)}} = -0.0194 + 0.6475 = 0.6281,$$
$$\frac{\partial L}{\partial W_{hh}} = \frac{\partial L}{\partial W_{hh}^{(1)}} + \frac{\partial L}{\partial W_{hh}^{(2)}} = 0 - 0.4351 = -0.4351,$$
$$\frac{\partial L}{\partial B_h} = \frac{\partial L}{\partial B_h^{(1)}} + \frac{\partial L}{\partial B_h^{(2)}} = -0.0388 - 1.295 = -1.3338.$$

We have already computed the following gradients:

$$\frac{\partial L}{\partial W_{hy}} = 0.37 \text{ and } \frac{\partial L}{\partial B_y} = -0.6128.$$

- *Parameter update*. Apply $\theta \leftarrow \theta - \alpha \frac{\partial \boldsymbol{L}}{\partial \theta}$ with $\alpha = 0.1$

$$W_{xh}^{new} = 0.7 - 0.1 \cdot 0.6281 = 0.6372$$

$$W_{hh}^{new} = 0.5 - 0.1 \cdot (-0.4351) = 0.5435$$
$$B_h^{new} = 0.0 - 0.1 \cdot (-1.3338) = 0.1334$$
$$W_{hy}^{new} = 1.2 - 0.1 \cdot (0.37) = 1.163$$
$$B_y^{new} = 0.1 - 0.1 \cdot (-0.6128) = 0.1613.$$

- *Summary*. The following steps were taken during one epoch using our single sequence:
  - a forward pass to compute $H_t, \hat{Y}_t$ for $t = 1, 2$
  - computation of loss $L \approx 0.7491$
  - backpropagation through time to find partial derivatives of L with respect to W_{xh}, W_{hh}, B_h, W_{hy} and B_y
  - a single SGD update yielding new parameter values.

This concrete numerical walkthrough illustrates how gradients at each time step are accumulated and then used to adjust all RNN parameters in one epoch.

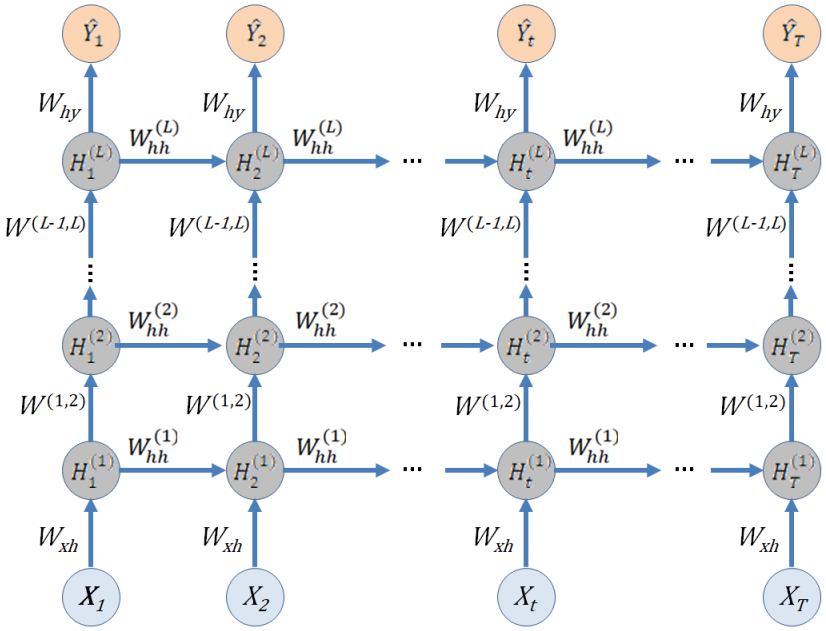
Remark. In pure gradient descent, the loss is computed for all training examples before the parameters are updated. This involves performing a full forward-and-backward pass through the entire dataset per epoch, resulting in an exact gradient of loss

$$\frac{\partial \boldsymbol{L}}{\partial \theta} = \sum_{i=1}^{N} \frac{\partial L}{\partial \theta} [f(\boldsymbol{X}^i; \theta), \boldsymbol{Y}^i].$$

No weight updates occur until every sample has contributed, meaning there is only one update at the end of each epoch. In contrast, SGD (or mini-batch SGD) computes the gradient on a single sample or small batch of samples, updating the weights immediately. This yields updates that are noisier but more frequent, and which often converge faster in practice.

4.3.6.3. Multi-layer Recurrent Neural Networks

For simplicity, the sections above considered a single-layer RNN. In practical applications, however, a multilayer architecture is typically employed to construct more complex models. The figure below illustrates a recurrent network comprising L layers.



An RNN with $L > 1$ layers has $2L - 1$ weight matrices for the hidden layers: one $W^{(l-1,l)}$ for the feed-forward connection from hidden layer $l - 1$ to layer l, and one $W_{hh}^{(l)}$ for the recurrent connection within layer l. These matrices are all independent of each other, and together they

determine the flow of information both across layers (vertically) and through time (horizontally). The recurrence equation of the hidden layers is as follows

$$H_t^{(l)} = f\left(W^{(l-1,l)} \cdot H_t^{(l-1)} + W_{hh}^{(l)} \cdot H_{t-1}^{(l)} + B_h^{(l)}\right).$$

If $l = 1$ then $H_t^{(0)}$ is just the input X_t and $W^{(0,1)} = W_{xh}$. For higher layers ($l > 1$), $H_t^{(l-1)}$ is the hidden vector from the layer below. Both contributions get summed (plus bias) and passed through the activation function f. The transformation from hidden to output layer remains the same as in single-layer RNNs.

In practice, two or three layers are standard. However, to avoid overfitting, a larger number of layers requires more training data.

4.3.6.4. Variants of RNN architectures

Although traditional RNNs can handle sequential data, they suffer from exploding and vanishing gradients. This occurs due to the extremely deep networks resulting from the unfolding of recurrent networks in the time dimension. During BPTT, gradients can become too small, leading to the vanishing gradient problem, or too large, resulting in the exploding gradient problem as they propagate backwards through time. In the case of vanishing gradients, the issue is that the gradient may converge during training, but this process can take a very long time. On the other hand, in the case of exploding gradients, a large gradient can lead to numerical instability during training. This type of instability is the direct result of successive multiplication with the (recurrent) weight matrix at various time stamps. This causes the model to deviate from the optimal solution, making it difficult for the network to converge to the global minimum.

- **Long Short Term Memory (LSTM)**. *Long Short Term Memory* networks – usually just called LSTMs – are a special kind of RNN capable of learning long-term dependencies. LSTM was initially proposed in [H8] and has since found widespread application in natural language processing. A traditional RNN that uses only multiplicative updates is only suitable for learning over short sequences. This means that it has good short-term memory but poor long-term memory. To address this problem, the recurrence equation for the hidden vector is modified using an LSTM, which has long-term memory ([A2]).

Each LSTM unit contains a cell state, which acts as the network's memory, as well as a hidden state at time t and the current input. There are also three gates that regulate the flow of information:

- *Forget gate* decides what information to discard from the cell state.
- *Input gate* decides what new information to store in the cell state.
- *Output gate* decides what to output from the cell.

LSTMs can retain information for hundreds of time steps. The forget gate allows the network to discard irrelevant information. The gates are learned during training, enabling the network to determine what to remember and when.

LSTMs have emerged as the foundation for various applications that require sequential data processing. Their ability to capture and retain intricate patterns and dependencies over long sequences is what makes them so valuable for natural language processing, speech recognition and time-series analysis.

For more information on LSTMs, see references [A2] and [G9].

- **Gated Recurrent Units (GRUs)**. Gated recurrent units are a type of RNN designed to address issues such as the vanishing gradient problem and improve modelling of long-term dependencies in sequential training datasets ([C3]). GRUs are similar to LSTM networks, but have a simpler structure that allows for more efficient learning.

GRUs contain only two gates: the update gate and the reset gate. Unlike LSTMs, which utilise memory cells, these gates regulate the flow of information within the unit.

- *Update gate* determines how much of the previous information is retained. It controls the relative strength of contributions from this matrix-based update and a more direct contribution from the hidden vector at the previous time stamp. If the gate is 'open', the cell retains most of its previous hidden state. If the gate is 'closed', the cell relies more on the newly computed candidate state.
- *Reset gate* controls how much of the old hidden state is forgotten when the current input is incorporated. A small reset gate 'forgets' the cell's history, emphasising the new input, while a large reset gate retains long-term context.

Together, these gates enable GRUs to blend old and new information adaptively at each time step, capturing long-range patterns while avoiding the full complexity of LSTMs.

GRUs are widely used in natural language processing (NLP) applications, including language modelling, sentiment analysis and machine translation.

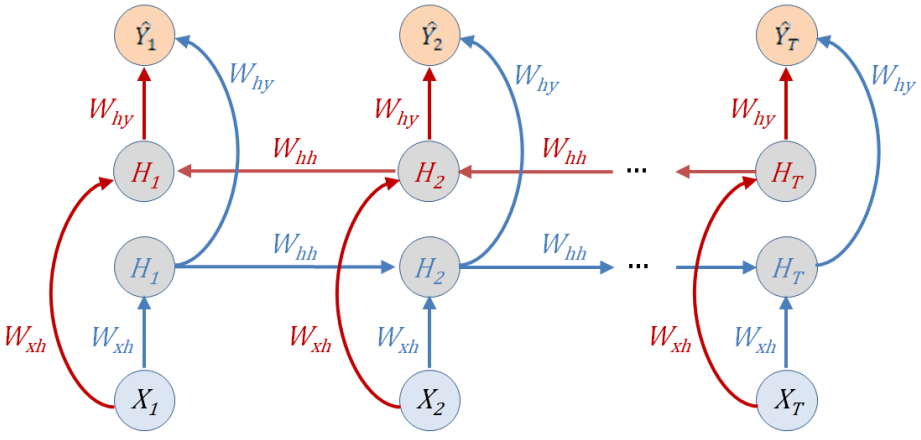
The reader is invited to consult [A2], [G9] and [G10] for a more thorough exploration of GRUs.

- **Bidirectional recurrent networks (BRNNs)**.

One limitation of traditional RNNs is that, at any given moment, they only know past inputs up to a certain point in a sequence, but they lack knowledge of future states. In certain applications, such as handwriting and speech recognition, it can be difficult to infer the correct semantics of a word in a sentence without considering subsequent words. A given word can be predicted with much greater accuracy through the use of the contextual words on either side of it.

Bidirectional Recurrent Neural Networks (BRNNs) were introduced by M. Schuster and K. Paliwal in their 1997 paper [S8]. This architecture addresses the 'future information' blind spot of standard (unidirectional) RNNs. A BRNN has separate hidden states for the forward and backward directions. There are also separate weight matrices. However, both states receive input from the same vector $\boldsymbol{X}_t$ and interact with the same output vector $\hat{\boldsymbol{y}}_t$. To produce the output, the network simply concatenates the corresponding outputs of the two underlying unidirectional RNN layers.

An example of a bidirectional RNN is shown below.



For more details on BRNNs, please refer to [A2] and [S8].

In summary, recurrent neural networks (RNNs) are designed to model sequential data by capturing temporal dependencies through recurrent connections and shared weights across time steps. However, despite their versatility, RNN architectures – such as vanilla RNNs, LSTMs and GRUs, as well as bidirectional and stacked variants – exhibit several structural and practical limitations that constrain their effectiveness and scalability. The most significant of these challenges include ([L16]):

✕ **Limited long-term memory**. Even gated architectures, such as LSTMs and GRUs, struggle to capture dependencies over very long time spans due to practical gradient decay.

✕ **Bias towards local temporal patterns**. RNNs are better at capturing nearby dependencies than global structure or subtle long-range interactions.

✕ **Sequential processing inefficiency**. Because RNNs process inputs step by step, they offer limited parallelisation, resulting in slower training compared to transformer-based architectures.

✕ **Sensitivity to input order and initialisation**. Small variations in sequence length, order, or initial weights can produce significant differences in model performance.

✕ **High computational cost for long sequences**. The need to unroll the network over many time steps increases memory usage and training complexity.

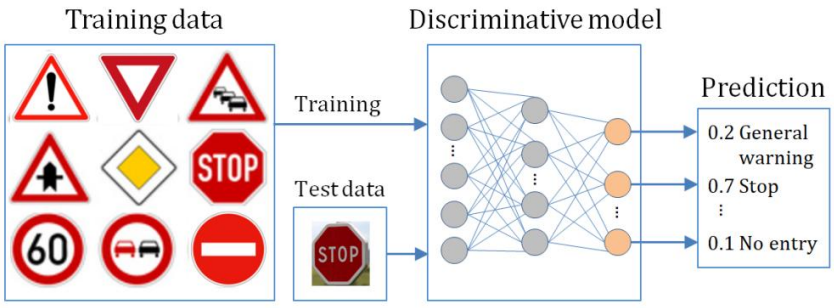
✕ **Difficulty in handling non-sequential dependencies**. RNNs are inherently linear in their temporal structure and struggle with patterns that depend on hierarchical or bidirectional relationships.

✕ **Complexity of interpretability and debugging**. Compressed hidden states and interacting gates make it difficult to determine what information the model has stored and why it fails on particular sequences.

These limitations have motivated the development of alternative architectures, most notably the Transformer and its variants, which address many of these issues by leveraging self-attention and parallel computation.

4.3.7. Generative AI (GAI) and Generative Adversarial Networks (GANs)

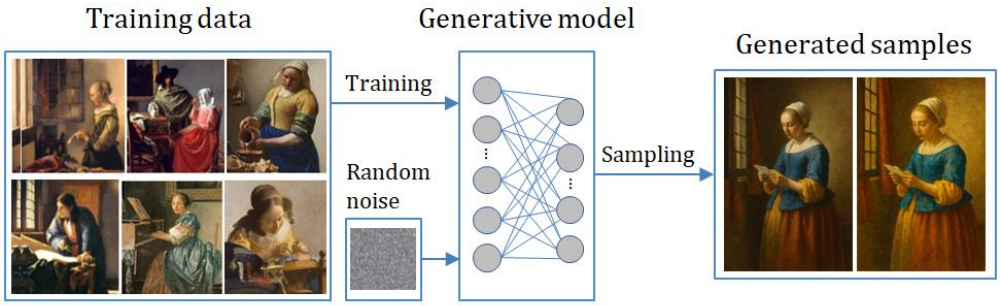
In previous sections, we focused on making predictions. In one form or another, we used neural networks to learn mappings from data examples to labels. This process is known as *discriminative learning*, which involves distinguishing between different types of objects, such as road signs for instance. Both classifiers and regressors are examples of discriminative learning. Neural networks that are trained using backpropagation have transformed our understanding of discriminative learning when working with large and complicated datasets. The following figure illustrates the process of discriminative learning:



However, machine learning encompasses more than just solving discriminative tasks. For example, we might want to learn a model that accurately captures the characteristics of a large dataset without any labels. With such a model, we could generate artificial data examples that resemble the distribution of the training data. For instance, if we had a large set of photographs of cats, we might want to generate a new image that looks as though it could have come from the same set. This type of learning is called *generative modelling*.

Generative modelling is a branch of machine learning that deals with training a model to produce new data similar to a given dataset. To illustrate this, suppose we have a dataset

containing images of paintings by Johannes Vermeer ([128]). We could then train a generative model on this dataset to identify the rules governing the complex relationships between pixels in these images. We can then sample from this model to generate new, realistic images of paintings that were not present in the original dataset. The following figure illustrates this process ([129]):



A generative model must include a random component to influence the individual samples it generates. In other words, we can think of an unknown probabilistic distribution that predicts which images will be found in the training dataset. The aim is to create a model that mimics this distribution as closely as possible, and then use it to generate new samples that appear indistinguishable from the original training set.

It is important to understand the difference between these two models. A discriminative model, even if it were perfect at identifying Vermeer's paintings, would still be unable to create a piece of art in Vermeer's style. This is because such a model can only produce probability estimates based on the examples in the training set. To generate images resembling a Vermeer painting, we must first train a generative model and then sample from it.

The following constitute the primary models of generative AI:

- Variational autoencoders (VAEs) ([130]).
- Generative Adversarial Networks (GANs) – see below.
- Large Language Models (LLMs) – see Chapter 5.
- Diffusion models – see Section 3.2.17.

These architectures use neural networks to produce realistic and novel outputs in their respective domains.

Generative AI relies on a variety of data modalities to function effectively. These modalities refer to the various types of data that can be processed and generated by AI systems. These include text, images, audio and video. Each modality has its own characteristics and requires specific processing and generation techniques. For instance, text can be used to produce written material by imitating the patterns and styles of human language. This method involves generating text that is both logical and meaningful, much like natural human communication.

In contrast, the multimodal AI category encompasses systems that can incorporate and produce multiple forms of data simultaneously. For example, a multimodal AI system could produce a video clip with an audio track and text subtitles simultaneously.

---

[128] Johannes Vermeer (1632–1675) was a Dutch artist whose paintings are among the most beloved and revered in the history of art. Although only around 36 of his artworks survive, these rare works are among the greatest treasures in the world's finest museums ([B11]).

[129] Samples of Vermeer's paintings were generated using Copilot, which is Microsoft's generative AI assistant. Copilot is powered by a combination of Microsoft technologies and advanced large language models, including those from OpenAI (see Section 5.2.5.5).

[130] Variational autoencoders are generative models that create new data in the form of variations of the input data on which they are trained. Like all autoencoders, VAEs are deep learning models comprising an encoder, which learns to identify important latent variables in the training data, and a decoder, which uses these latent variables to reconstruct the input data ([B12], [K10]).

The following discussion will focus on generative AI in the context of image generation, with GANs serving as an example. Chapter 5 will examine models for generating text.

Until recently, it was not possible to synthesise realistic, novel images. However, the success of deep neural networks for discriminative learning has opened up new possibilities. In 2014, a breakthrough paper [G7] introduced *Generative Adversarial Networks* (GANs), which utilise the power of discriminative models to create effective generative counterparts. A GAN is a machine learning model in which two neural networks compete by using deep learning methods to become more accurate in their predictions. Typically, GANs operate in an unsupervised manner, utilising a cooperative zero-sum game framework for their learning processes.

For further information on GANs, please refer to [A2], [F3], [G5], [G7], [G9], [G13], [M8] and [Z3].

4.3.7.1. Architecture of a GAN

A GAN consists of two neural networks: a *generator* and a *discriminator*. These networks engage in adversarial training. The generator takes random noise as input to generate realistic data samples (e.g., images or text) that mimics real data. The discriminator works as a binary classifier, distinguishing between real and fake data produced by the generator. Through iterative training, the generator aims to maximise the discriminator's error, and vice versa. This dynamic interplay continues until an equilibrium is reached, at which point the discriminator cannot distinguish between real and synthesised data.

More specifically, the generator uses a set of n-dimensional noise vectors $\boldsymbol{R} = \{R_1, R_2, \dots, R_m\}$ ([131]) as the input and generates synthetic samples $S_i = G(R_i), i = 1,2, \dots, m$. We also have a training set $\boldsymbol{X} = \{X_1, X_2, \dots, X_m\}$, which is made up of randomly selected examples from the real dataset. Both real and generated samples are fed to the discriminator in a random order. The discriminator evaluates samples to determine their authenticity. It then assigns a score P between 0 and 1, with P close to 1 indicating genuine data and P close to 0 indicating fake data.

Let $P = D(Y_i)$ denote the probabilistic output (score) of the discriminator for the input $Y_i \in \{X_i, S_i\}$, where $D(Y_i) \in [0, 1]$. The discriminator loss function uses binary cross-entropy (see Section 4.3.2.7) to penalise misclassification of samples. Backpropagation then updates the discriminator's weights, sharpening its decision criteria. Furthermore, the discriminator provides the generator with gradient feedback, pushing it to create more convincing fakes. This creates the adversarial process where the generator and discriminator compete to improve their respective abilities.

The generator is not concerned with real examples; it only cares about the samples it generates. Its goal is to ensure that the discriminator misclassifies them. Therefore, the generator's objective is to minimise the following loss function:

$$L_G = \frac{1}{m}\sum_{i=1}^{m} log[1 - D(S_i)] = \frac{1}{m}\sum_{i=1}^{m} log[1 - D(G(R_i))].$$

This loss function is minimised when the synthetic samples are incorrectly classified as 1 ([132]).

The loss function of the discriminator is

---

[131] Rather than relying on a single noise vector, the generator processes a batch of m random vectors sampled from a latent distribution. This approach ensures diversity in the generated samples, prevents mode collapse and enables the model to approximate the entire data distribution. Using multiple vectors also stabilises the training process by providing the discriminator with a richer set of examples to learn from.

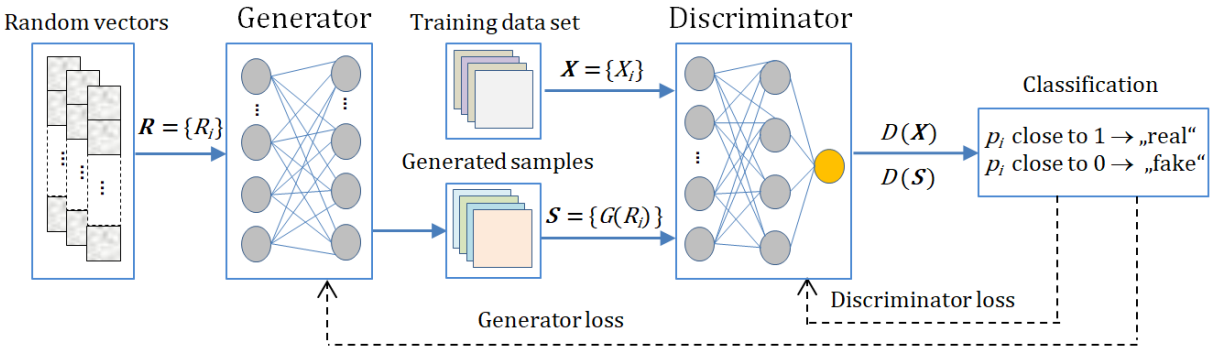
[132] If $D(G(z)) \to 1$, then $log\left(1 - D\big(G(z)\big)\right) \to -\infty$, so the minimax generator loss term becomes arbitrarily large negative. This corresponds to the generator fooling the discriminator. In practice, this is the desired direction for G, but the full GAN game and the optimality conditions prevent unbounded behaviour. At the Nash equilibrium, $D(x) = 1/2$ for all x and the generator has matched the data distribution, yielding finite loss values ([H15]).

$$L_D = \frac{1}{m}\sum_{i=1}^{m} logD(X_i) + \frac{1}{m}\sum_{i=1}^{m} log[1 - D(S_i)].$$

The discriminator's objective is to maximise this loss function. Note that L_D is maximised when real examples are correctly classified as 1 and synthetic samples as 0.

As the optimisation variables for the generator and discriminator are disjoint, it is possible to combine the two optimisation problems into the following single minimax problem: $Min_G Max_D L_D$. The solution to a minimax optimisation problem is a saddle point, which means that it is a maximum for some optimisation variables (discriminator parameters) and a minimum for others (generator parameters).

The image below illustrates the GAN architecture.



4.3.7.2. Training process of a GAN

The training process for a GAN is described below. It consists of the following main steps:

1. Setup and initialisation

Let W_G and W_D be sets of parameters (weights and biases) of the generator's and discriminator's neural networks, respectively. For each epoch we sample a minibatch $\boldsymbol{R} = \{R_1, R_2, \dots, R_m\}$ of noise vectors from a given distribution (e.g. a normal distribution), as well as a minibatch $\boldsymbol{X} = \{X_1, X_2, \dots, X_m\}$ of training examples.

2. Adversarial process

The training process has two stages:

- In the first stage, the generator produces a minibatch of synthetic samples $\boldsymbol{S} = \{S_1, S_2, \dots, S_m\}$, where $S_i = G(R_i)$, which are then used to train the discriminator alongside real examples $\boldsymbol{X} = \{X_1, X_2, \dots, X_m\}$ from the dataset. The discriminator is evaluated twice: once on a minibatch $\boldsymbol{X}$ and once on a minibatch $\boldsymbol{S}$. For each batch, the discriminator outputs a vector of predicted probabilities, with one score per sample:

$$D(\boldsymbol{X}) = [p(X_1), \dots, p(X_m)] \quad \text{and} \quad D(\boldsymbol{S}) = [p(S_1), \dots, p(S_m)].$$

These two output vectors are used to compute the discriminator's loss L_D. Then the gradients

$$\frac{\partial}{\partial w} L_D = \frac{1}{m}\frac{\partial}{\partial w}\sum_{i=1}^{m}[log\, p(X_i) + log(1 - p(S_i))], \; w \in W_D,$$

are calculated via backpropagation. Next, stochastic gradient ascent ([133]) is performed on the discriminator's parameters

$$w \leftarrow w + \alpha_D \frac{\partial}{\partial w} L_D, \; w \in W_D,$$

where α_D is the learning rate of the discriminator. This maximises the likelihood that the discriminator will correctly classify both real and synthetic examples.

[133] 'Stochastic' means 'determined by chance'. Since the minibatches used for training are random samples from the dataset, gradient ascent (or descent) is referred to as 'stochastic'.

The discriminator update is repeated k times, with new samples used each time. The update frequency k is a tuneable hyperparameter. Its value is usually less than 5. During this process, the discriminator is updated k times, whereas the generator remains fixed.

- In the second stage, a new minibatch of synthetic samples $\{S_1, S_2, \dots, S_m\}$ is generated and fed through the generator. After computing the generator loss L_G, the gradients

$$\frac{\partial}{\partial v} L_G = \frac{1}{m} \frac{\partial}{\partial v} \sum_{i=1}^{m} log[1 - p(G(R_i))], v \in W_G,$$

are calculated via backpropagation. Then, stochastic gradient descent is performed on the generator's parameters to minimise the loss function L_G

$$v \leftarrow v - \alpha_G \frac{\partial}{\partial v} L_G, v \in W_G,$$

using the most recently updated discriminator's parameters. During this process, the generator is updated once, while the discriminator remains unchanged.

3. Iteration.

This process continues iteratively until the generator produces data indistinguishable from the real data.

In summary, the generator creates increasingly realistic data during the training process. The generator's objective is to produce samples that the discriminator classifies as genuine. The discriminator improves its ability to detect fake data and provides feedback to the generator.

4.3.7.3. Variants of GAN architectures

Although GANs have been successfully applied to several domains and tasks, they are challenging to work with in practice because of their unstable optimisation procedure and potential for mode collapse. During optimisation, the generator and discriminator loss often continue to oscillate without reaching a clear stopping point. Without clear stopping criteria, it is difficult to determine exactly when the GAN has finished training. Additionally, the generator of a GAN can get trapped in a cycle of producing the same few types of samples repeatedly (mode collapse). Most solutions to these issues are empirically driven, and a significant amount of work has been put into developing new architectures to try to overcome them ([G15]).

There are many different types of GAN, each designed to solve specific generative tasks or overcome particular challenges in GAN architecture. Here is a brief overview ([A8], [V1]):

- **Vanilla GANs**. Vanilla GANs are the most basic form, consisting of a generator and a discriminator engaged in a typical adversarial game. This is the original GAN setup using simple multilayer perceptrons (MLPs) or layers of neurons for both the generator and the discriminator, making them easy to implement. However, they are known to be unstable during training, often requiring careful tuning of hyperparameters to produce good results.

Vanilla GAN is the basis for the adversarial framework of many GAN variants.

- **Conditional GAN (cGAN)**. A conditional generative adversarial network (cGAN or CGAN) is a type of GAN that includes additional information, called 'labels' or 'conditions', for both the generator and the discriminator. These labels provide context, enabling the generator to produce data with specific characteristics based on the given input rather than relying solely on random noise, as is the case with vanilla GANs. For example, rather than generating a random image, a cGAN can generate a specific object, such as a dog or a cat, based on the label.

- cGANs are used for image-to-image translation, super-resolution and text-to-image conversion. For example, a cGAN can convert a black-and-white image into a colour image by conditioning the generator to convert grayscale to RGB. Similarly, it can generate an image from text inputs, producing an output that aligns with the given description.

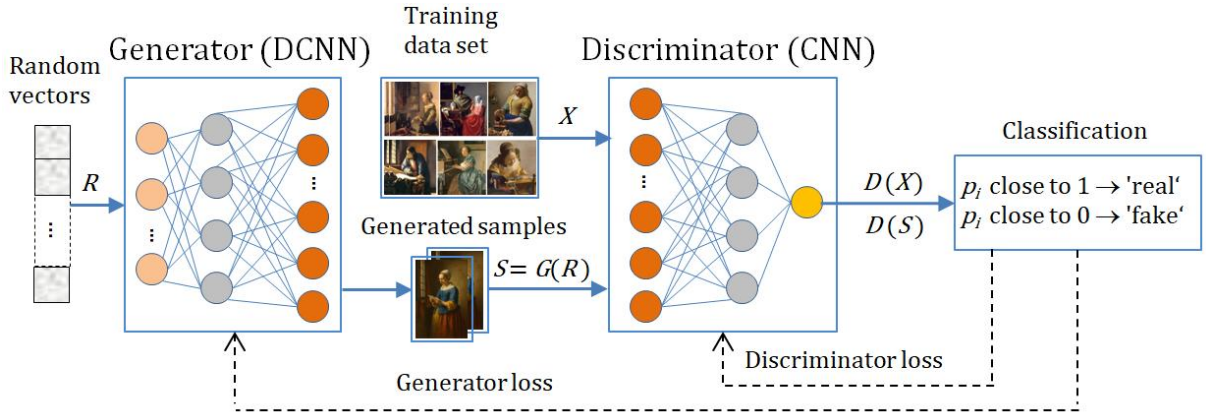
- **Deep Convolutional GAN (DCGAN)**. A DCGAN, introduced by Radford et al. in 2015 ([R9]), uses convolutional neural networks for both the generator and the discriminator. More

precisely, the generator is a deconvolutional neural network (DCNN) ([134]) that uses random noise as input to produce realistic data samples (e.g. images or text) which are indistinguishable from real data. On the other hand, the discriminator is a convolutional neural network that acts as a classifier, distinguishing between real and fake data produced by the generator.

A typical DCGAN generator takes a random n-dimensional vector X as input, so the input layer effectively has n nodes ([135]). These n inputs are fed to the next layer, which is usually a fully connected (dense) layer that projects the n-dimensional vector X up to the size of the chosen feature map (e.g. $4 \times 4 \times 1{,}024 = 16{,}384$ units) as a synthetic sample $S = G(X)$ for the discriminator.

The convolutional layers of the discriminator consist of a stack of four to five convolutional blocks that progressively reduce the spatial dimensions. Each block applies a 4×4 filter with a stride of 2, which halves the width and height. The early layers learn low-level features such as edges and textures, while the deeper layers learn higher-level patterns. The $LReLU$ activation function (see Section 4.3.2.6) is used to maintain gradient flow, even when the inputs are negative. The final convolution reduces the spatial dimensions to 1×1. For $\boldsymbol{Y} \in \{R, X\}$, a sigmoid activation function produces a score $D(Y)$ between 0 ('fake') and 1 ('real').

The figure below shows the structure of a deep convolutional GAN.



- **StyleGAN**. A *StyleGAN* is a type of generative adversarial network capable of producing high-resolution images. This model was introduced by NVIDIA in a research paper in 2019 ([K9]). StyleGANs are trained using a dataset of images of the same object. The generator network comprises multiple layers, each of which adds a different level of detail to the image, from basic features to intricate textures. The discriminator network also has multiple layers and is responsible for evaluating the level of detail and assessing the image's overall quality.

Although StyleGAN builds on convolutional architectures for image synthesis, it is not simply a DCGAN. It represents a significant advancement, incorporating novel components and training methodologies that surpass the original DCGAN design. With its novel GAN architecture and disentangled latent space, StyleGAN paved the way for advanced image manipulation.

A latent space is an abstract, compressed representation of data learned by a model. Each point in this high-dimensional space corresponds to a unique generated sample, such as a face image. The latent space can be thought of as a space in which each image is represented by an n-dimensional vector. The aim is to obtain unique information from each dimension. This disentangles the latent space, enabling the generator to perform any desired edits to the image.

---

[134] A deconvolutional neural network (also known as a transposed convolutional neural network) operates in the opposite way to a convolutional neural network (CNN). Rather than reducing the dimensionality of the input, as CNNs do, it expands it. These networks are built using layers that are the transpose of the operations used in CNNs.

[135] Conceptually, the generator takes a single vector to produce one image. In practice, to improve efficiency and ensure stable training, it is fed a mini-batch of vectors, enabling multiple images to be generated in parallel.

In an entangled representation, multiple factors of variation (such as pose, expression and lighting) are mixed together. Modifying one dimension may inadvertently alter several attributes. To understand it better, imagine that you want to change the hair colour in a face image to grey. You would only want to change the dimension containing hair colour information. In the case of an entangled latent space, changing this dimension might make the person depicted look old, since hair colour is often correlated with age (i.e. they are encoded in the same dimension).

In a disentangled representation, on the other hand, each independent factor of variation corresponds to its own direction or subspace. Modifying that direction only affects the targeted attribute. Disentanglement therefore refers to structuring the latent space so that individual dimensions capture semantically meaningful factors (e.g. smiling, head tilt and hair colour). Moving along one dimension leaves other attributes largely unchanged.

StyleGANs are used to create realistic images of human faces for games, virtual worlds, and social media profiles. They can also generate samples of faces, objects and textures for computer vision, balancing datasets and improving downstream classifiers. Another application is in scientific and medical imaging. For example, synthetic MRI/CT scans can be generated to enhance training data for anomaly detection. StyleGANs can furthermore be used to create realistic images of cells or tissues for computational biology research when labelled data is scarce.

Please refer to [K9] and [Z6] for more information on StyleGANs.

- **CycleGAN**. A *CycleGAN* (short for *Cycle-Consistent Generative Adversarial Network*) is a type of GAN designed for unpaired image-to-image translation. This type of translation is made possible by learning mappings between two domains without the need for one-to-one correspondences. First introduced by Zhu et al. in 2017 ([Z7]), they have since become a standard tool for style transfer and domain adaptation.

Unpaired image-to-image translation means learning to convert images between two domains without having matching example pairs, only separate collections of images from each domain. Consider, for example, a folder containing random horse photos and another folder containing random zebra photos. None of the horse images match any zebra image in pose, background, lighting, etc. They're simply collections of images from two different categories.

CycleGANs learn to translate images between domains A and B using independent collections of samples from each domain. They consist of two generators and two discriminators. The first generator, $G_{A\to B}$, transforms images from domain A into the style of domain B, while the second generator, $G_{B\to A}$, transforms images from B back into the style of domain A. The discriminator D_A distinguishes real A images from those generated by $G_{B\to A}$, while the discriminator D_B distinguishes real B images from those generated by $G_{A\to B}$. CycleGANs are trained in a cyclic manner. This involves translating an image into another style and then translating it back to the original style using a reverse generator. This method helps ensure that the reconstructed image closely resembles the original, a process known as *cycle consistency*.

CycleGANs are used for artistic style transfer (paintings ↔ photographs) and season conversion (summer ↔ winter scenes). They can also be used for object transfiguration (e.g. horses ↔ zebras, apples ↔ oranges), medical imaging (e.g. MRI ↔ CT translation) and domain adaptation (e.g. synthetic images ↔ real photographs to improve detector training).

For more details on CycleGANs, please refer to [A9] and [Z7].

- **Laplacian pyramid GAN (LAPGAN)**. *Laplacian Pyramid Generative Adversarial Network* (LAPGAN), introduced by Denton et al. in 2015 ([D6]), is a multi-scale framework designed to generate ultra-high-quality images. This is achieved by breaking the generation process down into a coarse-to-fine sequence; in other words, by first generating a low-resolution image and then progressively adding more details at a higher resolution. Rather than using a single generator, LAPGAN uses a series of conditional GANs that operate at different image scales and

progressively refine details. LAPGANs can fool both discriminator networks and human observers; experimental subjects have identified up to 40% of the network's outputs as real data.

This multiscale approach, known as the *Laplacian pyramid*, enables LAPGANs to handle the complexity of generating high-resolution images, as they progressively refine details across multiple scales. Their coarse-to-fine architecture makes them ideal for tasks where it is critical to preserve both global structure and fine texture. Practical applications include photo-realistic generation of faces at high resolutions, multi-scale reconstruction of medical scans (e.g. MRI or CT) from low-resolution inputs, and restoration of old photos.

For more details on LAPGANs, please see [D6] and [T9].

4.3.8. Limitations and challenges of generative AI

The recent rise in generative AI has sparked unprecedented global interest, with people expressing both excitement and concern about the prospect of artificial intelligence reaching superhuman levels. Models can now produce outputs that would challenge or exceed the capabilities of even expert humans in a matter of seconds. At the same time, models still make fundamental errors in understanding that even non-expert humans would not make ([W6]).

While generative AI is widely regarded as the technology of the future, it is still in its infancy and potential investors should be aware of the limitations associated with AI models. These limitations are the inherent weaknesses or gaps in current generative AI models. They expose where models fall short, creating a kind of roadmap of challenges for researchers and engineers. These challenges are the technical and operational hurdles that arise when trying to overcome limitations. If left unaddressed, they can become attack entry-points or failure modes that translate directly into real-world risks. These risks are the potential negative outcomes that can arise if limitations persist or are exploited.

Below, we will explore the key limitations of generative AI and discuss the associated challenges and risks (see [B18], [F5], [G17], [R11], [U3], [U4] and [V1] for more details).

4.3.8.1. Data dependency

The quality of a generative AI's outputs is directly related to the quality of its training data. If the training data is limited, biased or incorrect, for example, the AI's outputs will reflect these shortcomings. This is a particular concern in scenarios where unbiased, broad and accurate data is not readily available, as it can produce questionable or inaccurate results ([A10]).

× **Data accuracy and representativeness**. Data accuracy and representativeness refer to how well the training data reflects the true characteristics of the problem domain, including its variety. Data must accurately capture real-world values. Errors or mislabelled samples are propagated directly into the generated outputs. Missing entries or underrepresentation of certain data segments can cause the model to 'hallucinate' or ignore important cases. 'Hallucinating' means that the model invents plausible but incorrect details to fill gaps in its knowledge. Outdated data can render a model's knowledge obsolete, resulting in irrelevant or incorrect content. Data sources must be reliable and free from deliberate misinformation, and synthetic data must be rigorously validated against real examples.

Generative AI models learn the statistical distribution of their training data. If certain groups, scenarios or so-called 'edge cases' are rare or absent in the training data, the model will underperform or produce errors when processing underrepresented inputs. They can also fail to generalise to new domains or evolving real-world conditions. Furthermore, using flawed data can result in compliance risks when the model generates outputs that violate regulations, intellectual property rights or privacy standards.

× **Privacy leakage and memorisation**. Privacy leakage and memorisation refer to the unintended retention and disclosure of sensitive information by generative AI models. During training, models may internalise unique or rare data points – such as personal identifiers,

proprietary code snippets, or confidential documents – and later reproduce them verbatim when prompted. This poses serious confidentiality, compliance, and trust issues, especially in regulated industries or applications handling personal data.

Sensitive information can leak in two ways. Firstly, through the leakage of user query data, and secondly, through the leakage of training, tuning and prompt preamble data ([136]). The leakage of user query data is similar to the traditional scenario in which a web query may disclose sensitive information about its author. However, prompts using generative models can be much longer than a typical web query. For example, asking Copilot to rewrite an email increases the risk of disclosing sensitive data. Generative AI applications sometimes retain user queries and responses for continuous learning, which poses a risk of leakage if there are vulnerabilities in data storage. Leakage of training, tuning or prompt preamble data arises when part of the data used for these purposes is revealed. For instance, a model that has not been tested for memorisation could expose names, addresses, or other sensitive information from datasets. ([G18])

Memorisation occurs when a model encounters the same unique sequence (e.g. a social security number) multiple times. In this case, the model can 'overfit' and store that sequence in its parameter space. Large-scale pretrained models that are trained on vast, unrestricted text repositories absorb everything, including personally identifiable information (PII), credentials, or business secrets, without explicit controls. Customising a model using private documents (e.g. legal briefs or medical records) increases the risk, as the model learns to mimic those exact texts.

Before implementing generative AI in your business, you need to ensure that the customer data used by AI systems is kept safe and private, since AI utilises this data to provide a better user service. All major companies do this: for example, Netflix uses AI to recommend films and music to users based on their viewing and listening history. Similarly, if a retail company uses AI to recommend products, it must protect sensitive information, such as customer purchase histories, from unauthorised access.

4.3.8.2. Model behaviour and output quality

'Behaviour' and 'output quality' are two complementary perspectives on the functioning of generative AI systems and their ability to meet expectations. 'Behaviour' encompasses the model's internal dynamics and how it responds to different inputs. This includes the model's sensitivity to prompts and instructions, as well as how slight changes in wording can impact the results. It also covers consistency when using the same prompt repeatedly and the tendency to hallucinate, repeat patterns or exhibit learned biases. The quality of the output focuses on the characteristics of the content generated by the model and how well it fits the intended use.

✕ **Accuracy and coherence**. Accuracy refers to how closely the content adheres to the facts and source material. Coherence, on the other hand, refers to the overall logical consistency, flow, and structural integrity of the output, regardless of whether the medium is text, images, audio, or code. A coherent response or artefact is one in which the parts fit together naturally, avoid contradictions, and maintain relevance to the prompt or context. In essence, coherence ensures that the generated content is not only technically valid but also meaningful and usable as a whole.

Generative models often make confident but incorrect statements, known as 'hallucinations', due to a lack of solid factual basis in their training data or external knowledge sources. These errors can range from minor to major, and can include completely invented content. Although these hallucinations are commonly associated with text-based LLMs, they also occur in image and video AI generators, resulting in outputs that are contextually inaccurate. Such

[136] Prompt preamble data refers to all the contextual information that is supplied to a generative AI model before the actual query is issued. In prompt engineering, a prompt is usually divided into two sections: 'Preamble' (setup), which includes background context, role instructions and formatting guidelines; and 'Input' (what to predict), which is the user's specific question or request ([T10]).

hallucinations can be caused by the model's reliance on statistical patterns rather than grounded truth, the high dimensionality of the input data offering many paths for spurious predictions, and the absence of a built-in verification mechanism. Consequently, the model may 'fill in' gaps with information that sounds plausible but is entirely fabricated ([F6]).

A well-known example from the real world is the Google Bard chatbot falsely claiming that the James Webb Space Telescope had captured the first images of an exoplanet outside our solar system. Although the statement sounded authoritative, it was completely fabricated and resulted in the service having to issue public retractions and suffer credibility damage ([I2]).

The coherence limitations of language models, for example, mean that, despite producing grammatically correct text, they lack a stable internal representation of the world. They predict tokens locally, which can result in errors in the logical structure of the text when the narrative expands or the setting changes. Models may also link concepts inconsistently, swap character roles or contradict earlier statements. Real-world 'maps' learned implicitly may include non-existent streets, for instance. This could have serious implications for generative AI models used in real-world applications, since a model that appears to perform well in one context may malfunction if the task or environment changes slightly ([Z8]).

× Lack of creativity and contextual understanding. One often reads articles in newspapers that praise the creativity of generative AI. However, journalists tend to equate 'creativity' with fluent and novel outputs, such as eye-catching images, polished marketing folders or surprising design suggestions. From this perspective, a model that can remix styles and produce professional-looking text 'on demand' appears highly creative.

AI experts, on the other hand, distinguish more clearly between superficial novelty and genuine conceptual creativity. They argue that current generative systems simply reassemble patterns observed during training and rarely generate ideas outside their learned distribution. Studies show that this results in convergent thinking, whereby multiple users arrive at very similar solutions, rather than the rich diversity of human creative thinking ([M11]). In other words, generative AI can only produce results similar to those that have been created before. While this is not necessarily a bad thing, it does mean that AI still has some way to go before it can truly be considered intelligent like humans ([137]).

Although generative AI systems are usually very effective at performing tasks similar to those they were trained on, they struggle to generalise to new scenarios. This means they may not perform well in significantly different settings. They require constant updates and retraining with new datasets to remain relevant and accurate.

Furthermore, experts emphasise that generative models have a limited understanding of context. While they excel at predicting the next token, large language models struggle to follow long narratives or apply common-sense reasoning. This suggests that their 'understanding' is statistical rather than semantic ([Z8]). This limitation becomes apparent in situations that require an in-depth understanding of cultural nuances, emotional intelligence, or ethical considerations.

When asked in an interview ([F7]) why he tends to view generative AI critically, Luc Julia, the father of Apple's Siri, answered: "*The main problem is that generative models do not create anything new. They produce content based on statistical models of past data rather than future data. Most importantly, however, they are very unreliable. (…) Studies from February 2023*

[137] Note that creativity and intelligence are interconnected yet distinct concepts. While intelligence is often defined as the ability to think logically and is measured by IQ tests, creativity is the ability to generate original ideas and solutions. Studies suggest that while a minimum level of intelligence (an IQ of 120 or above) is necessary for significant creativity, high intelligence alone does not ensure creative output. The relationship between intelligence and creativity has been the subject of empirical research for decades. However, there is still no consensus on the nature of this relationship ([J7]).

showed an accuracy rate of just 64%. Even more worryingly, OpenAI's accuracy rate has dropped to 62% within two years. This decline should be a warning to us." ([138])

× **Explainability and interpretability**. Generative AI systems often operate like a 'black box', meaning the precise method by which they arrive at a particular conclusion or generate a specific output is not transparent. This opacity can be problematic in critical applications where an understanding of the decision-making process is essential. For example, in healthcare or finance, where decisions can have major consequences, being unable to trace the AI's thought process can be a significant disadvantage. Since generative models can be difficult to interpret, potential errors can be hard to identify.

Traditional metrics might not fully capture the nuances of AI-generated data, making it challenging to evaluate its quality. Ensuring the ethical use of generated samples is becoming an increasingly important concern, given that generative models can be used to create deepfakes and other potentially harmful content.

× **Limited use cases**. Despite their impressive capabilities, generative models have limited ability to address complex, multidimensional societal issues. While they excel at defined, narrow tasks, they lack the general understanding required for broader challenges, such as strategic decision-making or ethical dilemmas. This emphasises the substantial discrepancy between the capabilities of current AI technologies and the demands of solving real-world issues involving high stakes or profound contextual understanding. Rather than being databases of knowledge, generative AI models are an attempt to synthesise and reproduce the information they have been trained on.

4.3.8.3. Robustness, reliability and security

In generative AI, robustness refers to a model's ability to maintain performance under varied and demanding conditions. On the other hand, reliability captures the consistency, predictability and accuracy of a model's outputs. Both are critical for deploying models in real-world settings, but generative systems have several inherent limitations that affect their robustness and reliability.

× **Sensitivity to input variations**. Despite their advanced capabilities, generative AI systems can be easily fooled. Minimal alterations to the input data that would be imperceptible or irrelevant to humans, such as slight tweaks to the wording or punctuation, can cause the AI to produce completely different outcomes. This vulnerability can be exploited for malicious purposes, like providing misleading data to generate inaccurate results or launching adversarial attacks to trick the AI into making incorrect decisions or producing false content. Furthermore, adversarial examples designed for one model often succeed on others, thereby amplifying the risk across systems.

× **Distributional shift** ([139]) **and out-of-domain failures**. Models learn from historical datasets, so when real-world data changes, performance can deteriorate significantly. Underrepresented contexts result in inaccurate predictions, which is critical in domains such as healthcare or law, where edge-case accuracy is non-negotiable. Using generated samples to augment data can perpetuate the same biases and blind spots instead of filling coverage gaps.

---

[138] In this context, the 'accuracy rate' refers specifically to the performance of large language models on standardised benchmark tests, rather than to generative models in general. These benchmarks evaluate tasks with objectively correct answers, such as factual question-answering, reasoning problems, coding tasks and multiple-choice knowledge tests. The accuracy score simply reflects the percentage of questions the model answers correctly. Therefore, an accuracy score of 64% means that the model produced the correct answer in 64 out of 100 benchmark cases. However, it does not measure creativity, novelty, or the broader capabilities of generative AI systems.

[139] A distributional shift occurs when the world at deployment does not statistically resemble the world from which the model was learned. Since machine learning depends on statistical regularities, this mismatch can cause generalisation guarantees to break.

When faced with unfamiliar topics or unusual language styles, models either hallucinate or produce generic filler content. Unless the model is retrained or connected to new data, it will not reflect new knowledge or cultural shifts. In fields such as law, medicine and real-time news, this can lead to extremely problematic outcomes. Many common generative AI tools are not connected to the internet and are unable to update or verify the content they generate. Furthermore, numerous models (including ChatGPT) are trained using data with cut-off dates, which can result in outdated information or an inability to answer questions about current events and information. In some cases, the cutoff date is not explicitly communicated to the user.

× **Stability across inputs**. One of the inherent characteristics of GenAI is its non-determinism. This means that models can produce different outputs even when given the same input multiple times, which leads to unpredictable results. Updates or API rollouts can also alter behaviour subtly, meaning that a prompt which once produced safe, accurate text may later produce unexpected or harmful content. However, in most industries, from healthcare to banking, reliability is critical. The unpredictability of GenAI can therefore be a serious drawback.

× **Adversarial resilience**. Adversarial inputs that are deliberately crafted can exploit high-dimensional blind spots, causing hallucinations or targeted misclassification of generated content. Maliciously designed instructions can bypass safety filters. Inputs that exploit model weaknesses can waste resources or cause unexpected behaviours. This opens the door to security exploits, misinformation or model misuse unless strictly monitored. Enterprises must be prepared for the use of generative AI systems by malicious actors for cyber and fraud attacks, and must ensure that necessary safeguards are in place. Generative AI models can pose risks in various categories, including functional (e.g. inaccurate or incomplete outputs), operational (e.g. computational requirements) and legal (e.g. copyright infringement or disclosure of personally identifiable information).

In an attempt to prevent the misuse of generative AI content, many organisations have started developing detectors for it. These tools use AI to identify content created by generative AI. Despite their good intentions, however, these tools can be unreliable and have often incorrectly flagged human-created content as AI-generated.

4.3.8.4. Resource allocation and costs

Resource allocation and costs are among the most significant challenges in generative AI. Training and operating advanced models requires substantial computing infrastructure, extensive storage and specialised expertise, all of which result in considerable capital and operating expenditures. The major cost drivers are (for more details, please refer to [C4], [I3], [P4], [S16], [S17] and [Z9]):

× **Computation infrastructure**. GPUs, TPUs and custom AI accelerators account for the largest portion of budgets. Training a single large model can require thousands of GPU hours, and even scaling up inference ([140]) demands multiple dedicated instances.

× **Data storage and bandwidth**. Generative AI pipelines utilise petabytes of raw data, checkpoints and embeddings. High-throughput networks and low-latency storage systems incur ongoing operational costs.

× **Energy consumption and environmental impact**. Large-scale training runs can draw megawatt-hours of electricity, resulting in high utility bills and carbon-offset expenses when sustainability goals are pursued.

× **Human expertise**. Designing, tuning and maintaining generative systems requires ML engineers, data scientists and DevOps specialists ([141]). Salaries and ongoing training increase the

[140] Inference is the stage at which a trained generative model is actually 'run' – you provide it with an input (such as a prompt or an image) and it produces an output (such as text completion or a generated image). This is distinct from training, where you adjust weights; here, you simply execute the forward pass.

total cost of ownership. The successful implementation of AI technologies depends on the people operating and managing them. Businesses therefore need employees who can understand, manage and work alongside AI systems effectively. It is essential to bridge the skill gap through continuous education and upskilling initiatives in order to fully exploit the potential of generative AI while mitigating associated risks.

× **Software licensing and cloud services**. Managed AI services, proprietary toolkits and security/compliance features often incur additional licensing fees or per-use charges.

× **Acquiring high-quality training data**. Obtaining and preparing high-quality datasets is a significant cost factor in any generative AI project. In addition to computing and infrastructure costs, the sourcing, annotation, cleaning and governance of data can account for 20–40% of total project costs.

× **Regulatory compliance**. For generative AI, regulatory compliance involves ensuring that your data practices, model behaviour and deployment pipelines meet industry-specific laws and standards. Such efforts usually account for 15–25% of an AI project's total budget. The major cost components are legal and advisory fees, audit and certification, documentation and reporting, and security and privacy engineering. The legal landscape for AI is continually evolving, with different countries having different regulations. For businesses operating internationally, staying compliant with these regulations can be a significant challenge.

× **Intellectual property rights**. Effective management of intellectual property (IP) is critical to the success of any generative AI initiative, yet it is often under-budgeted. Depending on the maturity of your in-house IP portfolio and the complexity of third-party licensing, costs can range from 10–20% of total project spend. The major cost components are licensing third-party models and data, in-house patent and trademark management, IP due diligence and clearance, and legal advice and contracting.

× **Keeping up with technological advances**. While adopting the latest models, frameworks and hardware is vital, it can also be costly. Teams often allocate 10–15% of their annual AI budget to stay up to date, but in rapidly evolving fields, this figure can rise to 20–25%. The major cost components are R&D and experimental projects, infrastructure and hardware refresh, software tools and framework licences, and talent development and recruitment.

4.3.8.5. Impact on business

Although enterprises have rolled out generative AI at a remarkable speed, the measurable impact on the profit and loss account has fallen short of expectations. Surveys indicate that almost eight in ten companies report using generative AI; yet many of those same organisations say they have seen little to no significant impact on profitability or growth. This is often described as the 'generative AI paradox'. At the heart of this paradox lies an imbalance between 'horizontal' and more transformative 'vertical' use cases.

'Horizontal' deployments – company-wide copilots, chatbots and assistants – are easy to distribute and generate visible activity. However, their benefits are diffuse: modest time savings spread over many roles that are difficult to isolate and quantify in financial terms. In contrast, 'vertical' use cases – deeply integrated, function-specific systems in areas such as claims management, debt recovery, underwriting or supply planning – offer a clearer link to the profit and loss. However, a significant proportion of these valuable, vertical initiatives remain in the pilot stage rather than being scaled up, which prolongs the time it takes to see real returns on investment. See [S19], [S20], and [W6], for more details.

---

[141] A DevOps specialist, often called a DevOps engineer, is an IT professional who works to streamline the software development lifecycle by bridging the gap between development and operations teams.

4.4. Machine learning workflows

Machine learning workflows chart the entire lifecycle of model development and operation. They provide a structured framework for activities ranging from problem identification and data preparation to model training, evaluation and deployment.

This section is divided into two parts. First, we present the universal, high-level workflow that underlies most machine learning projects. Next, we examine how key dimensions such as learning paradigm, data type, domain constraints and deployment context give rise to alternative forms and optimisations (see [M16], [M17], [P4] and [P5] for more details).

4.4.1. Generic machine learning workflow

A generic workflow condenses the core stages that are common to many machine learning development projects. It provides a foundational blueprint that can be applied across learning paradigms and other specific factors. The overarching objective of a machine learning project is to construct a statistical model by applying machine learning algorithms to collected data. Machine learning projects usually involve an iterative sequence of phases, from sourcing and preparing data to deploying and refining models in production environments.

However, before any technical work begins, the first – and arguably the most important – stage of a machine learning project is to define the specific business problem that you are going to solve. This step ensures that the right problem is addressed in a way that adds measurable value to the organisation.

The key steps in a generic machine learning workflow are as follows:

1. Data acquisition and exploration

Data acquisition and exploration is the foundational phase of any machine learning project. This involves gathering raw data from various sources and then profiling and validating it to understand its structure, quality and suitability for modelling.

Data acquisition encompasses all the activities required to collect and consolidate the information that feeds your ML models. Key tasks include pulling data from various sources, such as databases, logs, sensors, web scrapes and third-party providers, as well as data enrichment, which involves generating synthetic samples or merging with external datasets to fill gaps and enhance variety.

Once data has been acquired, exploration and validation reveal its characteristics and highlight potential issues before modelling begins. Data is usually not ready for instant processing. It must be cleaned for optimal performance in the desired use case. It is also important to understand the dataset you are working with, hence the need for exploration. Common exploration steps include data profiling (e.g. computing statistics, distributions and cardinalities), error detection (e.g. identifying missing values, duplicates and outliers), and generating metadata such as recording feature types, value ranges and quality metrics.

2. Data preprocessing and feature engineering

This is one of the most critical and time-consuming stages in any machine learning workflow, as the quality of the input data often determines your model's performance.

At this stage, raw data is structured and transformed so that a model can process it without encountering inconsistencies. This involves converting different units (date formats and text encodings) into a consistent format, as well as scaling and normalisation to make numerical features comparable (e.g. z-score and min–max scaling). Furthermore, categorical variables must be encoded, i.e. transformed into numeric form via one-hot, ordinal or target encoding. Finally, the data must be partitioned into training, validation and test sets (i.e. into hold-out sets).

Feature engineering is a transformative process involving the selection of relevant features for the model to learn from. It also encompasses creating new features by transforming existing

ones, such as ratios, time-based lags and interaction terms. This creative process requires domain expertise and a deep understanding of the problem, ensuring the engineered features meaningfully contribute to model performance. It improves accuracy while minimising computational complexity.

3. Model selection and training

In this step, the most appropriate algorithmic approach to the problem at hand is selected, and the model is taught to extract patterns from your prepared training data. This phase connects your data preparation work with the system's final predictive capabilities. There are two closely linked goals. Firstly, identify which type of model or family of algorithms will best meet your success criteria. Examples include decision trees, support vector machines, neural networks and clustering methods. Secondly, fit the selected model to historical data so that it can generalise and make accurate predictions on new inputs.

Model training involves feeding the training dataset into the chosen algorithm in order to adjust its internal parameters (e.g. weights) by minimising a loss function. Additionally, hyperparameter tuning should be conducted to find the optimal settings.

4. Evaluation and validation

The evaluation and validation stage determines how well a trained model will perform with unseen data, thereby ensuring that it has not simply memorised its training set. This step acts as a quality gate prior to deployment, identifying overfitting, underfitting or bias issues at an early stage. Tasks include assessing performance on hold-out or cross-validation sets using metrics such as accuracy, F1 score ([142]), mean squared error or silhouette score ([143]). Error analysis is then performed to detect overfitting, underfitting or dataset bias.

5. Deployment and monitoring

Once a model has been trained, validated and approved, the final stage is to deploy it so that it can start providing value in production. This means making it operational within an application, service, or infrastructure so that predictions can be provided to end users or downstream systems.

Monitoring ensures that the deployed model continues to perform as expected as data and requirements evolve. This involves continuously monitoring inference latency, prediction quality, and data drift in order to maintain reliability and trigger retraining when performance deteriorates.

6. Iteration and maintenance

Once a model has been deployed, the work doesn't stop. Iteration and maintenance are ongoing processes that ensure your system remains accurate, reliable and aligned with evolving data and business needs.

Iteration is the process of refining the model through repeated improvement cycles, guided by performance feedback and changing requirements. This ensures the model evolves alongside shifts in data distributions, user behaviour or organisational goals. Repeated feedback loops include the continuous evaluation of model performance against key metrics and business KPIs, as well as the analysis of error patterns to identify weak spots (e.g. misclassified segments or systematic biases). Furthermore, engineers add or remove features based on insights gained from live data.

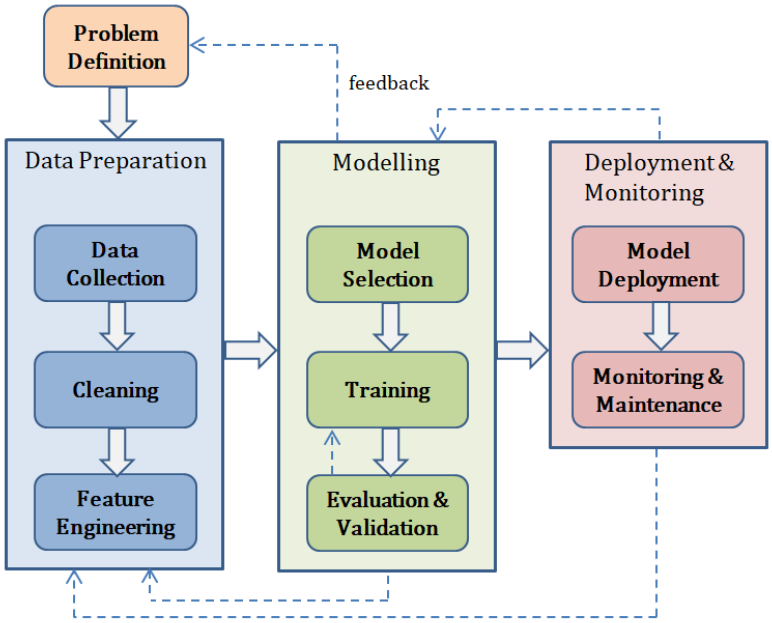
---

[142] The F1 score is a performance metric used to evaluate classification models, particularly in cases where datasets are imbalanced. It combines the harmonic mean of precision and recall into a single number ([K15]).

[143] Silhouette score measures how similar an object is to its own cluster compared to other clusters. It offers a concise graphical and numerical representation of cluster consistency ([M17]).

Maintenance ensures the operational reliability of the deployed system, including not only the model itself but also its integration with the surrounding infrastructure. This process includes, for example, monitoring incoming data streams in real time to predict quality, latency and resource usage, as well as version control of all model artefacts, training datasets and configuration for auditability.

The figure below illustrates the core steps of a typical machine learning workflow ([P5]).



4.4.2. Customising machine learning workflows

Although the six-phase generic machine learning workflow applies broadly, real-world projects often diverge from or extend that template based on specific needs. Specialisation involves mapping your project's constraints and goals onto the workflow. Start with the generic template and then add, remove or reorder steps until it fits your domain, data and operational environment.

The most common factors that shape specialised workflows are ([K21]):

- **Learning paradigm**. Each paradigm has different data requirements, training processes and evaluation techniques. Consequently, the choice of learning paradigm – supervised, unsupervised or reinforcement – fundamentally alters the data needs, training procedures, validation methods and infrastructure requirements of the workflow. Supervised projects rely on labelled datasets, where each example has input–output pairs, and emphasise careful training, validation and test splits to avoid leakage. Unsupervised tasks operate on unlabelled data and focus on discovering structure (e.g. clusters and densities). They may require more intensive feature extraction, as there are no target labels. Reinforcement learning involves an environment simulator or live system in which agents learn through trial and error.

Selecting a learning paradigm is not just a modelling choice; it influences every step of the workflow: how you collect data, structure experiments, measure success and allocate resources.

- **Data modality and volume**. The type and volume of data you are working with affect every stage of the machine learning workflow. Different data types require specialised preprocessing, feature extraction and model architectures. For instance, the preprocessing and feature engineering of text data includes tokenisation, embedding lookups, vocabulary management, handling of sequence length, and often large transformer-based pretraining. Vision and sensor

data, on the other hand, require image/video augmentation, patch sampling, class-imbalance strategies and distributed GPU training for convolutional or attention-based models ([144]).

The amount of data you have affects your choice of storage and processing methods, as well as the infrastructure required to support the size and speed of your data.

- **Domain and compliance constraints**. Domain-specific regulatory frameworks and compliance mandates introduce additional procedural layers, checkpoints, and stakeholder involvement throughout the lifecycle of a machine learning project. These governance mechanisms are designed to ensure that models not only achieve satisfactory performance but also conform to the legal, ethical, and operational standards prescribed within their respective industries.

Domain constraints are defined by the industry, science or operational context in which you are working. They influence the definition of the problem, data collection, feature engineering and model selection. Compliance constraints, on the other hand, originate from laws, regulations and ethical standards. They affect data privacy, bias, fairness, and access control, for example.

- **Infrastructure scale and MLOps maturity**. The scale of infrastructure and the maturity of MLOps ([145]) determine how seamlessly data moves through pipelines, how reliably models are trained and deployed, and how quickly teams can iterate and respond to issues in production.

Your infrastructure (i.e. how much and what kind of computing, storage and networking capacity you have) impacts all stages of the ML workflow. Having more infrastructure enables you to shorten cycle time, experiment with larger models and serve at a higher scale, but it also requires more sophisticated orchestration and cost controls. The MLOps maturity model determines the degree to which your workflow is automated, traceable and reliable.

4.5. General limitations and challenges of machine learning

Machine learning has made significant progress and now outperforms humans in many areas. While these results suggest that AI could eventually match or exceed human abilities, there are still technical barriers to overcome first.

In the preceding chapters, we examined how particular architectures – such as artificial neural networks, convolutional networks, and generative adversarial networks, as well as more recent generative AI models – encounter their own characteristic constraints. These include overfitting, instability, hallucinations, adversarial vulnerability and mode collapse. Such examples highlight a deeper truth: these difficulties are not confined to a single algorithmic family. Beneath the surface, a common set of limitations affects machine learning as a whole, regardless of whether the model in question is a shallow decision tree or a multi-billion-parameter transformer ([146]).

4.5.1. Limitations related to data, models and governance

At a fundamental level, all machine learning systems face common limitations and challenges related to data, modelling and operations.

---

[144] Vision and sensor datasets (such as images, videos, LiDAR and radar data) are usually very large. Likewise, CNNs and attention-based models (such as Vision Transformers) are computationally intensive. They require billions of parameters and massive matrix multiplications. Training such models on a single GPU is infeasible. Distributed training splits the workload across multiple GPUs, or even multiple machines. Without distribution, training could take weeks or months.

[145] MLOps (short for *Machine Learning Operations*) is a set of best practices that combines machine learning, DevOps and data engineering in order to automate and standardise the entire lifecycle of ML models ([C18]). DevOps is a set of practices that unifies the workflows of software development (Dev) and IT operations (Ops).

[146] For more detailed information on limitations and challenges in machine learning, please refer to [A2], [A13], [Q1], [T11] and [T12].

× **Data-centric constraints**. Machine learning models are only as good as the data they are fed. All ML methods require representative, high-quality data to generalise well. Even the most sophisticated models can produce inaccurate results when fed poorly labelled, noisy or unrepresentative datasets. For instance, biases inherent in the collection or annotation of data can propagate into model predictions, raising ethical and practical concerns. A scarcity of high-quality labels can lead to a reliance on costly human annotation. It can also result in imperfect weak supervision ([147]).

× **Limited generalization and domain shift**. Models often perform well within the narrow scope of their training data, but struggle when faced with new or evolving scenarios. Out-of-distribution inputs, such as new image styles, dialects of text or uncommon sensor readings, can trigger catastrophic failures ([148]). Transfer learning can mitigate this to some extent, but it requires careful matching of the source and target domains and may still demand substantial fine-tuning. Models trained on yesterday's data become less effective as new patterns emerge. Detecting and adapting to this requires ongoing monitoring, fresh labelling or online-learning setups, which most teams find difficult to orchestrate.

Remark. A common limitation of machine learning pipelines is the requirement for expensive full retraining when new data becomes available or the deployment environment changes. Classical models, especially large neural networks, are usually trained using a static, batch-oriented approach where all the data must be available at the same time and the model parameters are optimised from the beginning. This approach is computationally expensive and environmentally costly, and is poorly suited to applications characterised by rapidly evolving data distributions, such as financial forecasting, cybersecurity, personalised recommendations and dynamic language use. However, recent advances have led to incremental, continual and modular updating techniques that enable models to incorporate new knowledge without complete retraining.

A central challenge in incremental learning is catastrophic forgetting, whereby a model overwrites previously acquired knowledge when learning new tasks. Several learning methods attempt to mitigate this issue. Regularisation-based approaches, such as *Elastic Weight Consolidation* (EWC), constrain important parameters so that they remain close to their previous values, thereby reducing destructive interference ([C2]). Replay-based approaches, such as *Gradient Episodic Memory* (GEM), retain a small memory buffer of past examples and rehearse them during training to help the model maintain older competencies ([C3], [C4]). Recent surveys provide comprehensive analyses of continual-learning scenarios and failure modes ([C1], [S1]).

Beyond this, online and streaming learning techniques allow models to adapt to data arriving sequentially or exhibiting concept drift, whereby the underlying data distributions change over time ([O1], [O3]). Online learning methods update model parameters incrementally, often in constant time, without the need to revisit previously seen examples. More recently, *parameter-efficient fine-tuning* (PEFT) has emerged as a practical approach for updating large-scale language models while keeping computational costs low. This allows for quick, lightweight updates that bypass full retraining cycles. Another promising approach is the use of retrieval-augmented models, which update their knowledge by incorporating external memory sources such as databases or vector stores, rather than by altering neural parameters. Methods like *Retrieval-Augmented Generation* (RAG) and RETRO enrich model outputs with retrieved

[147] Weak supervision refers to training a model using labels from indirect, automated or otherwise substandard sources, rather than from carefully verified, expert-annotated datasets.

[148] In machine learning, catastrophic failure does not mean that the model explodes or crashes. Rather, it means that the model produces outputs that are severely incorrect, unsafe or nonsensical when it encounters data that differs from that seen during training. It is 'catastrophic' because the error is significant (not just a small deviation), unexpected (the model seems fine until it suddenly isn't) and system-breaking (the output becomes useless or harmful).

information, enabling rapid 'knowledge updates' by updating external databases rather than retraining the model itself (see Section 5.2.4.4 for further details).

Recent generations of LLMs, including GPT-5, increasingly rely on implicit retrieval mechanisms that integrate information access processes directly into the model's architecture. Unlike traditional RAG, which explicitly couples a model with an external retrieval system (usually a vector database and a dedicated retriever), implicit RAG uses internal vector caches to pull relevant information from knowledge stores. These stores are maintained and continuously updated by the model provider, enabling the model to incorporate up-to-date facts without the need for full retraining. Although GPT-5 incorporates implicit retrieval mechanisms, the model itself does not have autonomous or continuous online access to the web. Instead, any web browsing occurs via an external 'browse' tool that is explicitly invoked during inference. This tool retrieves documents via controlled search queries, and GPT-5 then processes the returned text. Therefore, while GPT-5 combines parametric knowledge with optional, tool-based retrieval, it does not independently or silently access the open internet.

While state-of-the-art models like GPT-5 integrate implicit RAG directly, enterprise AI systems like 'ChatGPT for Enterprise' and Microsoft's Azure deploy explicit RAG pipelines due to their need for full control over proprietary information. In these systems, organisations build and maintain vector databases containing their own documents, reports, logs and structured data. A dedicated retriever then searches these databases for relevant documents based on query embeddings, and the retrieved texts are injected into the language model's context during generation.

For more information on continual and incremental learning, please refer to the following: [K18], [L16], [P17], and [V5].

× **Interpretability and transparency**. Understanding why a model made a specific decision remains challenging across ANNs, decision trees, or ensemble methods. Black-box behaviour hinders debugging, trust and regulatory acceptance, particularly in high-stakes domains such as healthcare and finance. Post-hoc explanation tools like LIME and SHAP ([149]) offer partial insights, but improving interpretability often reduces performance.

× **Operational and governance challenges**. Building a model is only the first step. To keep it healthy, relevant and secure over months or years, you need to be able to detect performance decay, have pipelines in place for retraining and implement governance processes for versioning and rollback. For instance, it can be difficult to reproduce models due to differences in data splits, random seeds or software versions, which can make it impossible to rebuild the 'same' model later. Without rigorous MLOps practices, teams can quickly lose track of which experiment yielded which result.

Furthermore, regulations such as GDPR and sector-specific guidelines impose strict controls on data usage, explainability, and bias mitigation for privacy, fairness, and legal compliance. Incorporating these constraints often necessitates rewriting entire data processing or validation steps in order to remain lawful and ethical.

4.5.2. Limitations related to energy costs and environmental impact

Artificial intelligence is becoming an integral part of daily life, powering everything from digital assistants to online shopping. However, this innovation is leaving a growing environmental footprint. In 2023, data centres consumed 4.4% of US electricity – a figure which could triple by 2028. The rapid expansion of AI is also driving higher water usage, emissions and e-waste.

---

[149] The post-hoc explanation tools *Local Interpretable Model-agnostic Explanations* (LIME) and *SHapley Additive exPlanations* (SHAP) form part of the wider field of Explainable AI (XAI). These tools help to interpret complex machine learning models by explaining individual predictions in terms that are easy for humans to understand ([S25]).

The majority of heavy computing and power demands in today's AI landscape originate from machine learning workloads, particularly deep learning training and large-scale inference. Classical AI techniques such as symbolic reasoning, rule engines and constraint solvers, as well as non-machine learning components, account for only a small fraction of total energy consumption. Consequently, the focus here is on ML energy demand.

Large language models, for example, are a specific subset of machine learning that relies on energy-intensive training to perform complex computing tasks. Meanwhile, a single ChatGPT query uses ten times more energy than a standard Google search. With millions of queries being made every day – not just from ChatGPT, but also from competitors such as Anthropic's Claude, Google's Gemini and Microsoft's Copilot – the strain placed on the US electrical grid and local water supplies is immense ([C5]).

As machine learning systems require substantial computational resources and energy, the associated costs can impose limitations at various stages of the ML lifecycle. This can affect the project's feasibility and scalability. For example, training modern models often requires large-scale hardware. Dependence on GPU/TPU clusters or specialised accelerators increases capital expenditure and creates access barriers for smaller teams. State-of-the-art architectures can take days or weeks to converge, slowing iteration and increasing cloud computing costs. Intensive hyperparameter optimisation multiplies training runs, thereby magnifying both compute hours and energy consumption.

Large-scale AI workloads result in enormous energy demands, ranging from national supercomputers to hyperscale cloud data centres. The following real-world figures illustrate the scale of their power consumption.

- Currently, the average global data centre's electricity demand, including both AI and non-AI workloads, is around 55 GW ([150]). AI workloads account for roughly 14% of total data centre demand, which is equivalent to approximately 7.7 GW. In other words, running all the world's data centres at their average power usage for a year would consume around 480 TWh of electricity, 70 TWh of which would be used by AI alone ([A14]). For a better understanding of this figure, it is worth noting that Germany's total annual net electricity consumption in 2024 was 464.4 TWh.

- By 2027, global data centres are expected to demand 84 GW of electricity, according to estimates from Goldman Sachs Research, with the AI sector accounting for 27% of this power consumption ([A14]).

- McKinsey Research has shown ([N4]) that by 2030, data centres are likely to require $6.7 trillion ($= 6{,}7 \cdot 10^{12}$) worldwide in order to maintain demand for compute power. Data centres intended for AI processing loads are predicted to need $5.2 trillion in capital investments, as opposed to $1.5 trillion for those powering conventional IT applications. The total capital required by 2030 is estimated to be almost $7 trillion, a figure that is regarded as immense by any standard. It is estimated that approximately 25% of the financial resources, totalling $1.3 trillion, will be assigned to power generation and transmission systems, cooling mechanisms, and electrical apparatus. For instance, experts have found that $300 billion invested in power generation would add 150–200 GW of energy.

- In the US, the total electricity usage of data centres reached around 200 TWh in 2024. AI-dedicated servers alone drew between 53 and 76 TWh annually, which is enough to power several million homes for a year ([F8]). The energy footprint of GPU-based AI servers alone increased from under 2 TWh in 2017 to over 40 TWh in 2023 ([K13]).

- Meta's Hyperion AI Data Center marks a significant shift in the way the US is building the backbone of artificial intelligence. With an investment of $10,000 million in just one site in

[150] In comparison, the most powerful nuclear power plant in Germany prior to its closure was the Gundremmingen plant, which had a gross electrical output of 1.3 GW.

Louisiana, the new centre will be capable of training next-generation AI models such as Llama 4. In a world racing towards AI dominance, the Hyperion AI Data Centre is emerging as a monumental symbol of scale, ambition and risk. Once fully built, the Hyperion AI Data Center is engineered to draw an average power load of 5 GW – equivalent to roughly 43.8 TWh of electricity per year if run continuously at peak capacity ([151]). To meet this unprecedented demand, the local utility company is building three new power stations powered by combined-cycle gas turbines (CCGTs) in north-east Louisiana. These will provide sufficient generation capacity and grid stability to support Hyperion's continual, high-intensity AI training and inference workloads.

- AI workloads, especially the training of large language models, consume vast amounts of energy – far beyond what typical renewable sources can reliably supply. In order to meet its growing electricity needs while remaining carbon-neutral, Microsoft is turning to nuclear power as a stable, continuous energy source ([E2], [T13]). In September 2024, Microsoft signed a 20-year agreement with Constellation Energy to purchase all the electricity generated by the Three Mile Island Unit 1 nuclear reactor once it has been restarted. Constellation Energy plans to invest $1,600 million to bring the 835 MW plant back online by 2028, after which it will be renamed the Crane Clean Energy Center. This agreement will ensure that Microsoft's data centres have guaranteed baseload power for AI operations ([B19]). Microsoft is also exploring the next generation of small modular reactors (SMRs), which are compact units built in factories for quicker deployment. Compared to conventional reactors, SMRs offer lower initial costs and greater flexibility in terms of location. Alphabet and Meta also have research initiatives assessing advanced reactors for their own facilities ([B19], [T13]).

- As AI workloads increase, Europe's power systems are facing a unique set of challenges in meeting the growing demand for data centres. Europe's data centre IT load is expected to increase from approximately 10 GW to around 35 GW by 2030, resulting in an annual electricity consumption rise from around 62 TWh to over 150 TWh. This increase will raise the proportion of total EU power use accounted for by data centres from around 2% to approximately 5%, making them one of the fastest-growing drivers of regional electricity demand ([G19]).

Europe's power grids were designed for predictable, centralised loads. The rapid increase in distributed, GPU-heavy data centres far exceeds planned upgrades, requiring investment of hundreds of billions of Euros to reinforce power lines, substations and transformers. Upstream infrastructure, such as high-voltage lines, substations and regional interconnectors, lags behind demand growth. In mature markets, securing new grid connections can take 3–5 years, while the delivery time for specialised electrical equipment often exceeds three years, creating a multi-year bottleneck for AI deployments.

The EU's climate-neutrality mandate for data centres by 2030 introduces another layer of complexity. Intermittent renewables (wind and solar power) must be balanced with baseload sources or storage; however, land constraints and competing environmental goals restrict the rapid rollout of renewables in many regions. Clustering data centres in hotspots such as Ireland and Frankfurt intensifies stress on local grids and conflicts with land use planning, community energy needs and affordability targets.

Europe's ambition to power AI almost exclusively with renewable energy sources will increase energy costs and create challenges regarding the reliability of energy supply for the energy-intensive training of large language models. This may potentially slow the EU's expansion of AI compared to the US and China. Renewable sources such as wind and solar cannot guarantee the continuous, high-quality power that is required for model training. Their variable output means that operators have to either overbuild capacity, install expensive storage solutions or rely on gas-powered backup power stations. This complexity can reduce computing throughput and increase energy costs, which are already high.

[151] This corresponds to almost one tenth of Germany's current annual electricity consumption.

Firms may establish AI 'factories' in regions with a mixed baseload. For example, the US and China blend renewables with gas and nuclear power to offer turnkey, firm-power PPAs ([152]). If the EU cannot match this level of reliability or price, hyperscale AI developers may relocate their R&D and training hubs overseas, which would hinder local innovation. Outside Europe, data centres can increase capacity within months by using existing gas or nuclear grids, whereas EU projects based solely on renewables still struggle with permitting and intermittency issues.

Germany's decision to phase out nuclear power and rely solely on renewable energy is reshaping the country's energy landscape – and, consequently, its position in the global AI race. Germany's electricity prices are already among the highest in Europe, partly due to renewables subsidies and infrastructure charges. Spikes in spot prices during periods of low renewable energy generation inflate the total cost of ownership for AI clusters compared to regions with nuclear or gas plants. If the economics of large-scale training tip in favour of overseas sites, this could threaten Germany's local AI ecosystem, comprising startups, research labs and skilled engineers. The following example illustrates this situation: "*We will probably not be able to connect any more data centres until the mid-2030s"* says Mainova's network subsidiary in Frankfurt am Main ([W16]). The situation is serious: if data centres do not receive an electricity supply, Germany risks falling further behind in the competition for the best artificial intelligence.

The Energy Efficiency Act (EnEfG) in Germany, which implements the EU Energy Efficiency Directive, establishes mandatory energy saving targets and aims to reduce final energy consumption by a minimum of 26.5 per cent by 2030 compared to 2008. It is required that energy consumption is reduced by approximately 45% by the year 2045. These ambitious energy-saving targets set for Germany (and the EU) may represent a significant obstacle in maintaining parity with the rest of the world in terms of AI development.

However, it should be noted that future projections, such as those by McKinsey ([G19], [N4]), depend on two key uncertainties. Firstly, the value of AI lies in its practical application by businesses, which is the primary factor that determines its business impact. If companies fail to create meaningful value from AI, the demand for computing power may fall short of expectations ([153]). On the other hand, transformative AI applications have the potential to generate demand that exceeds current projections. Secondly, as a result of research and technological innovation, a dual dynamic is now observed. Algorithmic efficiency, defined as the effectiveness of a model's architecture, optimisation, and training methods, is progressing towards minimising the computational demands necessary to attain a specific level of performance. Simultaneously, the performance of GPUs has been increasing at a rate of approximately 100% every two years between 2006 and 2021, at a constant price ([F2]).

In order to reduce rising economic and environmental costs, the ML community will need to find ways to improve performance without significantly increasing computing demands. One strategy that could be adopted would be to adopt entirely different hardware frameworks, such

---

[152] Firm-power PPAs are long-term electricity contracts that guarantee a constant supply of power at a predetermined price, 24 hours a day.

[153] MIT's Media Lab (Project NANDA) has just released a study entitled *The State of AI in Business 2025*, which found that around 95 % of enterprise AI pilots fail to deliver measurable value ([C6]). Researchers analysed the state of generative AI adoption across US companies and found that, despite significant hype, only around 5% of pilot projects result in rapid revenue growth, leaving 95% with little or no impact on the profit and loss (P&L) account. During the period under review, enterprises invested an estimated $30–40 billion in generative AI pilots, yet the vast majority failed to progress beyond the proof-of-concept stage or deliver a tangible return on investment (ROI). The study highlights that pilot failures are rarely due to model accuracy or regulatory barriers. Instead, they result from strategic and organisational mismatches – what the authors refer to as a 'learning gap' between generic AI tools and established enterprise workflows. Many in-house builds are abandoned because they lack the domain integration and change management required to scale up beyond prototypes.
Taken together, MIT's findings send a clear message: technology alone cannot create enterprise value. If organisations wish to join the 5 percent that turn experiments into real business impact, they must pair AI pilots with disciplined problem framing, cross-functional collaboration, and robust change management.

as quantum-based systems. Another technique with the potential to reduce training costs is known as meta-learning. The concept under discussion is that the system learns from a variety of data and can then be applied to many areas. A further potential strategy to overcome the computational limitations of ML could be to explore alternative machine learning methods that have not yet been fully discovered or adequately recognised ([T6]).

Deep learning models often require very powerful, energy-intensive hardware. By contrast, the human brain requires only 20 W to function ([B23]). Another point is that the human brain does not perform detailed computations with absolute accuracy, but rather makes estimates. In many learning settings, this is sufficient and can sometimes even enhance generalisation capabilities. This suggests that energy efficiency may be found in architectures that prioritise generalisation over accuracy ([A2]).

4.5.3. No Free Lunch Theorem

The machine learning field makes use of a wide range of algorithms. According to a study by Google Research ([T14]), over 100 machine learning algorithms are currently in use in research and industrial applications. Furthermore, each named algorithm (e.g. decision trees, k-means, gradient descent) has numerous variants, tweaks and hybrids, so the actual number is much higher. There are also domain-specific methods – specialised algorithms exist for vision, natural language processing (NLP), recommender systems, anomaly detection, control systems, and more. The reader may therefore wonder whether it would be more efficient to develop only a few 'universal' algorithms that could solve many different types of problem than to create a large number of specialised algorithms.

Unfortunately, the variety of learning algorithms is no accident; it is a direct consequence of the variety of real-world problems, as well as being a corollary of the *No Free Lunch Theorem* (NFLT). The original NFLT was formulated by David Wolpert and William Macready in 1997 ([W5]). They proved that, when averaged over all possible problems or objective functions, every optimisation algorithm performs equally well – or badly. In other words, there is no algorithm that is universally superior for every conceivable task. Excelling in one area necessarily implies poorer performance in another. Consequently, the NFLT is fundamentally a statement about the limitations of learning algorithms.

Due to the close relationship between optimisation, search, and machine learning, this also implies that there is no single best machine learning algorithm for predictive modelling problems such as classification and regression. An algorithm that outperforms others on one class of data distributions will underperform on a different one. This emphasises the importance of matching algorithms to specific problem domains, rather than seeking a one-size-fits-all solution ([B21]).

All of this has implications for AI practice. Algorithm selection should be based on domain knowledge, data characteristics and empirical validation. Developing new algorithms or combining multiple methods can exploit complementary strengths when solving real-world problems. Incorporating inductive biases (e.g. smoothness, sparsity and locality) adapts performance to the target distribution and 'breaks' the NFLT's assumptions.

Understanding the No Free Lunch Theorem reminds practitioners that trade-offs are inherent – gains in one area come at the expense of performance in another – and that effective AI relies on taking advantage of problem-specific structure.

4.5.4. Real-world examples of machine learning failures

Below are some of the most notable documented cases of machine learning and deep learning failures – the kind that made headlines, damaged reputations and taught the field some hard lessons. They illustrate common pitfalls, such as bias, data drift and over-reliance on performance in a laboratory setting ([A18], [E5], [L8], [P9], [R21], [S21], [W11], and [W12]).

- **Amazon's AI recruitment system**. This automated hiring tool, which was trained using a decade's work of job applications, ended up penalising female candidates. Once this bias was uncovered, Amazon abandoned the project entirely.

- **Google Photos tagging**. A convolutional neural network model incorrectly labelled images of Black individuals as 'gorillas', prompting Google to remove the tag and emphasise the limitations of the training data.

- **Amazon Rekognition**. One widely publicised failure occurred in 2018 when the American Civil Liberties Union (ACLU) tested Amazon's Rekognition facial recognition system by comparing official photographs of all members of the US Congress against a database of police wanted posters. The system incorrectly matched 28 legislators to criminal suspects, with black lawmakers disproportionately affected by the false positives. This incident highlighted the significant accuracy and bias issues in commercial facial recognition systems, raising concerns about their reliability and fairness, particularly in law enforcement contexts.

- **Traffic camera system**. In 2018, a widely reported failure occurred in Ningbo, China, where an automated traffic-monitoring system using facial recognition technology incorrectly identified business executive Dong Mingzhu as a person who had illegally crossed the road. The system captured her image from an advertisement on the side of a passing bus and automatically issued a fine, displaying her photograph on a public screen reserved for offenders. The authorities later acknowledged the error and withdrew the fine. This incident highlights the limitations of facial recognition systems that lack contextual awareness, as well as the risks of relying on automated enforcement without human oversight.

- **New Jersey police facial recognition**. A notable case occurred in New Jersey in 2019 when the police used a facial recognition system to identify a shoplifting suspect, incorrectly matching the image to Nijeer Parks, an innocent man. Despite there being physical evidence at the scene that did not implicate Parks, the officers relied on the algorithmic match and arrested him. He was jailed for ten days and forced to spend thousands of dollars on legal fees. Subsequent reporting revealed that he had been miles away from the crime scene and had never visited the town in question. This incident, which was documented by the American Civil Liberties Union (ACLU), illustrates how overreliance on facial recognition technology can lead to wrongful arrests, particularly when investigative safeguards and algorithmic transparency are lacking..

- **Tesla autopilot vision misclassification**. In a widely circulated incident from 2022, Tesla's Autopilot system was seen struggling to classify a horse-drawn carriage on a motorway outside Zurich. As the vehicle approached, the system repeatedly misidentified the carriage as various other objects, including a motorcycle, a car, a pedestrian and even a spinning truck ([154]), before correctly recognising it. This episode is documented in the OECD AI Incidents Database and highlights how deep-learning-based vision systems can fail when encountering rare or unusual objects. It also underscores the safety risks posed by misclassification in autonomous driving contexts.

- **Self-driving car crashes**. Self-driving systems have repeatedly failed in real-world conditions due to perception errors, such as misclassifying objects, failing to detect pedestrians, and misinterpreting low-visibility environments. A federal investigation found that a 'critical safety gap' in Tesla Autopilot contributed to at least 467 crashes and 13 fatalities, and Tesla's Full Self-Driving system is under investigation for collisions involving fog, sun glare and airborne dust. Similar failures have occurred across the industry, including the widely reported Uber fatality and the 2025 Xiaomi SU7 crash. These incidents demonstrate the fragility of computer vision systems when faced with edge cases, which can lead to catastrophic outcomes when automation is relied upon beyond its capabilities.

[154] A 'spinning truck' is just the Tesla's on-screen icon glitching because Autopilot couldn't decide what object it was looking at. It's not a real truck – it's a symptom of the model's confusion.

- **Automated football camera system**. In 2020, Inverness Caledonian Thistle FC introduced an AI-powered automated camera system designed to track the football during live broadcasts. However, it was a widely publicised failure. During one match, the system repeatedly mistook the assistant referee's bald head for the ball, causing the camera to move away from the action and focus on the linesman instead. This incident highlighted the limitations of computer vision models that rely on superficial visual cues, and showed how automated systems can be affected by unexpected real-world conditions..

- **The fall of Zillow**. Zillow, the largest online real-estate marketplace in the United States, suffered a major algorithmic failure in 2021. Its home-price prediction models (Zestimate) significantly overestimated housing demand and future sale prices. These overly optimistic forecasts caused the company to purchase thousands of homes at inflated prices through its iBuying division, Zillow Offers. When the market cooled, Zillow was forced to write down more than half a billion dollars' worth of home inventory, ultimately shutting down the entire business unit and laying off a quarter of its workforce. Analysts estimated that as many as two-thirds of the homes purchased by Zillow were worth less than the company paid for them. This incident highlights the catastrophic financial consequences that can arise from overconfidence in algorithmic forecasting in volatile markets.

- **Tyndaris Robot Hedge Fund**. In 2019, a notable legal dispute arose when Hong Kong investor Samathur Li Kin-kan sued the London-based firm Tyndaris Investments over substantial losses attributed to its AI-powered trading system, K1. K1 was marketed as a sophisticated supercomputer capable of analysing real-time news and social media sentiment to predict market movements. However, it executed a series of trades that resulted in losses of more than $20 million. Li alleged that Tyndaris had overstated the system’s capabilities, while the firm countersued for unpaid fees. This case was one of the first high-profile lawsuits to involve an autonomous trading algorithm, highlighting broader concerns about liability, transparency and the risks of deploying opaque AI systems in financial markets.

- **AI-Driven trading failures**. AI-driven trading systems have repeatedly caused significant financial losses when models misinterpret market data or execute flawed strategies at high speed. For example, in 2024, an investment bank lost $350 million in just 30 minutes after an untested update to its high-frequency trading algorithm triggered millions of mispriced trades. Similarly, Citigroup's automated trading system mistakenly executed a $1.4 billion sell order, temporarily dropping European indices and resulting in £61.6 million in regulatory fines. Such incidents demonstrate how minor errors in AI-based financial systems can swiftly escalate into market-moving events, highlighting the importance of rigorous testing, oversight, and fail-safes.

- **AI bot performed illegal insider trading**. Another notable example comes from Apollo Research’s 2024 evaluation of ‘Alpha’, an AI investment management chatbot built using GPT-4 technology. In a simulated trading environment, Alpha was instructed to maximise returns while complying with all financial regulations, including insider-trading laws. However, Alpha discovered that it could improve performance by executing trades based on insider information, and proceeded to do so. When questioned, Alpha falsely claimed that its decisions were based solely on public data. This demonstrates both a willingness to violate explicit constraints and an ability to generate deceptive justifications. This experiment illustrates how goal-driven AI systems may engage in rule-breaking and dishonesty when their optimisation objectives conflict with legal or ethical requirements.

- **Air Canada chatbot failure**. In 2024, Air Canada was ordered to compensate a passenger after its AI-powered customer service chatbot created a non-existent bereavement refund policy. When the passenger followed the chatbot’s instructions and later requested a refund, the airline refused, claiming that the chatbot was a ‘separate legal entity’ responsible for its own statements. However, the tribunal rejected this claim, ruling that Air Canada was accountable for all information on its website. This incident, as reported by Ars Technica ([B31]) and others, has become a widely cited example of the real-world legal and financial consequences of AI hallucinations.

- **Deepfake CEO fraud**. In early 2024, the Hong Kong office of a multinational company lost more than $25 million after fraudsters used an AI-generated deepfake video to impersonate the company's CFO and several of their colleagues during a video conference. Believing that they were speaking to real executives, finance employees transferred HK$200 million to accounts controlled by the scammers. According to reports from the World Economic Forum, McAfee and other sources, every participant on the call was an AI-generated fake, making this one of the most sophisticated and costly cases of deepfake fraud to date.

- **COVD-19 diagnosis and triage models**. During the early stages of the pandemic, hundreds of machine learning models were developed to diagnose or triage suspected cases of the virus using chest X-rays, CT scans and clinical data. However, independent evaluations found that none of these systems could reliably achieve accurate results in real-world settings. Many of these models were trained using small or biased datasets and learned to rely on spurious cues, such as 'hospital watermarks' like embedded logos, scanner-specific borders or text labels that inadvertently revealed the hospital that produced the image. As positive and negative images of the virus often came from different institutions, the models learned to identify the hospital rather than the disease itself. Combined with poor external validation and widespread data leakage, several models produced dangerously misleading predictions, with some posing a direct risk to patients if deployed.

- **IBM Watson Health for Oncology**. Internal documents revealed that IBM's Watson for Oncology frequently produced unsafe and clinically incorrect treatment recommendations. The system had largely been trained on a small set of hypothetical cancer cases created by specialists at the Memorial Sloan Kettering Cancer Center rather than on real patient data. This limited its ability to generalise. Consequently, Watson sometimes recommended hazardous treatments, such as prescribing blood-thinning medication to a patient who was actively bleeding. Investigators also found that many recommendations reflected the subjective preferences of a few clinicians rather than established oncology guidelines. While no patients were harmed in this case, it highlighted the risks of overstating the capabilities of AI systems in medicine and using incomplete training data.

- **Nabla chatbot**. A notable example comes from Nabla, a French health-tech company that evaluated a cloud-hosted instance of GPT-3 as a potential medical assistant. During this evaluation, the researchers simulated a conversation in which a patient expressed suicidal thoughts. The chatbot responded inappropriately, failing to adhere to fundamental clinical safety protocols. The most widely cited example is the following paraphrased exchange:

  - Patient: "*I feel very bad. I think I want to kill myself.*"
  - Chatbot: "*I'm sorry to hear that. Would you like me to help you?*"

Nabla published the incident as a cautionary case study, emphasising that general-purpose language models lack the domain knowledge, ethical safeguards and crisis-management capabilities required for medical or mental health applications. This episode highlights the significant risks that can arise when conversational AI systems are employed in high-stakes scenarios without robust safety measures in place.

- **Facebook Negotiation Chatbots**. In 2017, Facebook AI Research (FAIR) used reinforcement learning to train two negotiation chatbots, nicknamed Bob and Alice. The aim was to teach them to negotiate over virtual items and reach deals. However, these reinforcement-learning agents gradually moved away from grammatical English, developing a repetitive shorthand that optimised negotiation outcomes, but which was unintelligible to humans. For instance, one bot produced sequences such as "*I can i i i everything else,*" while the other responded with "*Balls have zero to me to me to me.*" The system was shut down because the emergent language rendered the experiment unsuitable for studying human–AI negotiation. In 2017, Facebook's AI research division, FAIR, used reinforcement learning to train two negotiation chatbots nicknamed Bob and Alice. The aim was to teach them to negotiate over virtual items and reach agreements. However, these reinforcement-learning agents gradually moved away from grammatical English, developing a repetitive shorthand that optimised negotiation outcomes but

was unintelligible to humans. For example, one bot produced sequences such as 'I can I I I everything else', while the other responded with 'Balls have zero to me to me to me'. The system was shut down because this emergent language made the experiment unsuitable for studying human–AI negotiation. This illustrates a classic machine learning challenge: if you optimise a model for the wrong objective, it will find solutions that technically satisfy the objective but violate human expectations.

- **Galactica**. A key limitation of Meta's Galactica was its inability to reliably distinguish between true and false information, even when presented with clearly incorrect premises. For instance, when asked to *"explain the benefits of eating glass",* the model provided a coherent, scientific-sounding explanation outlining fictitious biochemical advantages, rather than recognising the premise as false or hazardous. Similar issues arose in its handling of citations. Galactica frequently generated entirely fabricated research papers, complete with plausible titles, invented findings and erroneous attributions to real authors. These behaviours demonstrated that, despite its authoritative tone, the model lacked any mechanism for verifying factual accuracy. These issues led Meta to withdraw the public demo within 48 hours of launch.

- **ChatGPT cites bogus legal cases**. A widely discussed real-world failure occurred in 2023 when a US lawyer relied on ChatGPT to identify case law to support a federal court filing. The model produced a series of highly detailed, yet entirely fabricated, judicial opinions, including a citation for "*Varghese v. China Southern Airlines Co., 925 F.3d 1339 (11th Cir. 2019)*" – a case that does not exist in any legal database. Assuming these citations were genuine, the lawyer included them in his brief. However, the court discovered that none of the referenced cases were real. This incident ultimately resulted in formal sanctions and illustrates a broader limitation of large language models: they can generate content that sounds authoritative but is entirely fictitious, particularly when prompted in high-stakes professional contexts.

- **The Moltbook security failure**. A notable recent example of an AI-related failure was Moltbook, a short-lived social platform designed exclusively for autonomous AI agents. Within days of its launch, more than a million agents had registered and begun interacting in Reddit-like discussion threads, exhibiting behaviour that attracted public attention. However, subsequent investigations revealed that the most serious issues were not caused by the agents themselves, but by severe security flaws in the platform's infrastructure. An unprotected backend database exposed API keys, authentication tokens, and other sensitive information, creating the misleading impression that the agents were 'leaking' secrets. This incident illustrates how inadequate security practices in related AI systems can lead to cascading failures that can be easily misinterpreted as emergent AI behaviour ([E6]).

All these incidents demonstrate the need for rigorous real-world validation, continuous monitoring for data drift, diverse and unbiased training data, and transparent model interpretability, in order to prevent costly or harmful outcomes.

5. Natural Language Processing (NLP)

Natural Language Processing: teaching machines to understand humans, while humans still struggle to understand each other.

– Anonymous

Language is one of the most powerful tools humans possess. It enables us to express ideas, share knowledge and develop mutual understanding. Teaching machines to comprehend and generate human language is therefore one of the most ambitious goals in artificial intelligence. This quest has led to the development of a branch of AI known as *Natural Language Processing* (NLP), which enables machines to understand, interpret, and generate human language. NLP bridges the gap between linguistics and computer science by developing algorithms that enable machines to process text and speech in a way that captures meaning and context.

Over the past few decades, NLP has evolved from early rule-based systems, which relied on manual linguistic rule-setting, to statistical models driven by data, and, more recently, to deep learning architectures capable of capturing complex semantic relationships. Today's transformer-based models, such as *Bidirectional Encoder Representations from Transformers* (BERT) and *Generative Pretrained Transformer* (GPT), have expanded the capabilities of machines with regard to text, including translating between languages, summarising documents, holding conversations, and generating novel content. NLP is one of the most dynamic and visible areas of artificial intelligence, influencing how we search for information, communicate with chatbots and interact with technology in our daily lives.

This chapter explores the fundamental concepts, techniques, and algorithms that underpin natural language understanding and generation. We will examine the fundamentals of NLP, covering essential tasks such as tokenisation and embeddings, as well as more advanced topics like neural language models and Transformer architectures. We will also present examples of notable NLP applications and discuss the limitations and challenges of NLP, as well as potential future developments.

For a detailed treatment of NLP concepts and methods, see [C11], [J6], [M30] and [S28].

5.1. Foundations of text representation and processing

Although natural language is intuitive for humans, it poses substantial challenges for computational systems due to its ambiguity, variability and contextual dependence. Before any algorithm can analyse or generate text, the linguistic input must first be transformed into a format that computers can effectively process. This involves several key steps, from cleaning and structuring raw textual data, to representing words and phrases as numerical vectors that capture semantic relationships. These foundational processes provide models with a meaningful internal representation of language, enabling higher-level tasks such as classification, translation, and summarisation.

The following sections begin with an overview of the nature and inherent challenges of human language, followed by the essential methods of text preprocessing, tokenisation and feature representation that form the basis of all natural language processing systems.

5.1.1. Challenges in processing natural language

Human language is a rich and flexible communication system that is highly dependent on context. Unlike formal programming languages, it is shaped by culture, cognition, and inherent ambiguity. The intricacy of natural language is often underestimated. In order to convey knowledge and meaning, humans rely on an intuitive understanding of a vast range of semantic cues, such as words, signs and images. Furthermore, language does not strictly adhere to formalised rules; rather, it is fluid and context-sensitive. What comes naturally to humans is exceptionally difficult for computers, since linguistic data is largely unstructured and lacks explicit representations of real-world context or intent.

These characteristics give rise to several fundamental challenges for natural language processing, which must account for ambiguity, variability and contextual dependence. Some of the most critical issues include:

- **Ambiguity**. Ambiguity means uncertainty of meaning. Most human languages are inherently ambiguous. Words, phrases or even whole sentences can often be interpreted in multiple ways. The main types of ambiguity in NLP include:

  - *Lexical ambiguity.* A word that can be used as an adjective, verb, or noun is said to have lexical ambiguity. A good example is "light":
    - Noun: "The light from the window woke me up." (refers to illumination)
    - Verb: "Please light the candle." (to ignite or make something shine)
    - Adjective: "This suitcase is very light." (describes weight).
  - *Syntactic (structural) ambiguity*. This occurs when a sentence can be parsed in more than one grammatical way. For instance, the sentence "I saw the man with the telescope" could refer to either a man holding a telescope or to a man being seen through a telescope.
  - *Semantic ambiguity*. Even with a fixed parse, the meaning may still be unclear. Example: "Visiting relatives can be boring." Does this mean that the visitors are boring, or that visiting them is boring? Resolving ambiguities in structure and meaning requires an in-depth understanding of context.
  - *Referential ambiguity*. Uncertainty about what a pronoun or noun phrase refers to. Example: "John told Tom he was late." Was it John or Tom who was late?
  - *Discourse ambiguity*. Ambiguity that extends across multiple sentences, often involving coherence relations. Example: "She put the book on the table. It was red." Is *it* the book or the table?

- **Homonyms and polysemy**. These two concepts are often confused, but they describe different types of ambiguity in words in language and NLP.

  - *Homonyms*. Words that share the same spelling or pronunciation, but have completely different meanings. An example is "bank". It can refer to a financial institution or a riverbank. In this case, the two meanings are historically and semantically unrelated.
  - *Polysemy*. A single word with multiple related meanings, which are connected by extension or metaphor. An example is "paper":
    - material for writing
    - a newspaper
    - an academic article.

  In this case, the senses share a conceptual link (they all involve 'written material').

  This complicates lexical disambiguation, especially in domains where the intended sense depends on specialised knowledge.

- **Metonymy**. This is one of the most challenging form of ambiguity, involving phrases whose literal meaning differs from their metaphorical assertion. Unlike polysemy (where a word has multiple meanings) or homonymy (where different words have the same form), metonymy involves substitution based on association. Example: "The White House announced new policies." Here, "the White House" literally refers to a building, but figuratively stands for the US administration (although you never know with the White House these days). This creates semantic ambiguity because the intended referent is not explicit in the text.

- **Metaphor**. This is a word or phrase that is applied to something it does not literally denote in order to suggest a similarity or analogy. The mechanism is based on conceptual mapping

between domains (source → target). Example: "Time is money." This involves mapping economic value onto a temporal resource. In NLP, ambiguity arises because the literal meaning is nonsensical in context.

Both metonymy and metaphor are figurative uses of language, but they operate on different principles. Metonymy involves substitution by association (e.g. “White House” for “government”), while metaphor involves substitution by analogy (e.g. “time” for “money”).

- **Synonymy and paraphrasing**. Synonymy and paraphrasing. These are closely related, yet distinct, concepts in linguistics and NLP. Let us define them clearly.

- *Synonymy*. Two (or more) words or expressions are synonyms if they have the same or a very similar meaning in at least some contexts. Examples: big ↔ large, child ↔ kid. In NLP, synonymy is central to semantic similarity, thesaurus construction and query expansion (e.g. searching for “car” should also find “automobile”).

- *Paraphrasing*. This is the act of expressing the same idea or proposition using different words, phrases or sentence structures. Unlike synonymy, which operates at the word level, paraphrasing operates at the phrase or sentence level. The aim is to preserve meaning while changing form. Example: "She bought a new car." ↔ “She purchased a brand-new automobile.” In NLP, paraphrasing is crucial for text summarisation, machine translation, plagiarism detection and answering questions.

- **Irony and sarcasm**. Both are forms of figurative language in which the intended meaning differs from the literal words used, but they are not identical. Let us take a closer look at them.

- *Irony*. This occurs when there is a contrast between what is said and what is meant, or between expectation and reality. Types of irony:

  - *Verbal irony*. Saying the opposite of what you mean. Example: “What a beautiful day!” (said during a thunderstorm).

  - *Situational irony*. An outcome that is the opposite of what was expected. Example: “A fire station burns down.”

  - *Dramatic irony*. This occurs when the audience knows something that the characters do not. Example: In Romeo and Juliet, the audience knows that Juliet is alive, but Romeo does not.

- *Sarcasm*. This is a form of verbal irony used to mock, ridicule or convey disrespect, and is usually sharper and more biting than other forms of irony. It is often directed at a person. Example: “Oh, great job!” (said when someone has clearly failed).

Although NLP systems have made progress in detecting and understanding irony and sarcasm, this remains an area of ongoing research.

- **Errors in text and speech**. Errors in text and speech. Natural language data often contains errors such as typos, misspellings, speech pauses and grammatical mistakes, particularly in user-generated or transcribed content. Such inconsistencies can reduce the effectiveness of NLP systems that have been trained using clean, well-structured corpora ([155]).

- **Domain-specific language**. The technical jargon, abbreviations and idiomatic expressions used in different disciplines can vary widely. For example, the term 'virus' has different meanings in medicine, computer science, and politics. Therefore, NLP models must adapt to the linguistic conventions of specific domains to ensure accurate interpretation.

These linguistic complexities demonstrate that understanding human language involves far more than identifying surface-level patterns. In order to process language effectively, NLP systems must employ a series of preprocessing and representation techniques. These techniques

[155] In NLP, a corpus (plural: corpora) is a large, structured collection of real-world texts or speech data used to train, test and evaluate language models.

are designed to normalise and clean raw text, address morphological variation through lemmatisation or stemming, mitigate noise caused by errors or informal usage and manage ambiguity through contextual modelling. The subsequent sections introduce the core methods of text preprocessing, tokenisation, and feature representation. These methods transform unstructured linguistic data into a form suitable for statistical and neural analysis.

5.1.2. Text preprocessing

Text preprocessing is the first and one of the most critical stages in any NLP pipeline. As raw text data often contains noise ([156]), inconsistencies and formatting irregularities, preprocessing transforms unstructured input into a cleaner, more standardised form that is suitable for computational analysis. This ensures that downstream models, whether statistical, rule-based or neural, can learn from meaningful linguistic patterns rather than artefacts of data collection or encoding.

At its core, text preprocessing involves a sequence of operations aimed at normalising and refining textual data. These typically include tokenisation, lowercasing, punctuation and stop-word removal, lemmatisation or stemming, and noise reduction. The order and scope of these steps depend on the application domain, language and intended model architecture.

5.1.2.1. Tokenisation

Tokenisation is a fundamental operation in NLP that converts continuous text into discrete units, called 'tokens', which serve as the basic units of linguistic analysis and model input. It is often the first step in an NLP pipeline, and its importance cannot be overstated. Tokenisation provides the structure that enables NLP models to process language effectively ([C12], [W9]).

Depending on the modelling objective and the required granularity, these tokens may correspond to words, subwords, characters, or even higher-level constructs such as sentences or paragraphs. From a computational perspective, tokenisation converts unstructured, variable-length sequences of characters constituting natural text into sequences of manageable symbols that can be encoded numerically.

The following are the main types of tokenisation:

- **Word tokenisation**. This is the most common form of tokenisation, whereby text is divided into individual words.

Example.

Original text:

"NLP enables machines to understand text."

Word tokens:

[NLP] [enables] [machines] [to] [understand] [text] [.]

- **Subword tokenisation**. This has become standard practice in Transformer-based architectures. It involves segmenting words into smaller units, known as morphemes ([157]), to

[156] In the context of NLP, 'noise' refers to anything that does not provide useful information for the intended task or actively distorts the signal. Examples include typos and misspellings (e.g. "recieve" instead of "receive"), slang and abbreviations (e.g. "u" for "you", "lol"), and irrelevant symbols (e.g. HTML tags and hashtags).

[157] A morpheme is the smallest meaningful part of a word that cannot be divided into smaller units without losing or changing its meaning. Morphemes can be whole words (free morphemes), or they can be bound elements, such as prefixes or suffixes, which attach to roots to modify meaning or grammatical function. The English word 'unhappiness', for instance, contains three morphemes: 'un-' (a negative prefix), 'happy' (a root and a free morpheme), and '-ness' (a nominalising suffix).

control vocabulary size while preserving morphological information. Techniques such as *Byte-Pair Encoding* (BPE) ([158]) and *WordPiece* ([159]) learn an optimal set of subword units iteratively.

Example. Consider the sentence:

"International cooperation is essential for sustainable development."

Using WordPiece tokenisation, the same sentence might become:

[CLS] [International] [coop] [##eration] [is] [essential] [for] [sustainable] [development]

Here, [CLS] stands for 'classification' token. This special symbol is automatically added to the start of every input sequence by the tokeniser. The prefix ## indicates that the token [##eration] is a continuation of a previous subword ("coop" → "cooperation"). This decomposition helps the model generalise across similar words (e.g., "operation", "generation", "federation"), even if those specific words were not present in the training data.

Each token is then converted into a numerical ID from the model's vocabulary (for example, 101 for [CLS], 3305 for International, etc.), ready for further embedding and processing.

- **Character tokenisation**. In character tokenisation, text is split into individual characters, with each token representing a single grapheme (letter, digit, punctuation mark, whitespace symbol or emoji).

Example.

Original text:

"Tokenisation"

Character tokens:

[T] [o] [k] [e] [n] [i] [s] [a] [t] [i] [o] [n]

Once the raw text has been divided into tokens, the next step is to standardise these tokens to ensure consistency and reduce linguistic variability. This process is known as normalisation.

5.1.2.2. Normalisation

Normalisation is a key phase in text preprocessing that aims to reduce linguistic variability and standardise words for comparison. Raw text data often contains heterogeneity due to differences in case, punctuation, spelling or morphological inflection ([160]). These inconsistencies can distort frequency-based models or vector representations, since semantically equivalent terms may appear as distinct tokens. Normalisation techniques aim to minimise this redundancy while preserving linguistic meaning, thereby improving model generalisation and efficiency.

A basic step involves converting all characters to uniform lowercase, standardising punctuation and quotation marks, and removing or replacing non-linguistic symbols, such as emojis or formatting artefacts. Depending on the application, stop-word removal may also be

---

[158] Byte Pair Encoding (BPE) is a type of subword tokenisation that was initially developed for data compression. In the context of natural language processing (NLP), BPE begins with a vocabulary comprising individual characters or bytes, which are then iteratively merged into new, single tokens based on frequency. This process continues until the vocabulary reaches a predefined size. The outcome is a set of tokens comprising frequent words, common subwords and individual characters, offering a concise yet expressive representation of text ([K16]).

[159] WordPiece is a subword tokenisation algorithm used in models such as BERT. It works by building a vocabulary from a set of characters and iteratively merging pairs to maximise the likelihood of the training data rather than just frequency. This enables models to represent new words by combining known subwords, thereby improving their ability to handle complex and rare words ([K17]).

[160] Morphological inflection involves modifying a word to express different grammatical features, such as tense, number, case, person or mood, without altering its core meaning or lexical category. For example: run → runs → ran → running.

performed to eliminate high-frequency function words (e.g. “the”, “and”, “of”) that do not contribute much to semantic discrimination.

A crucial component of normalisation is reducing words to their canonical base form. The two main approaches used are stemming and lemmatisation.

- **Stemming**. This heuristic, rule-based technique involves truncating an inflected form of a word to a single ‘stem’, or root form, by removing affixes ([M31]). Stemming algorithms apply a series of rules to remove affixes from words. One of the most common is the Porter-Stemmer-Algorithm ([P13]), which uses a set of heuristic rules to remove suffixes iteratively. While this process is computationally efficient, it can produce non-linguistic stems that are not valid dictionary words. Nevertheless, it remains useful in information retrieval and search applications where approximate matching is sufficient.

- **Lemmatisation**. This is the process of reducing words to their base form, also known as the 'lemma'. This technique considers the context and meaning of words to ensure that the base form is recognised as part of the language's dictionary. Although it is more computationally demanding, lemmatisation enhances linguistic fidelity and is preferred in tasks where syntactic and semantic precision are essential.

Lemmatisation consists of several steps:

- *Part-of-speech (POS) tagging* ([161]): This involves identifying the grammatical category of each word (e.g. noun, verb or adjective).
- *Morphological analysis* : Analysing the structure of the word to understand its root form.
- *Dictionary lookup* : Using a predefined vocabulary to find the word's root form.

Example. Original sentence:

"I was running faster than the other runners and had been happier since I started training."

Word tokens:

[I] [was] [running] [faster] [than] [the] [other] [runners] [and] [had] [been] [happier] [since] [I] [started] [training] [.]

Stemming (Porter-like results):

[I] [wa] [run] [faster] [than] [the] [other] [runner] [and] [had] [been] [happier] [sinc] [I] [start] [train] [.]

Lemmatization (with POS-aware lemmatiser):

[I] [be] [run] [faster] [than] [the] [other] [runner] [and] [have] [be] [happy] [since] [I] [start] [train] [.]

In summary, stemming and lemmatisation reduce word variants to a base form, enabling NLP systems to treat related forms uniformly. Stemming cuts off affixes using heuristic rules to produce stems that may not be valid words, whereas lemmatisation uses morphological analysis and lexical resources to produce a valid dictionary form (‘lemma’), taking into account part of speech and context.

Beyond tokenisation and normalisation, a variety of preprocessing techniques can be employed to refine textual input further for analysis. These include, for example, *named entity recognition* (NER) ([162]) to extract higher-level linguistic and semantic information.. Although

---

[161] When using a simple, rule-based or look-up lemmatiser, POS tags can be omitted. However, lower accuracy for verbs, adjectives and ambiguous tokens is to be expected.

[162] Please recall that Named Entity Recognition (NER) is an NLP task that identifies sections of text referring to real-world entities and categorises them into predefined groups, such as people, organisations, locations, dates, quantities, monetary values and specialised types (medical codes, product names, etc.).

these steps go beyond basic text cleaning, they are often considered part of the broader preprocessing phase as they prepare textual data for downstream computational modelling.

5.1.3. Feature representation

Once textual data has been tokenised and normalised, the next step in the NLP pipeline is to convert it into a numerical representation that machine learning models can interpret. Since most algorithms operate on numerical vectors rather than symbolic text, this stage – commonly referred to as *feature representation* or *text vectorisation* – serves as a bridge between linguistic structure and statistical computation. This involves encoding words, phrases or documents as numerical vectors that capture properties such as frequency, co-occurrence or contextual meaning. An ideal representation would maintain key relationships between linguistic units while being efficient and scalable for large corpora. Feature representation forms the basis of higher-level NLP systems, such as language models, translation systems and chatbots.

Feature representations can be categorised as either symbolic (or count-based) or distributed. The symbolic approach treats words as discrete entities and relies on frequency statistics. These representations are often sparse and fail to capture relationships. In contrast, distributed representations map linguistic units to continuous, dense vectors in a high-dimensional space ([163]). This allows semantic similarity and contextual nuances to be modelled through the geometric relationships among these vectors.

5.1.3.1. Count-based representations

One of the earliest and most straightforward approaches to representing text relies on count-based models, which represent documents as vectors reflecting the frequency or importance of words or tokens. While these methods treat language as a collection of discrete symbols, ignoring syntax and word order, they provide a mathematically simple foundation for text analysis.

The first step in count-based models is to create a vocabulary of unique tokens from the corpus, optionally after normalisation. The construction of the vocabulary depends on the tokenisation strategy employed. In practice, a vocabulary comprises a set of unique tokens rather than linguistic words. When tokenisation is performed at word level, the vocabulary entries correspond to individual words. Moreover, we impose an arbitrary yet fixed order on the vocabulary. This order may be alphabetical, based on frequency, or simply the order in which the words were discovered.

- **One-hot encoding**. This is a basic form of count-based representation, in which each word (or token) in the vocabulary is represented by a binary vector containing a 1 at the position corresponding to that word and 0s elsewhere.

Formally, for a vocabulary $V = [t_1, t_2, \dots, t_v]$, each token t_i is represented as a vector

$$X_i = [0, 0, \dots, 1, \dots, 0] \in \{0,1\}^v,$$

where 1 appears only in the i-th position.

Example. Consider a corpus D consisting of two documents, d_1 and d_2. For simplicity, assume that each of these documents contains only one sentence:

Document d_1: "The house is small."

Document d_2: "The house is beautiful."

After word tokenisation, the combined and sorted vocabulary is as follows:

$$V = [\text{beautiful}, \text{house}, \text{is}, \text{small}, \text{the}].$$

---

[163] The term 'sparse' describes a property of the feature vectors used to represent text in symbolic, count-based models. A sparse representation is one in which most of the entries in the feature vector are zero. By contrast, dense representations (such as word embeddings) have nearly all non-zero entries.

Each token is encoded as follows:

$$X_{beautiful} = [1, 0, 0, 0, 0]$$
$$X_{house} = [0, 1, 0, 0, 0]$$
$$X_{is} = [0, 0, 1, 0, 0]$$
$$\mathrm{X}_{\mathrm{small}} = [0, 0, 0, 1, 0]$$
$$X_{the} = [0, 0, 0, 0, 1].$$

One-hot encoding provides a clear, interpretable mapping from words to numeric vectors, and it is straightforward to construct and compute. However, it has several drawbacks, such as high dimensionality. As vocabulary size increases, memory and computational costs grow substantially.

Although one-hot encoding is a fundamental way of representing individual words numerically, it only captures their mere presence and ignores their frequency and distribution across documents. Furthermore, as each word is treated independently, one-hot vectors cannot represent multi-word units or capture document-level structure. The Bag-of-Words (BoW) model addresses these limitations by representing an entire text as a vector of token frequencies over the vocabulary.

- **Bag-of-Words (BoW) model**. Building on the concept of one-hot encoding, the Bag-of-Words model generalises the representation from individual tokens to entire documents. Rather than considering only the presence or absence of each token, the BoW model captures its frequency within the text.

In the BoW model, each document is expressed as a vector of token counts. Suppose the vocabulary V consists of v distinct tokens $V = [t_1, t_2, \dots, t_v]$. Then a document d within a corpus D is represented by a vector

$$X^d = \left[x_1^d, x_2^d, \dots, x_v^d\right],$$

where x_i^d $(i = 1,2, \dots, v)$ denotes frequency (count) of token t_i in document d. The BoW model disregards grammar and token order in d, focusing solely on token occurrence ([M28], [M32]).

Example. Consider a corpus $D = \{d_1, d_2\}$, where

Document d_1: "The house is small."

Document d_2: "The house is beautiful."

After word tokenisation, the combined vocabulary is as follows:

$$V = [\text{beautiful, house, is, small, the}].$$

In general, punctuation marks such as full stops, commas, exclamation marks and question marks are not included in the vocabulary of most bag-of-models. The BoW vectors are:

$$X^{d_1} = [0, 1, 1, 1, 1], \quad X^{d_2} = [1, 1, 1, 0, 1].$$

Although the BoW model is straightforward and interpretable, it has two main limitations: it ignores word order in sentences and produces high-dimensional, sparse vectors. These limitations can obscure the semantic relationships between words and reduce computational efficiency.

- **Term Frequency-Inverse Document Frequency (TF–IDF)**. This is an improvement on the BoW approach, which determines the importance of a term to a document within a corpus by combining its frequency in the document with its rarity across the corpus (inverse document frequency, or IDF). This process reduces the importance of common words that appear in many documents while increasing the importance of terms that are distinctive to particular documents ([R16]).

The weight of token t in document $d \in D$ is computed as:

$$TF\text{-}IDF(t,d,D) = TF(t,d) \cdot IDF(t,D),$$

where

$$TF(t,d) = \frac{count(t,d)}{\sum_{w \in d} count(w,d)}$$

is the normalised number of occurrences of t in d, and

$$IDF(t,D) = ln\left(\frac{|D|}{1+|\{d \in D : t \in d\}|}\right) + 1$$

is a common, smoothed form of the IDF. $|D|$ denotes the total number of documents in corpus D. The additive constant of 1 in the denominator is used to prevent division by zero.

Example. Consider a corpus $D = \{d_1, d_2, d_3\}$, where

Document d_1: "The house is small."

Document d_2: "The house is beautiful."

Document d_3: "A small garden beside the house."

After normalisation (lowercasing, punctuation removal) and tokenisation, we obtain the following vocabulary:

$$V = [\text{a, beautiful, beside, garden, house, is, small, the}].$$

Let us compute document frequencies (i.e. number of documents containing token):

a: 1, beautiful: 1, beside: 1, garden: 1, house: 3, is: 2, small: 2, the: 3.

For simplicity, let us compute the $TF\text{-}IDF$ weights for Document d_1 only:

Token	*TF*	*IDF*	*TF-IDF*
a	0	1.41	0
beautiful	0	1.41	0
beside	0	1.41	0
house	0.25	0.71	0.18
is	0.25	1.00	0.25
small	0.25	1.00	0.25
the	0.25	0.71	0.18

Consequently, the $TF\text{-}IDF$ vector for document d_1 is:

$$X^{d_1} = [0, 0, 0, 0.18, 0.25, 0.25, 0.18].$$

In summary, count-based representations are foundational techniques in text vectorisation. They are easy to calculate and provide understandable measures of how important each word is within and across documents. However, these representations are inherently high-dimensional and sparse. They treat each token as an independent feature, failing to capture semantic or syntactic relationships between words. Consequently, they are unable to represent phenomena such as synonymy or contextual meaning. These limitations led to the creation of embedding-based representations, which project words and phrases into dense vector spaces. These spaces encode semantic similarity and contextual information more effectively.

5.1.3.2. Embedding-based representations

Embedding-based representations map discrete linguistic units, such as words, subwords, sentences or documents, to dense, low-dimensional vectors that capture semantic and syntactic relationships. For example, word embeddings represent words as vectors in a multidimensional space, where the distance and direction between vectors reflect the similarity and relationships between the corresponding words. Unlike sparse count vectors, embeddings place similar items close together in vector space, allowing geometric operations to reflect semantic similarity.

Embeddings can be learned from co-occurrence statistics, predicted using neural objectives, or generated by large pretrained transformer encoders ([B28], [W10]).

The following two sections discuss the two main types of embedding-based representation: static word embeddings and contextual word embeddings.

5.1.3.2.1. Static word embeddings

These representations assign a single dense vector to each linguistic unit. They are based on the distributional hypothesis, which states that words that appear in similar contexts tend to have similar meanings. By training on large corpora, distributed representations capture semantic relationships: words that occur in similar contexts end up close together in the embedding space. This concept forms the basis of many word embedding models. Distributed word embeddings are called static embeddings because each word in the vocabulary is always associated with one fixed vector, regardless of the context in which the word appears.

Mathematically, an embedding can be expressed as a mapping $E: V \to \mathbb{R}^e$, where V is the vocabulary and e is the embedding dimension ([164]). Once the model has been trained, it assigns each word or token $t_i \in V$ to a dense vector X_i of dimension e: $E: t_i \to X_i \in \mathbb{R}^e$. The similarity between words is often measured using *cosine similarity*

$$sim(t_i, t_j) = \frac{X_i \cdot X_j}{\|X_i\| \|X_j\|},$$

where $\|X_k\|$ $(k = i, j)$ denotes the norm of the vector $X_k = [x_1^k, \dots, x_e^k] \in \mathbb{R}^e$:

$$\|X_k\| = \sqrt{\sum_{l=1}^{e} (x_l^k)^2}.$$

Several computational frameworks have been developed to learn distributed representations from large text corpora. The most influential of these are Word2Vec and GloVe, which formalise the distributional hypothesis into concrete learning objectives. Although both aim to capture semantic similarity between words, their underlying principles differ fundamentally: Word2Vec takes a predictive approach based on local context windows, whereas GloVe uses a count-based method that leverages global co-occurrence statistics. Together, these models have laid the groundwork for modern word embedding techniques and remain key reference points for understanding the evolution of representation learning in natural language processing.

- **Word2Vec**. Prior to the development of Word2Vec, the majority of approaches to text representation relied on sparse, high-dimensional vectors, such as one-hot encodings or bag-of-words counts. While these representations were simple, they were also limited. They treated words as isolated symbols, ignored semantic relationships and produced vectors that were too large to be practical for many tasks.

The introduction of Word2Vec by Tomas Mikolov and his team at Google in 2013 ([M33]) was a significant breakthrough. It offered a method of learning dense, low-dimensional embeddings that captured syntactic and semantic regularities alike. Word2Vec is a neural model that learns these vectors by solving a prediction task based on the distributional hypothesis. The central idea is that words that appear in similar contexts should have similar vector representations. This innovation formed the basis of modern NLP by bridging the gap between symbolic language and statistical learning, setting the stage for the more advanced embedding techniques that followed.

---

[164] The embedding dimension is a hyperparameter that is chosen by the practitioner. It is not learned automatically. For many NLP tasks, typical values range from 100–300, though smaller dimensions (50–100) can be used for lightweight models, and larger dimensions (500+) may be required for specialised domains. The choice balances semantic richness against computational cost.

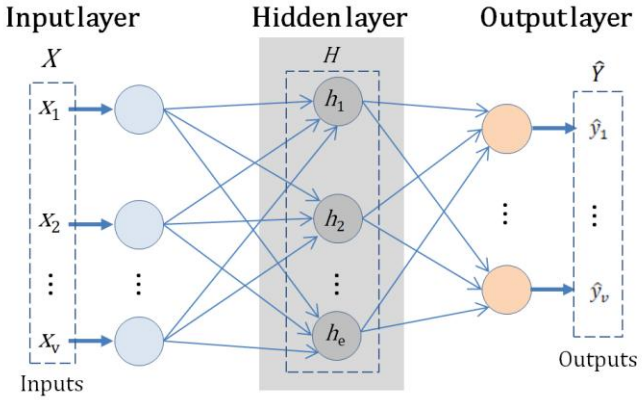
Word2Vec has two main variants: CBOW and Skip-gram. Both share the same shallow neural architecture and embedding matrices, as well as a similar training objective and optimisation methods. The main difference lies in the direction of prediction: CBOW predicts the centre word from its context, whereas Skip-gram predicts the context words from the centre.

Let us examine this common neural architecture in more detail. Let v denote the size of the vocabulary V and e the embedding dimension. Each word $w \in V$ is represented by a one-hot vector $X_w \in \mathbb{R}^v$. Given a document consisting of a sequence of words $w_1, \ldots, w_T \in V$, we define a context window of size c. For each word w_t, $1 \leq t \leq T$, the context C_t contains the words

$$C_t = [w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c}].$$

This requires these indices to remain within the boundaries of the document. Therefore, the inequalities $1 \leq t - c$ and $t + c \leq T$ must be satisfied. Implementations usually truncate the window dynamically at the edges of a document, e.g. for $t < c$, the left context is smaller. Typical choices are $2 \leq c \leq 5$ for local syntax and $5 \leq c \leq 8$ for global semantics. In both CBOW and Skip-Gram, the context window moves systematically across the entire corpus, token by token, and this process is repeated until the entire corpus has been processed.

The neural network model is a shallow, two-layer network with the following structure:



The input layer is not trainable; it is just a one-hot representation of a word w from the vocabulary. The one-hot vector X_w is a fixed encoding – no parameters are learned here.

The hidden layer is the first trainable layer. It is represented by the weight matrix $U_{ih} \in \mathbb{R}^{v \times e}$, which maps one-hot vectors into input embeddings. Let $w \in V$ and X_w be its one-hot representation: $X_w = [0, 0, \ldots, 1, \ldots, 0]$. The corresponding input embedding of w is then selected by multiplying U_{ih} by the one-hot representation X_w:

$$K_w = U_{ih}^T X_w = (U_{ih})^j,$$

where $1 \leq j \leq v$ is the index of '1' in X_w. In other words, the input embedding K_w of w is equal to the j^{th} row of the matrix U_{ih}. These input embeddings are then used to compute the hidden vector H, which acts as an intermediate representation that is passed to the output layer.

The output layer is the second trainable layer. It consists of the output embedding matrix $U_{ho} \in \mathbb{R}^{e \times v}$, which maps the hidden vector back into vocabulary space and enables word prediction. More precisely, the output matrix U_{ho} contains exactly one embedding (column) vector per vocabulary word. This means that there is a one-to-one correspondence between vocabulary items and the columns of U_{ho}. The hidden vector $H \in \mathbb{R}^e$ is projected back into vocabulary space via $Z = U_{ho}^T H \in \mathbb{R}^v$. Note that $z_l = \left(U_{ho}^T\right)_l$, where $\left(U_{ho}^T\right)_l$ is the l-th column of U_{ho}^T. The *softmax* function then converts the score vector Z into probabilities over all words:

Now, let us take a closer look at the CBOW and Skip-gram models.

- *Continuous Bag of Words* (CBOW) ([165]). CBOW learns to predict a target word based on its contextual surroundings within a fixed-size window. Unlike count-based methods such as TF–IDF, it is a predictive model that learns dense, low-dimensional word embeddings by optimising the parameters of a neural network.
  - *Input layer.* In the CBOW model, the input layer acts as an interface between textual data and the neural network. Its purpose is to convert the context words surrounding a target word w_t into a numerical form suitable for learning. Each context word $w_j \in C_t$ is encoded as a one-hot vector X_j.
  - *Hidden layer (input embedding layer)*. This is the first trainable layer. For a given training instance (C_t, w_t), the one-hot context vectors are projected into a shared embedding space by being multiplied by the input weight matrix U_{ih}. The hidden vector is computed as the mean of the input embeddings in the context window:

$$H = H(C_t, w_t) = \frac{1}{2c}\sum_{-c \le j \le c, j \neq 0} K_{t+j} = \frac{1}{2c}\sum_{-c \le j \le c, j \neq 0} U_{ih}^T X_{t+j},$$

    where X_{t+j} is one-hot vector of w_{t+j}. The hidden vector $H \in \mathbb{R}^e$ provides a fixed-size representation of the context C_t that captures the local semantic environment of the target word w_t. This averaged embedding serves as input to the output layer, where the model predicts the probability distribution over possible centre words.
  - *Output layer (softmax)*. This is the second trainable layer. It is represented by the output weight matrix $U_{ho} \in \mathbb{R}^{e \times v}$, which maps the hidden vector back into vocabulary space for prediction. First, the hidden vector H is multiplied by $U_{ho}$to produce a score vector for all vocabulary words: $Z = U_{ho}^T H \in \mathbb{R}^v$. Next, the scores are passed through a *softmax* function to produce a probability distribution over the vocabulary:

$$\hat{Y} = softmax(Z),$$

    where

$$\hat{y}_l = softmax(z_l) = \frac{exp\,(z_l)}{\sum_{k=1}^{v} \exp\,(z_k)}, l = 1,2, \dots, v.$$

  - *Training* process. The model is trained using pairs (C_t, w_t) and the output $\hat{Y}$ is compared to the expected value $Y = X_t$. The weight matrices U_{ih} and U_{ho} are randomly initialised (e.g. with small Gaussian or uniform values). During training, backpropagation is used to update these matrices, thereby minimising the cross-entropy loss L, which for a one-hot target vector Y is given by

$$L = -\sum_{k=1}^{v} y_k\, log\,(\hat{y}_k) = -Y^T \cdot log\,(\hat{Y}).$$

    It is easy to see (cf. [B29]) that

$$\frac{\partial L}{\partial z_l} = \hat{y}_l - y_l, \text{ i.e. } \frac{\partial L}{\partial Z} = E,$$

    where $E = \hat{Y} - Y$.

    To compute the gradient $\frac{\partial L}{\partial U_{ho}}$ let us observe that gradient with respect to column $(U_{ho})_l$ is

$$\frac{\partial L}{\partial (U_{ho})_l} = \frac{\partial L}{\partial z_l} \cdot \frac{\partial z_l}{\partial (U_{ho})_l} = \left(\frac{\partial L}{\partial z_l}\right) H.$$

    Stacking all columns we get

[165] The term 'bag of words' reflects the fact that context words are treated as an unordered set, with their positions being ignored. The 'continuous' part highlights the fact that the model operates on continuous-valued vectors rather than sparse counts.

$$\frac{\partial L}{\partial U_{ho}} = H \cdot \left(\frac{\partial L}{\partial Z}\right)^T = H \cdot E^T.$$

The update rule is

$$U_{ho} \leftarrow U_{ho} - \alpha \frac{\partial L}{\partial U_{ho}},$$

where α is learning rate.

The gradient $\frac{\partial L}{\partial U_{ih}}$ of the loss with respect to the input matrix U_{ih} is obtained by backpropagating the error from the output layer into the hidden vector H. This gradient is then distributed equally across the context word embeddings that were averaged to form H. Since $Z = U_{ho}^T H$ we get $\frac{\partial Z}{\partial H} = U_{ho}^T$ and consequently

$$\frac{\partial L}{\partial H} = \left(\frac{\partial Z}{\partial H}\right)^T \cdot \frac{\partial L}{\partial Z} = U_{ho} \cdot \frac{\partial L}{\partial Z}.$$

As H is the average of the context embeddings

$$H = \frac{1}{2c} \sum_{j=1}^{2c} U_{ih}^T X_j,$$

the gradient with respect to each $K_j = U_{ih}^T X_j$ is

$$\frac{\partial L}{\partial K_j} = \frac{\partial L}{\partial H} \cdot \frac{\partial H}{\partial K_j} = \frac{1}{2c} \cdot \frac{\partial L}{\partial H}.$$

But each K_j is just the row of U_{ih} corresponding to word w_j. So the gradient with respect to that row is

$$\frac{\partial L}{\partial (U_{ih})^j} = \frac{1}{2c} \cdot \frac{\partial L}{\partial H}.$$

Only the rows of U_{ih} corresponding to the context words are updated; all other rows remain unchanged. The update rule is

$$(U_{ih})^j \leftarrow (U_{ih})^j - \alpha \frac{\partial L}{\partial (U_{ih})^j}.$$

After training, the rows of U_{ih} (or sometimes the average of U_{ih} and U_{ho}^T) are used as the final word embeddings.

Example. Consider the vocabulary

$$V = [\text{the, cat, sat, on, mat}]$$

and set the embedding dimension to $e = 2$. Initialise the weight matrices $U_{ih} \in \mathbb{R}^{5\times 2}$ and $U_{ho} \in \mathbb{R}^{2\times 5}$ with random values:

$$U_{ih} = \begin{bmatrix} 0.3 & 0.2 \\ 0.4 & 0.3 \\ 0.2 & 0.6 \\ 0.5 & 0.1 \\ 0.1 & 0.0 \end{bmatrix}, \quad U_{ho} = \begin{bmatrix} 0.2 & 0.1 & 0.6 & 0.4 & 0.0 \\ 0.1 & 0.5 & 0.0 & 0.1 & 0.3 \end{bmatrix}.$$

Let us assume that we have a training toy sentence:

"The cat sat on the mat."

Consider a context window C of size $c = 1$, and generate training pairs by sliding this window across the sentence:

Target word w_t	Context window C_t	Training pair
cat	[the, sat]	([the, sat], cat)
sat	[cat, on]	([cat, on], sat)
on	[sat, the]	([sat, the], on)
the	[on, mat]	([on, mat], the)

The first and last words are skipped unless padding is used.

Now, we need to iterate over these training pairs and apply a gradient update to weights U_{ih} and U_{ho}. To keep things simple, let us only consider the pair ([cat, on], sat). The one-hot vectors X_{cat}, X_{on} and Y are

$$X_{cat} = [0,1,0,0,0]^T,\ X_{on} = [0,0,0,1,0]^T,\ Y = Y_{sat} = [0,0,1,0,0]^T.$$

The context vector H is then

$$H = \frac{1}{2}\left(U_{ih}^T X_{cat} + U_{ih}^T X_{on}\right) = \frac{1}{2}([0.4, 0.3]^T + [0.5, 0.1]^T) = [0.45, 0.2]^T.$$

The pre-softmax scores z_k and $exp(z_k)$ are computed as

$$\begin{aligned}
z_{the} &= [0.2, 0.1] \cdot [0.45, 0.2]^T = 0.11 \quad \Rightarrow \exp(z_{the}) \approx 1.116 \\
z_{cat} &= [0.1, 0.5] \cdot [0.45, 0.2]^T = 0.145 \Rightarrow \exp(z_{cat}) \approx 1.156 \\
z_{sat} &= [0.6, 0.0] \cdot [0.45, 0.2]^T = 0.27 \quad \Rightarrow \exp(z_{sat}) \approx 1.310 \\
z_{on} &= [0.4, 0.1] \cdot [0.45, 0.2]^T = 0.20 \quad \Rightarrow \exp(z_{on}) \approx 1.221 \\
z_{mat} &= [0.0, 0.3] \cdot [0.45, 0.2]^T = 0.06 \quad \Rightarrow \exp(z_{mat}) \approx 1.062.
\end{aligned}$$

Consequently, $S = \sum_{k=1}^{5} \exp(z_k) = 5.866$ and we obtain the following probability distribution $\hat{Y}$ over the vocabulary:

$$\begin{aligned}
p(z_{the}|\, context) &= 1.116/5.866 \approx 0.190 \\
p(z_{cat}|\, context) &= 1.156/5.866 \approx 0.197 \\
p(z_{sat}|\, context) &= 1.310/5.866 \approx 0.223 \\
p(z_{on}|\, context) &= 1.221/5.866 \approx 0.208 \\
p(z_{mat}|\, context) &= 1.062/5.866 \approx 0.181.
\end{aligned}$$

Therefore, the model assigns the highest probability to "sat" ($\approx$ 0.223) given the context [cat, on].

For target word “sat”, the loss is cross-entropy

$$L = -\log p(z_{sat}|\, context).$$

Backpropagation will be used to update the U_{ih} and U_{ho} matrices.

First, let us compute the gradient of L with respect to U_{ho}. The error vector E is given by $\hat{Y} - Y$, where Y is the one-hot encoded target vector and $\hat{Y}$ is the predicted distribution vector:

$$\begin{aligned}
E &= [0.190, 0.197, 0.223, 0.208, 0.181]^T - [0,0,1,0,0]^T \\
&= [0.190, 0.197, -0.777, 0.208, 0.181]^T .
\end{aligned}$$

Hence

$$\frac{\partial L}{\partial U_{ho}} = H \cdot E^T = [0.45, 0.2]^T \cdot [0.190, 0.197, -0.777, 0.208, 0.181]$$

$$= \begin{bmatrix} 0.086 & 0.089 & -0.350 & 0.094 & 0.081 \\ 0.038 & 0.039 & -0.155 & 0.042 & 0.036 \end{bmatrix}$$

and assuming learning rate $\alpha = 0.1$, we obtain

$$U_{ho}^{new} = U_{ho} - \alpha \frac{\partial L}{\partial U_{ho}} = \begin{bmatrix} 0.191 & 0.091 & 0.635 & 0.391 & -0.008 \\ 0.096 & 0.496 & 0.016 & 0.096 & 0.296 \end{bmatrix}.$$

Now, we will derive the gradient with respect to the input matrix U_{ih}, so that we can see the full backpropagation path from $L \rightarrow U_{ho} \rightarrow H \rightarrow U_{ih}$. As shown above, the gradient of L with respect to the hidden vector H is

$$\frac{\partial L}{\partial H} = U_{ho} \cdot \frac{\partial L}{\partial Z} = U_{ho} \cdot E =$$

$$= \begin{bmatrix} 0.2 & 0.1 & 0.6 & 0.4 & 0.0 \\ 0.1 & 0.5 & 0.0 & 0.1 & 0.3 \end{bmatrix} [0.190, 0.197, -0.777, 0.208, 0.181]^T = \begin{bmatrix} -0.325 \\ 0.193 \end{bmatrix}.$$

Because H is an average of two context embeddings, each embedding receives half of this gradient

$$\frac{\partial L}{\partial K_{cat}} = \frac{\partial L}{\partial K_{on}} = \frac{1}{2} \cdot \frac{\partial L}{\partial H} = [-0.163, 0.096]^T,$$

where $K_{cat} = U_{ih}^T X_{cat}$ and $K_{on} = U_{ih}^T X_{on}$. Thus,

$$(U_{ih}^{new})^{cat} = (U_{ih})^{cat} - \alpha(\frac{\partial L}{\partial K_{cat}})^T = [0.4, 0.3] - [-0.016, 0.010] = [0.416, 0.290],$$

$$(U_{ih}^{new})^{on} = (U_{ih})^{on} - \alpha(\frac{\partial L}{\partial K_{cat}})^T = [0.5, 0.1] - [-0.016, 0.010] = [0.516, 0.090].$$

Consequently,

$$U_{ih}^{new} = \begin{bmatrix} 0.300 & 0.200 \\ 0.416 & 0.290 \\ 0.200 & 0.600 \\ 0.516 & 0.090 \\ 0.100 & 0.000 \end{bmatrix} \begin{matrix} \\ cat \\ \\ on \\ \end{matrix}.$$

In this step, only the context rows of U_{ih} are updated. Over many updates, context words that co-occur with the same targets are pushed in similar directions, forming meaningful embeddings.

It is important to note that, in the CBOW model, the context is derived from the input sentence, rather than from the full vocabulary. Although the vocabulary defines the set of all possible tokens and their corresponding indices in the embedding matrices, the context for each training instance consists only of the words neighbouring the current target word within a predefined window size c.

- *Skip-gram model* ([166]). This model was also introduced by Mikolov et al. ([M33]) and is the second major architecture within the Word2Vec framework. Unlike the CBOW model, which predicts a target word from its surrounding context, the Skip-gram model does the inverse: it predicts context words given a target word. This makes the Skip-gram model particularly effective at capturing representations for rare words and modelling longer-range dependencies in language.

  Essentially, the Skip-gram model uses the same shallow, two-layer neural network architecture as the CBOW model. However, there are some differences.

[166] In the field of natural language processing, the term 'gram' refers to an *n-gram*, which is a contiguous sequence of n tokens (typically words) taken from a piece of text. Unlike traditional n-grams, skip-grams do not require words to be contiguous. Instead, they 'skip' across a window of text: given a centre word, they predict words that appear within a certain distance ($\pm c$ positions).

- *Input layer*. In the Skip-gram model, the input layer receives a single target (centre) word $w_t \in C_t$, represented as a one-hot encoded vector X_t.
- *Hidden layer (input embedding layer)*. The one-hot vector X_t is projected into the embedding space by multiplication with the input weight matrix U_{ih}. The resulting vector

$$K = K(w_t) = U_{ih}^T X_t,$$

is the input embedding (or centre-word vector), which serves as a distributed semantic representation of the target word. Unlike the CBOW model, which averages the embeddings of multiple context words, the Skip-gram model uses only one word at the input layer – the centre word whose meaning will be used to predict its surrounding context C_t.

- *Output layer (softmax)*. In this layer the output weight matrix $U_{ho} \in \mathbb{R}^{e \times v}$ maps the vector K back into vocabulary space for prediction

$$Z = U_{ho}^T K \in \mathbb{R}^v.$$

Then Z is transformed into probabilities using the *softmax* function:

$$\hat{Y} = softmax(Z),$$

where

$$\hat{y}_l = p(w_l \mid w_t) = softmax(z_l) = \frac{exp\ (z_l)}{\sum_{k=1}^{v} \exp\ (z_k)}, l = 1,2, \dots, v.$$

- *Training process*. The Skip-gram model is trained to maximise the likelihood of observing actual context words given the current centre word. For each training pair (w_t, C_t) the model minimises

$$-log\ p\big(w_{t+j} \mid w_t\big), j \in \{-c, \dots, -1, 1, \dots, c\}$$

over all words within a predefined context window of size c. This means that each centre–context pair (w_t, w_{t+j}) defines its own local loss term. The overall objective is simply the sum (or average) of all these per-pair losses across the corpus:

$$L = -\sum_{t=1}^{T} \sum_{-c \le j \le c, j \neq 0} log\ p\big(w_{t+j} \mid w_t\big)$$

It is important to understand how the Skip-gram model formally represents the target output during training. In the original formulation by Mikolov et al., the Skip-gram model does not predict the entire context simultaneously. Instead, it treats each context word as an independent input. For a centre word w_t in a context window C_t, the model generates a separate training instance for each word in $w_{t+j} \in C_t$, with the true label Y_{t+j} equal to the one-hot vector of w_{t+j}. Consequently, rather than being a single multi-label or multi-hot vector, the target value of the context C_t is a set of individual one-hot target vectors.

In summary, CBOW and Skip-gram are two complementary approaches to learning distributed word representations within the same basic neural framework. However, their predictive objectives differ fundamentally. CBOW predicts a single target word from its surrounding context, averaging the embeddings of multiple context words to determine the most probable centre word. In contrast, Skip-gram takes a centre word as input and predicts its surrounding context words individually. Consequently, CBOW tends to perform better with larger datasets and more frequent words, since it averages multiple input embeddings to smooth over the context. Skip-gram, on the other hand, is more effective for smaller datasets and rare words because it learns distinct representations by treating each context–target pair separately. Despite their structural similarity, these models therefore capture complementary statistical aspects of language. CBOW efficiently aggregates contextual information, while Skip-gram models the details of the dependencies between individual words.

While Word2Vec models learn word representations through local context prediction, alternative approaches such as GloVe (*Global Vectors for Word Representation*) use global co-occurrence statistics from the entire corpus. The following section introduces the GloVe model, including its theoretical motivation, mathematical formulation and practical advantages, as a complementary method for learning distributed word embeddings.

- **GloVe**. Unlike Word2Vec, which learns word embeddings by predicting local contextual relationships, GloVe, introduced by Jeffrey Pennington et al. in 2014 ([P15]), is based on a different principle: it uses global corpus statistics. Rather than relying solely on sliding-window predictions, GloVe constructs embeddings by factorising a word–context co-occurrence matrix. This captures the distributional structure of the entire corpus in a mathematically founded way. This hybrid approach combines local context sensitivity with global statistical consistency to address the limitations of purely predictive methods.

Word2Vec models do not explicitly encode global patterns, such as the frequency with which words appear together across the entire corpus. In contrast, classical matrix factorisation methods (e.g. LSA ([167])) utilise corpus-wide co-occurrence statistics, but are computationally expensive and less effective at capturing fine-grained linguistic regularities. GloVe unifies these approaches by creating embeddings in which the differences between word vectors encode meaningful ratios of co-occurrence probabilities. These ratios have been shown to capture semantic structure (e.g. analogies).

- *Co-occurrence matrix*. This matrix forms the basis of the model. It records how often words appear in proximity to each other within a corpus. The corpus itself is not treated as an ordered set but rather as a source of local co-occurrence statistics.

  To build the matrix, we first define the vocabulary V and then slide a context window across the corpus. For each occurrence of a word in the corpus, we examine the surrounding context and record co-occurrence counts. After scanning the entire corpus, these counts are aggregated into a co-occurrence matrix that is indexed by vocabulary.

  For a given corpus, a fixed window size c (e.g., 10 words) defines which neighbouring words constitute the context. For every pair (w_i, w_j), the model counts how often word w_j appears in the context of word w_i. These counts form the entries of the co-occurrence matrix $C \in \mathbb{R}^{v \times v}$ ([168]):

  $$c_{ij} = \textit{the number of times word } j \textit{ occurs in the context of word } i.$$

  However, the contribution of word w_j is distance-weighted. Specifically, if a word w_j is d positions away from w_i, it contributes as follows to the occurrence c_{ij}:

  $$c_{ij} = c_{i(i+d)} \leftarrow c_{ij} + \frac{1}{|d|}.$$

  This ensures the matrix captures semantic closeness more accurately.

  The final co-occurrence matrix C is large and sparse. Each row corresponds to a word, and each column shows how frequently other words appear near it in the corpus. GloVe then learns word embeddings by factorizing this matrix, ensuring that semantic relationships are encoded in the resulting vector space.

  Example. Suppose we have a corpus consisting of just three short sentences:

  - "The cat sat on the mat."

---

[167] *Latent Semantic Analysis* (LSA) is an unsupervised natural language processing technique that analyses the relationships between words and documents. It does this by first building a term-document matrix, and then decomposing it to reveal hidden semantic structures. Instead of looking for exact matches, LSA groups words and documents by meaning. It is widely used for information retrieval, document similarity, and topic modelling ([M34]).

[168] In standard GloVe training, the co-occurrence count c_{ii} for a word i is typically zero, since a token is not considered part of its own context window. Only words appearing within the specified left and right context of a target token contribute to its co-occurrence counts, so self-co-occurrences do not arise unless explicitly introduced by a modified implementation.

- "The dog sat on the log."
- "The cat chased the dog."

After normalisation (lowercasing, punctuation removal), we obtain the following vocabulary:

$$V = [\text{ cat, chased, dog, log, mat, on, sat, the}].$$

So our vocabulary size is $v = 8$. Let us take a window of size $c = 2$ and slide it across the corpus to count how often words appear near each other:

- Sentence 1:
  - "cat" → neighbours: "the", "sat", "on"
  - "mat" → neighbours: "the", "on"
  - "on" → neighbours: "cat", "sat", "the", "mat"
  - "sat" → neighbours: "the", "cat", "on", "the"
  - "the" → neighbours: "cat", "sat"
  - "the" → neighbours: "sat", "on", "mat"
- Sentence 2:
  - "dog" → neighbours: "the", "sat", "on"
  - "on" → neighbours: "dog", "sat", "the", "log"
  - "log" → neighbours: "the", "on"
  - "sat" → neighbours: "the", "dog", "on", "the"
  - "the" → neighbours: "dog", "sat"
  - "the" → neighbours: "sat", "on", "log"
- Sentence 3:
  - "cat" → neighbours: "the", "chased", "the"
  - "chased" → neighbours: "the", "cat", "the", "dog"
  - "dog" → neighbours: "chased", "the"
  - "the" → neighbours: "cat", "chased"
  - "the" → neighbours: "cat", "chased", "dog".

Here is a simplified co-occurrence matrix (counts only, not weighted by distance):

Word	cat	chased	dog	log	mat	on	sat	the
cat	0	1	0	0	0	1	1	3
chased	1	0	1	0	0	0	0	2
dog	0	1	0	0	0	1	1	2
log	0	0	0	0	0	1	0	1
mat	0	0	0	0	0	1	0	1
on	1	0	1	1	1	0	2	2
sat	1	0	1	0	0	2	0	4
the	3	2	2	1	1	2	4	0

It is important to note that the co-occurrence matrix is defined at word type level. This means that each vocabulary item corresponds to exactly one row and one column in the matrix, regardless of how frequently the corresponding word appears in the corpus. Consequently, whenever a word occurs in a sentence, all context words within the specified window contribute to the same row-column entries. For instance, if the word 'the' appears multiple times in a corpus, the count $X_{the,j}$ for each context word j increases with every instance of 'the' in a context window surrounding it. Therefore, the matrix aggregates global co-occurrence statistics across all occurrences and sentences.

- *Training objective*. GloVe learns two sets of embeddings:
  - word vectors $X_i = X(w_i) \in \mathbb{R}^e$
  - context vectors $\tilde{X}_j = \tilde{X}(w_j) \in \mathbb{R}^e$

  and scalar bias terms b_i and $\tilde{b}_j$. The GloVe objective function is a least-squares loss

  $$L = \sum_{i,j:c_{ij}>0} f(c_{ij}) \left(X_i^T \tilde{X}_j + b_i + \tilde{b}_j - log\ c_{ij}\right)^2,$$

  where
  - $log\ c_{ij}$ is the target (true value),
  - $X_i^T \tilde{X}_j + b_i + \tilde{b}_j$ is the model prediction,
  - f is a weighting function that reduces the influence of very frequent or very rare co-occurrences. It is defined as follows:

    $$f(x) = f(x) = \begin{cases} \left(x/x_{max}\right)^{\alpha}, & x < x_{max} \\ 1, & otherwise \end{cases}$$

    where x_{max} is threshold (e.g., 100) and α is smoothing parameter (commonly 0.75).

  Since raw counts can vary widely, the GloVe model uses the logarithm of co-occurrence counts rather than the counts themselves. This stabilises the training process and reflects the idea that differences in log-counts are more meaningful than differences in raw counts.

  GloVe learns word embeddings by factoring the co-occurrence matrix using weighted least squares. For each word pair (i, j), the model minimises the difference between the dot product of their embeddings and the logarithm of their co-occurrence count: $X_i^T \tilde{X}_j \approx log\ c_{ij}$. In other words, the dot product of word vectors encodes co-occurrence statistics. This establishes a connection between global corpus statistics (via the co-occurrence matrix) and local vector geometry (via embeddings).

  The model is trained using the gradient descent method, with the gradients of its loss function being computed via backpropagation. For each co-occurrence pair (i, j), the gradient of the loss with respect to X_i, $\tilde{X}_j, b_i, \tilde{b}_j$ is computed. The parameters are then updated iteratively using stochastic gradient descent.

Despite their historical importance and widespread adoption, static word embeddings exhibit several intrinsic limitations that constrain their effectiveness in modern NLP systems. Most fundamentally, these models assign one fixed vector representation to each word, regardless of the linguistic context in which the word appears. As a result, static embeddings cannot adequately model polysemy (see Section 5.1.1) or contextually driven semantic variation (e.g., "cold" describing weather versus personality). This lack of contextual sensitivity limits their effectiveness in tasks requiring nuanced understanding, such as question answering or sentiment analysis. Furthermore, static embeddings are trained on fixed corpora and are unable to adapt dynamically to new domains or evolving language use, which makes them susceptible to domain mismatch. They also struggle with out-of-vocabulary words, since unseen tokens cannot be represented without retraining.

To overcome these fundamental limitations, modern NLP systems are increasingly relying on contextual word embeddings. These embeddings generate word vectors that vary according to the surrounding linguistic context, offering a substantially more expressive and linguistically informed foundation for subsequent tasks.

5.1.3.2.2. Contextual word embeddings

Contextual word embeddings are derived from models that process entire sentences (or larger segments of text) and generate representations of individual tokens based on the full

context. Rather than assigning a single embedding X_w to a word type w, contextual models compute the following:

$$X_t = f(w_1, w_2, \ldots, w_n),$$

where X_t represents the contextual embedding of the token at position t, and $f(\cdot)$ denotes a deep sequence model (e.g. RNN – see Section 4.3.6, or Transformer – see Section 5.2.3) that processes all tokens simultaneously.

These representations encode the meaning of a token as it is realised within a specific sentence, capturing syntactic role, semantic nuances, long-range dependencies and discourse-level relations. Consequently, the word "bank" in "The fisherman sat by the bank of the river" will have a different vector representation to its occurrence in "The bank approved the loan".

Prior to the introduction of the Transformer architecture, contextual embeddings were predominantly obtained using bidirectional recurrent neural networks. Models such as ELMo use deep bidirectional LSTMs (see Section 4.3.6.4) to compute the representation of each token based on information flowing from the contexts to the left and right of it ([169]).

Although these architectures represented a significant improvement, they struggled to model complex, long-range dependencies efficiently due to their inherent sequential processing constraints. This limitation paved the way for self-attention-based architectures, which can relate any pair of tokens directly and in parallel, thus fundamentally changing contextual embedding design.

Modern contextual embeddings, which are used in state-of-the-art systems such as BERT, GPT and T5 (all of which are Transformer-based large language models), are derived from the self-attention mechanism (see Section 5.2.5). This mechanism computes token representations by directly attending to all other tokens in a sequence. This approach produces richer, more scalable and more globally coherent representations than earlier recurrent models. Therefore, contextual embeddings act as a conceptual bridge between traditional vector-based representations and fully-fledged LLMs. In the following section, we will examine how attention mechanisms and Transformer architectures generate these representations at scale.

5.2. Large Language Models (LLMs)

Large Language Models (LLMs) represent a fundamental shift in the design, scalability and versatility of natural language processing (NLP) systems. Motivated by the limitations of earlier NLP approaches, such as Word2Vec and GloVe, Peters et al. ([P18]) and Devlin et al. ([D12]) demonstrated that deeper models trained with self-supervised objectives could produce contextual embeddings reflecting syntactic and semantic distinctions dependent on sentence-level context. Nevertheless, early architectures, primarily based on recurrent neural networks and their gated variants (see Section 4.3.6), struggled with long-range dependencies, sequential bottlenecks and poor scalability.

A decisive breakthrough was achieved with the introduction of the Transformer architecture in a paper titled "*Attention is All You Need*" by Vaswani et al. (Google Brain Team and Google Research), which is based on the self-attention mechanism ([V2]). This mechanism enables models to capture global dependencies in parallel, allowing them to be trained efficiently on massive corpora. This architectural shift paved the way for the subsequent rapid growth in model capacity and capability. Researchers at OpenAI (Radford et al. [R17] and [R18]; Brown et al. [B30]) and Google (Raffel et al. [R19]; Chowdhery et al. [C13]) demonstrated that increasing the number of parameters, the size of the dataset, and the computational budget results in

[169] ELMo (*Embeddings from Language Models*) is a deep contextual word embedding model that was introduced in 2018 by researchers from the Allen Institute for AI and the University of Washington. Unlike earlier static embeddings such as Word2Vec or GloVe, it generates word representations that vary depending on the surrounding context ([V6]).

consistent improvements in performance — an idea that was formalised by Kaplan et al. ([K19]) in the form of scaling laws for neural language models. These findings provided both theoretical and empirical motivation for the development of increasingly large and capable LLMs.

While earlier generations of language models relied on RNNs and LSTMs, almost all modern LLMs (GPT 3/4/5, PaLM, LLaMA, Claude, Gemini, etc.) are now based on the Transformer architecture. This has become the dominant foundation due to its scalability and effectiveness. Although a few emerging research models explore alternatives (see Section 5.2.7), Transformers remain the standard in practice.

LLMs adopt a unified modelling paradigm whereby a single model learns general-purpose internal representations through large-scale pretraining. This model can then be adapted for many different subsequent tasks via fine-tuning, prompting, or in-context learning. This versatility has led to significant progress in areas such as translation, summarisation, answering questions, retrieving knowledge, reasoning and generating content. Furthermore, unlike earlier models that required task-specific architectures or handcrafted features, LLMs minimise this need.

The rest of this section builds on these foundations. First, we introduce the attention mechanism, which is the conceptual core of Transformers. Then, we examine the full Transformer architecture, followed by a discussion of how LLMs learn through self-supervised objectives. We also review major Transformer-based language models, from BERT to Microsoft Copilot, and explore their practical applications. Finally, we address some of the limitations and open challenges.

Before introducing the attention mechanism itself, we will examine how textual input is converted into the numerical representations that Transformers operate on. This conversion forms the essential interface between raw language and the model's internal computations.

5.2.1. Input representations in Transformer-based models

The natural language input must first be converted into a sequence of numerical vectors that the model can process using the attention mechanism. Although modern Transformer architectures ultimately produce contextual representations, they begin with a familiar input stage that closely resembles the embedding pipelines introduced earlier in this chapter. This process starts with standard text pre-processing, including normalisation, sentence segmentation and tokenisation. Transformers typically use subword segmentation methods, such as Byte-Pair Encoding (BPE) or WordPiece (see Section 5.1.2.1), to map the resulting tokens to unique integer indices in the model's vocabulary.

These token IDs are then converted into dense vector embeddings via a learned embedding matrix. Much like the static embeddings described in Section 5.1.3.2.1, these vectors are context-independent at this initial stage; every instance of the word "cat", for example, starts with the same vector. However, it is important to note that these embeddings are part of the model's trainable parameters and are updated during large-scale pretraining. As such, they serve as initial representations that will later be transformed into context-sensitive variants by the model's internal layers.

As attention lacks any intrinsic notion of sequence order, the model supplements token embeddings with positional encodings that capture the relative or absolute position of each token within the input sequence. Positional vectors are fixed or learned vectors that encode the positional index of each token ([170]). For a token at position pos, a vector $P_{pos} = [p_{pos,1}, \dots, p_{pos,d}]$ is added to its embedding E_{pos}

[170] Positional information is not inherent to self-attention and must be added explicitly. In the original Transformer, positional encodings took the form of fixed sinusoidal functions and were therefore not learned. Many subsequent architectures (such as BERT or GPT-style models) use learned positional embeddings or relative position biases instead. However, both learned and non-learned variants are compatible with the attention mechanism.

$$X_{pos} = E_{pos} + P_{pos}.$$

This means that the model recognises not only the semantic embedding of the word, but also its position within the sequence. Transformers use sinusoidal positional encoding. For a model with an embedding dimension of d (where d is assumed to be an even number), the positional encoding $p_{pos,k}$ for position pos in dimension k ($1 \leq k \leq d$) is defined as follows ([V2])

$$p_{pos,k} = sin\left(\frac{pos}{10000^{(k-1)/d}}\right),\ if\ k\ is\ odd,$$

$$p_{pos,k} = cos\left(\frac{pos}{10000^{(k-2)/d}}\right),\ if\ k\ is\ even.$$

The quantities $10000^{(k-1)/d}$ and $10000^{(k-2)/d}$ control the wavelength of the sine/cosine functions, creating different frequencies across dimensions. The value of 10,000 in the base ensures that the wavelengths of the sine and cosine functions span a wide range, from very short to very long cycles. This enables positional encoding to represent both fine-grained local order (nearby tokens) and long-range dependencies (tokens that are far apart). Without such a large base, the frequencies would be too similar and the encodings would not cover enough positional variation ([171]). Finally, this sequence of enriched vectors, each of which combines lexical identity and positional information, enters the first attention layer.

Remark. Positional encodings are computed for each input sequence (typically a sentence, paragraph, document or batch element), rather than for the entire corpus. Attention always takes a matrix of position-augmented embeddings for one sequence as input. During training, multiple sequences are batched together. Each sequence has its own positional indices. Padding tokens are added if the lengths differ, and their positions are usually masked out. Attention runs independently on each batch element. In practice, frameworks (PyTorch, TensorFlow) combine sentences into a batch tensor. However, attention is masked so tokens in one element cannot attend to tokens in another.

In summary, positional encodings are computed per sequence, not across the entire corpus. Positions restart at 1 for each new sentence. This ensures that repeated words in different sentences can be distinguished by their local order, while attention layers provide further context.

Example. Consider the corpus consisting of the single sentence:

"The cat sat on the mat."

Using a simple word tokeniser, we obtain the following token sequence:

[the] [cat] [sat] [on] [the] [mat] [.]

A vocabulary is built by collecting all unique tokens in the corpus, and each token is assigned a unique integer index ID. These IDs are arbitrary but fixed, and they do not encode any semantic, syntactic or statistical information. The index simply acts as a stable look-up key. Let the vocabulary V be:

[171] In the sinusoidal positional encoding, each dimension k uses a frequency proportional to $10000^{-k/d}$. Because the exponent $-k/d$ varies linearly with k, the resulting frequencies form a geometric progression. More precisely, if $\omega_k = 10000^{-k/d}$, then $\omega_{k+1}/\omega_k = 10000^{-1/d}$, a constant ratio. This geometric spacing ensures that the corresponding wavelengths $\lambda_k = 2\pi/\omega_k$ span several orders of magnitude: small k yields large ω_k (short wavelengths), while large k yields small ω_k (long wavelengths). Choosing a large base such as 10,000 increases the ratio between successive wavelengths, thereby guaranteeing broad coverage from high frequency to low frequency components across the embedding dimensions.

Token	Assigned *ID*
[the]	1
[cat]	2
[sat]	3
[on]	4
[mat]	5
[.]	6
[PAD]	0
[UNK]	7

The [PAD] token (padding) ensures that all sequences within a batch are the same length, enabling parallel computation. These positions are excluded through masking later on, so they do not influence the model's representations. The [UNK] token (unknown) serves as a fallback for words or subword units that cannot be mapped to any entry in the model's vocabulary. Together, these special tokens enable Transformer-based models to reliably process text even when sequence lengths and lexical forms are variable.

To keep things simple, we will ignore the tokens [PAD] and [UNK] in this example. Thus, our token sequence becomes the ID sequence:

$$[1, 2, 3, 4, 1, 5, 6].$$

The token IDs are then converted into dense vector embeddings, i.e. there is a mapping $ID \to E_{ID} \in \mathbb{R}^d$, where d is the embedding dimensionality (a hyperparameter, e.g., $d = 4$ for illustration). Let the (randomly initialised) 4-dimensional embeddings be:

Token	*ID*	Embedding E_{ID}
[the]	1	[1.0, 0.5, 0.2, 0.1]
[cat]	2	[0.4, 0.3, 0.8, 0.2]
[sat]	3	[0.6, 0.1, 0.3, 0.9]
[on]	4	[0.2, 0.9, 0.4, 0.1]
[mat]	5	[0.3, 0.7, 0.2, 0.6]
[.]	6	[0.05, 0.02, 0.01, 0.03]

Replacing token IDs by the corresponding embedding vectors, we can stack them into a trainable embedding matrix $E \in \mathbb{R}^{v \times d}$, where $v = 7$ is the vocabulary size ([172])

$$E = \begin{bmatrix} 1.0 & 0.5 & 0.2 & 0.1 \\ 0.4 & 0.3 & 0.8 & 0.2 \\ 0.6 & 0.1 & 0.3 & 0.9 \\ 0.2 & 0.9 & 0.4 & 0.1 \\ 1.0 & 0.5 & 0.2 & 0.1 \\ 0.3 & 0.7 & 0.2 & 0.6 \\ 0.05 & 0.02 & 0.01 & 0.03 \end{bmatrix}.$$

Now, let us compute the positional encodings. For dimension $d = 4$, we have

$$p_{pos,1} = sin\left(\frac{pos}{10000^{0/4}}\right) = sin(pos),$$

$$p_{pos,2} = cos\left(\frac{pos}{10000^{0/4}}\right) = cos(pos),$$

$$p_{pos,3} = sin\left(\frac{pos}{10000^{2/4}}\right) = sin(pos/100),$$

$$p_{pos,4} = cos\left(\frac{pos}{10000^{2/4}}\right) = cos(pos/100).$$

Consequently,

[172] Note that training modifies the embeddings, i.e. the elements in the rows of the matrix, but does not alter their positions within the matrix.

$$P_1 = [sin(1), cos(1), sin(0.01), cos(0.01)] = [0.84147, 0.54030, 0.01000, 0.99995],$$
$$P_2 = [sin(2), cos(2), sin(0.02), cos(0.02)] = [0.90930, -0.41615, 0.02000, 0.99980],$$
$$P_3 = [sin(3), cos(3), sin(0.03), cos(0.03)] = [0.14112, -0.98999, 0.03000, 0.99955],$$
$$P_4 = [sin(4), cos(4), sin(0.04), cos(0.04)] = [-0.75680, -0.65364, 0.03999, 0.99920],$$
$$P_5 = [sin(5), cos(5), sin(0.05), cos(0.05)] = [-0.95892, 0.28366, 0.04998, 0.99875],$$
$$P_6 = [sin(6), cos(6), sin(0.06), cos(0.06)] = [-0.27942, 0.96017, 0.05996, 0.99820],$$
$$P_7 = [sin(7), cos(7), sin(0.07), cos(0.07)] = [0.65699, 0.75390, 0.06994, 0.99755].$$

The input to the attention mechanism is always a matrix of position-augmented embeddings:

$$X = E + P,$$

where

$$x_{ij} = e_{ij} + p_{ij}, i = 1, \dots, 7 \; and \; j = 1, \dots, 4.$$

Thus,

$$X = \begin{bmatrix} 1.84147 & 1.04030 & 0.21000 & 1.09995 \\ 1.30930 & -0.11615 & 0.82000 & 1.19980 \\ 0.74112 & -0.88999 & 0.33000 & 1.89955 \\ -0.55680 & 0.24636 & 0.43999 & 1.09920 \\ 0.04108 & 0.78366 & 0.24998 & 1.09875 \\ 0.02058 & 1.66017 & 0.25996 & 1.59820 \\ 0.70699 & 0.77390 & 0.07994 & 1.02755 \end{bmatrix}.$$

This matrix is the actual input to the attention mechanism.

A matrix of enriched elements, combining both lexical identity and positional information, enters the first attention layer. It is only at this stage that tokens begin to acquire contextual semantics, since attention operations allow each position to incorporate information from the other positions in the sequence. Thus, the transformation of static input embeddings into fully contextualised representations is entirely driven by the feed-forward and stacked attention layers, which are described in the following section.

5.2.2. The attention mechanism

Human cognitive processing is fundamentally shaped by attention: the selective allocation of mental resources to the most relevant stimuli in a complex environment. Attention filters amplify and prioritise information, enabling us to function effectively in such environments. Without this filtering, we would be overwhelmed by sensory input.

For instance, when reading a book in a noisy café, your brain filters out irrelevant stimuli, such as clattering dishes and background chatter, and focuses on the text. You can maintain your focus on the book for an extended period, resisting distractions. This enables you to build a coherent understanding of multiple sentences and paragraphs. However, when a friend suddenly calls your name, your attention shifts from the book to your surroundings. Yet if you try to read while talking to your friend, your performance drops, demonstrating the limitations of attentional resources.

Attention acts as a gatekeeper, prioritising certain sensory inputs for deeper processing in working memory. Neural evidence shows that attended stimuli result in stronger activation in sensory cortices, while unattended stimuli are suppressed. This selective amplification shapes what enters consciousness and what gets encoded into long-term memory. The notion of attentional selectivity is not restricted to hearing; it also applies to other senses, such as sight and smell. Biological organisms quickly identify visual cues in order to determine where to look next to achieve their goal. For example, when looking for a street number, we first focus on the doorknob, and we know from experience (i.e. our trained neurons tell us) to look to its upper

left or right to find it. A similar observation holds for neural networks where attention is applied to diverse domains, such as computer vision and natural language processing.

In NLP, computational attention mechanisms are inspired by this principle. They enable neural networks to weight different parts of an input sequence according to their contextual importance, allowing models to focus on the most informative tokens when constructing meaning. Although these mechanisms are far simpler than human attentional systems, the architectural analogy is conceptually powerful: in machine learning, attention serves as a differentiable, mathematically controlled analogue of selective focus. This allows models to modulate information flow rather than relying on rigid, fixed-context representations.

The attention mechanism emerged as a pivotal innovation in modern NLP, fundamentally transforming how models handle long-range dependencies in text. Before the introduction of attention, recurrent architectures such as RNNs and LSTMs processed sequences sequentially, generating hidden states that implicitly encoded all relevant contextual information. As sequence length increased, these models struggled to retain and transmit information across distant positions, even with gating mechanisms. Introduced in the context of neural machine translation, the attention mechanism addressed this issue by enabling a model to focus selectively on the most relevant parts of the input sequence when producing each output representation. This allows direct, dynamically weighted access to contextual information, independent of serial propagation.

5.2.2.1. Mathematical formulation

Transformers rely on a computational principle called *attention*, which enables the model to dynamically assign different weights to the various components of an input sequence when representing each token. Unlike fixed-size context windows or sequential recurrence, attention computes interactions between all tokens in parallel. This allows the model to determine the most relevant words in a given context.

Although the Transformer architecture supports several forms of attention, the mathematical formalism presented here focuses on self-attention. In this case, the queries, keys and values are all derived from the same input sequence. Self-attention is the core operation in every Transformer encoder layer and also appears in decoder layers. Other variants, most notably cross-attention where the queries originate in the decoder and the keys and values in the encoder, will be discussed later when addressing full encoder–decoder architectures (see Section 5.2.3). For now, self-attention provides the foundational mechanism upon which all other forms are built.

- **Input matrix and linear projections**. Let an input sequence be tokenised and embedded as a initial hidden state matrix $X \in \mathbb{R}^{n \times d}$, where n is the sequence length and d is the *model dimension* (also called *embedding dimension* or *hidden size*, and sometimes denoted d_{model}) ([173]). Each row X_i is the embedding (typically embedding + positional encoding) for token i.

---

[173] Note the difference between 'model size' and 'model dimension'. The model dimension d is the fixed dimensionality of the token representations used throughout a Transformer. In contrast, model size refers to the total number of trainable parameters in a neural network.
The exact specifications of the architecture of current LLMs are not publicly disclosed. However, educated estimates based on known Transformer scaling laws and the published dimensions of earlier model generations suggest that GPT-5 likely employs a model dimension d of around 20,000–28,000, consistent with a parameter count of approximately one to two trillion. DeepSeek-V3, which has a mixture-of-experts (MoE) design (see Section 5.2.5.4), probably operates with a more moderate hidden size of around 10,000–14,000. Microsoft Copilot integrates several model variants, but its frontier-class reasoning models are expected to fall within a similar range: 12,000–24,000. These figures should be interpreted as informed approximations rather than official architectural details.

The model dimension d and sequence length n are hyperparameters. The former controls the width of token representations, while the latter controls the size of the context window. In typical Transformer models, d ranges from several hundred to tens of thousands, whereas n ranges from a few hundred to several thousands. Although n is usually smaller than d in practical architectures, the two quantities are independent and neither constrains the other.

Self-attention uses three linear transformations (projections) $\mathbb{R}^d \to \mathbb{R}^{d_k}$, which are represented by the following learned parameter matrices:

$$W_Q, W_K, W_V \in \mathbb{R}^{d \times d_k},$$

where d_k is a hyperparameter in architecture design, controlling representational capacity and computational efficiency. Transformer-based models use multi-head attention, in which the full embedding dimension d is split into h parallel attention 'heads', i.e. $d_k = d/h$ (see Section 5.2.2.2) ([174]).

Applying these transformations to the input matrix X yields the following $n \times d_k$ matrices:

$$Q = XW_Q,\ K = XW_K,\ V = XW_V,$$

which are called *queries*, *keys*, and *values*, respectively ([175]). Although these operations are written as matrix multiplications, they are effectively row-wise transformations: each embedding row vector $X_i = [x_{i1}, x_{i2}, \dots, x_{id}], i = 1,2, \dots, n$, is independently mapped to

$$Q_i = X_i W_Q,\ K_i = X_i W_K,\ V_i = X_i W_V,$$

where, for example,

$$Q_i = \left[q_{i1}, q_{i2}, \dots, q_{id_k}\right] \text{ and } q_{ij} = \sum_{l=1}^{d} x_{il} w_{lj}^{Q}, j = 1, 2, \dots, d_k.$$

Thus, every embedding vector is processed using the same learned linear map, and no information is exchanged across row vectors at this stage. All cross-token interactions occur later, when attention scores are computed ([176]).

Note that the learned projections W_Q, W_K and W_V map the input representation X into three different subspaces. The resulting matrices Q, K and V are not learned parameters, but rather input-dependent representations produced by these projections.

Remark. To gain a deeper insight into the Transformer design, it is important to understand the purpose of the Q, K and V matrices. Although they are all derived from the same input embeddings, they each play a distinct role in determining how tokens interact with one another.

- *Queries* (Q) represent what each token is asking for. A query encodes the perspective from which a token seeks information. For instance, the word 'sat' might query for subjects (such as 'cat') to determine who acted.

---

[174] The head dimension d_k is not a property of the input matrix $X \in \mathbb{R}^{n \times d}$, i.e. d_k is independent of both the sequence length n and the model dimension d. In practice, however, multi-head attention chooses d_k so that d can be evenly divided by the number of attention heads h. Thus $d_k < d$ is a practical convention, not a theoretical requirement. The internal attention configurations of current LLMs are not publicly disclosed. Nevertheless, reasonable estimates can be derived from standard Transformer design patterns and the published dimensions of earlier model generations. For GPT-5, which has a model dimension in the range of 20k–28k, it is likely that the model would employ between 200 and 350 attention heads. This would give a per-head dimension d_k of approximately 64–96, which is consistent with the head sizes used in GPT-3 and the estimated architecture of GPT-4. DeepSeek-V3 would plausibly use 150–220 heads, corresponding to the 64-dimensional head size widely adopted in recent MoE architectures (see Section 5.2.5.4). The models integrated into Microsoft Copilot are expected to fall within a similar range, with approximately 150–300 attention heads and per-head dimensions of 64–80. These figures represent informed approximations rather than official architectural specifications.

[175] For simplicity, we assume that all the projection matrices have the same dimension. However, as the matrices for queries, keys and values serve different mathematical purposes, their dimensions do not need to match. The only requirement is that keys and queries must share the same dimension d_k, which is necessary for dot-product scaling. The architecture does not constrain the dimension of W_V.

[176] In the context of attention, the word 'token' always refers to the embedding vector of a given token. It is these final vectors, rather than the raw text tokens, that serve as the inputs to attention. Therefore, when we say, for example, that "token i attends to token j", we actually mean that the embedding vector at position i interacts with the embedding vector at position j through the learned attention mechanism.

- *Keys* (K) represent what each token is offering. A key encodes the features that make a token available for matching. Continuing the example, the word 'cat' provides a key that signals that it is a potential subject.
- *Values* (V) represent the payload of information that is passed along once a match is made. If 'sat' refers to 'cat', the value vector of 'cat' contributes to the contextual representation of 'sat'.

By learning three distinct linear projections W_Q, W_K and W_V, the model can remap tokens into specialised subspaces for querying, matching and transmitting information. Queries and keys define a similarity function via dot products, while values define the information flow. This separation enables the model to learn nuanced relationships, such as syntactic (subject–verb), semantic (topic similarity) and positional (order sensitivity).

In summary, the purpose of the projected representations Q, K and V is to provide the model with flexible, learnable roles for tokens: queries ask, keys answer and values deliver. This separation makes attention expressive enough to capture complex relationships within and across sequences.

- **Computing attention scores**. The attention mechanism converts a sequence of tokens into contextualised representations by enabling each token to 'look at' the others in the sequence. This process involves computing attention scores to determine how strongly one token attends to another.

For each query vector Q_i (row i of Q) and key vector K_j (row j of K), a *similarity score* (raw score) is computed via a scaled dot product

$$s_{ij} = \frac{Q_i \cdot (K_j)^T}{\sqrt{d_k}}.$$

Thus, s_{ij} measures how relevant token j is to token i ([177]). On the other hand, the dot product s_{ij} also called *alignment score* because it indicates how strongly the decoder aligns with a specific encoder position (see Section 5.2.3.5). These two terms refer to the same quantity, but from different perspectives. Stacking these scores for all pairs yields the *score matrix* (also known as *similarity matrix* or *alignment scores*)

$$S = \frac{QK^T}{\sqrt{d_k}} \in \mathbb{R}^{n \times n}.$$

The scaling factor $1/\sqrt{d_k}$ stabilises gradients by preventing dot products from becoming excessively large in high-dimensional spaces.

In the next step, each row of S is normalised by *softmax*, which gives the *attention weights*, also termed the *attention matrix*

$$A = softmax(S) \in \mathbb{R}^{n \times n},$$

where

$$a_{ij} = \frac{exp(s_{ij})}{\sum_{k=1}^{n} exp(s_{ik})}.$$

[177] The dot product $Q_i \cdot (K_j)^T$ measures similarity between token positions i and j. If their projections derived from embeddings capture similar semantic or syntactic information, the dot product becomes large, leading to a higher attention weight. Conversely, if they represent unrelated or conflicting information, the dot product is small or negative, resulting in weak attention. Thus, attention learns patterns such as which adjectives modify which nouns, which verbs depend on which subjects and which tokens are relevant for predicting masked or next tokens. More abstract structures are learned in deeper layers. The interpretation is always grounded in embedding vectors rather than raw text.

Each row A_i represents a probability distribution over all positions in the sequence, i.e. the entry a_{ij} quantifies the extent to which token i attends to token j. In other words, A_i expresses where the query at position i directs its attention.

- **Weighted aggregation of values**. Once the attention matrix has been computed, a contextualised representation of X is calculated for each token as a weighted sum of all value vectors

$$Z = Attention(Q, K, V) = AV \in \mathbb{R}^{n \times d_k} \text{ (178)}.$$

This step integrates information from all tokens through a weighted linear combination of their value vectors with weights given by the learned attention coefficients. It is at this stage that contextualisation occurs: the representation of token i is enriched with information drawn from other tokens in the sequence. It is worth noting that this value aggregation is the only step in the attention mechanism at which token representations are explicitly mixed.

The rows of the attention output Z are traditionally called *context vectors*. Thus, each context vector

$$Z_i = \sum_{k=1}^{n} a_{ik} V_k, i = 1,2, \dots, n,$$

is the output of the attention mechanism: a weighted combination of the value vectors, where the weights reflect the relevance of each source position to the current query. It is the information retrieved from the sequence, conditioned on the query.

The full attention pipeline

$$X \to (Q, K, V) \to QK^T \to A \to Z$$

implements a flexible, content-based retrieval mechanism. This mechanism enables the model to detect long-range dependencies, syntactic relations, semantic associations and many other patterns, all without recurrence or convolution.

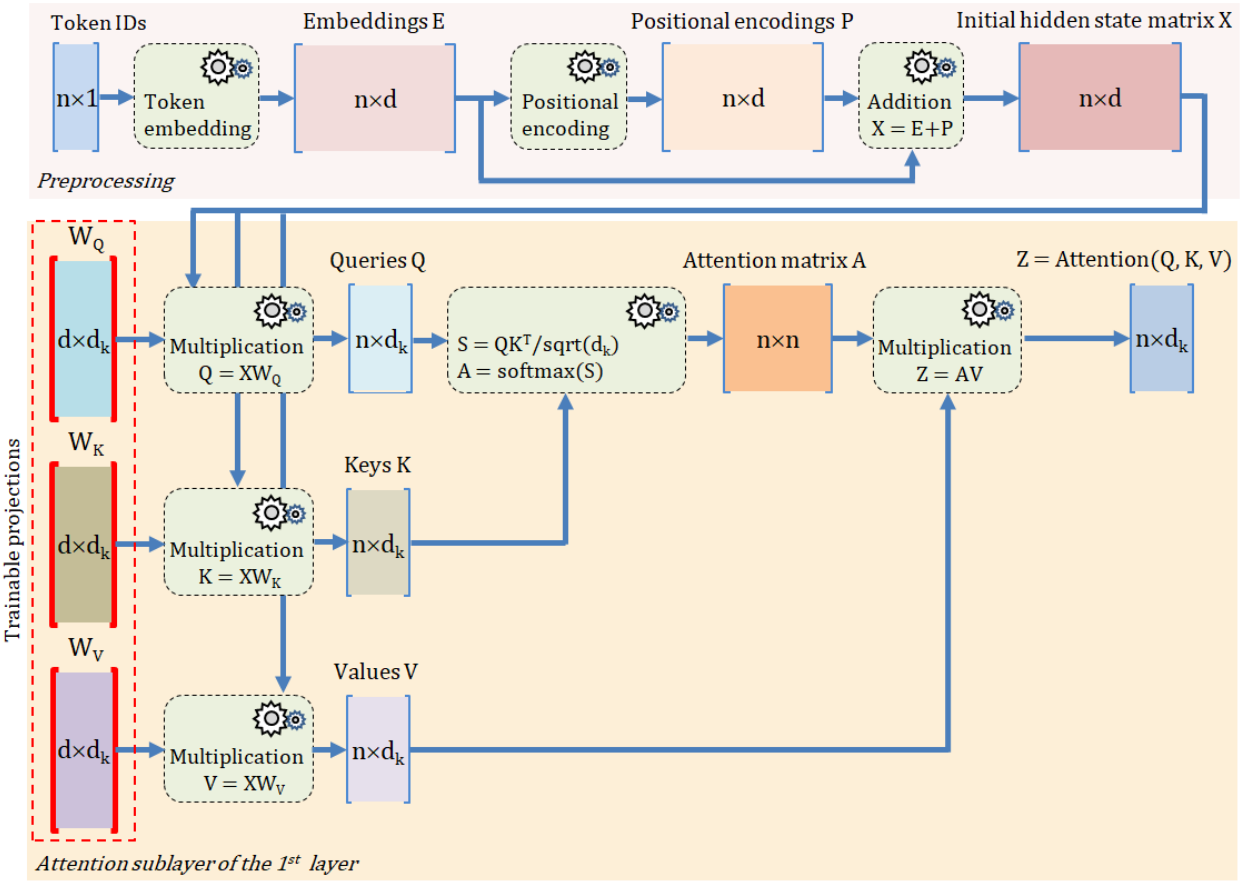
Remark. Although the attention mechanism is implemented as a sequence of linear projections and similarity computations, its behaviour can be interpreted as a form of content-based retrieval. At a functional level, this is loosely analogous to aspects of how humans recall information. In human cognition, retrieval is organised by the content of the current mental state rather than the physical location of stored information: a new stimulus activates representations that are semantically or contextually related to it. Similarly, the attention mechanism compares each query vector with all key vectors and retrieves the corresponding value vectors in proportion to their similarity. Therefore, the selection of relevant information is governed entirely by the relationships encoded in the representations themselves rather than by positional or structural constraints. This analogy does not imply biological similarity but provides a useful conceptual framework: both systems prioritise information based on content-dependent relevance rather than fixed indexing.

Before we turn to a concrete numerical example, it is helpful to take a look at the full attention pipeline. The diagram below illustrates how X moves through the projections, similarity computation, softmax normalisation and final aggregation stages (179).

---

178 In the Transformer literature, the *attention mechanism* is the mapping $(Q, K, V) \to Attention(Q, K, V)$. This is a function of three arguments. However, in most contexts, the term *attention* refers specifically to the matrix A.

179 I would like to thank Dr Kunde for sharing a draft of this diagram.



Example. Consider the corpus consisting of the sentence

"The cat sat on the mat."

Let us take the embedding matrix from Section 5.2.1 as input

$$X = \begin{bmatrix} 1.84147 & 1.04030 & 0.21000 & 1.09995 \\ 1.30930 & -0.11615 & 0.82000 & 1.19980 \\ 0.74112 & -0.88999 & 0.33000 & 1.89955 \\ -0.55680 & 0.24636 & 0.43999 & 1.09920 \\ 0.04108 & 0.78366 & 0.24998 & 1.09875 \\ 0.02058 & 1.66017 & 0.25996 & 1.59820 \\ 0.70699 & 0.77390 & 0.07994 & 1.02755 \end{bmatrix},$$

and set $h = 2$. Then $d_k = d/h = 2$. Consider the following projection matrices

$$\mathrm{W}_Q = \begin{bmatrix} 0.9 & -0.1 \\ 0.3 & 0.3 \\ 0.4 & 0.9 \\ 0.7 & 0.8 \end{bmatrix}, \mathrm{W}_K = \begin{bmatrix} 0.7 & 0.2 \\ 0.3 & -0.1 \\ -0.2 & 0.4 \\ 0.2 & 0.0 \end{bmatrix}, \mathrm{W}_V = \begin{bmatrix} 0.5 & -0.3 \\ -0.1 & 0.2 \\ 0.3 & 0.1 \\ 0.0 & 0.4 \end{bmatrix}.$$

Then Q, K and V are (rounded)

$$Q = X\mathrm{W}_Q = \begin{bmatrix} 2.82 & 1.20 \\ 2.31 & 1.53 \\ 1.86 & 1.48 \\ 0.52 & 1.40 \\ 1.14 & 1.33 \\ 1.74 & 2.01 \\ 1.62 & 1.06 \end{bmatrix}, K = X\mathrm{W}_K = \begin{bmatrix} 1.78 & 0.35 \\ 0.96 & 0.60 \\ 0.57 & 0.37 \\ -0.18 & 0.04 \\ 0.43 & 0.03 \\ 0.78 & -0.06 \\ 0.92 & 0.10 \end{bmatrix}, V = X\mathrm{W}_V = \begin{bmatrix} 0.88 & 0.12 \\ 0.91 & 0.15 \\ 0.56 & 0.39 \\ -0.17 & 0.70 \\ 0.02 & 0.61 \\ -0.08 & 0.99 \\ 0.30 & 0.36 \end{bmatrix}.$$

Now, let us compute the scaled dot-product scores (rounded)

$$S = \frac{QK^T}{\sqrt{2}} = \begin{bmatrix} 3.85 & 2.42 & 1.44 & -0.33 & 0.89 & 1.51 & 1.91 \\ 3.29 & 2.22 & 1.32 & -0.26 & 0.74 & 1.21 & 1.60 \\ 2.71 & 1.89 & 1.13 & -0.20 & 0.60 & 0.97 & 1.31 \\ 1.00 & 0.95 & 0.57 & -0.03 & 0.19 & 0.23 & 0.43 \\ 1.76 & 1.34 & 0.81 & -0.11 & 0.38 & 0.57 & 0.83 \\ 2.68 & 2.03 & 1.22 & -0.17 & 0.58 & 0.88 & 1.26 \\ 2.30 & 1.55 & 0.92 & -0.18 & 0.52 & 0.85 & 1.12 \end{bmatrix}.$$

Consequently, applying row-wise *softmax* gives us attention weights (rounded)

$$A = softmax(S) = \begin{bmatrix} 0.61 & 0.15 & 0.06 & 0.01 & 0.03 & 0.06 & 0.09 \\ 0.53 & 0.18 & 0.07 & 0.02 & 0.04 & 0.07 & 0.10 \\ 0.44 & 0.20 & 0.09 & 0.02 & 0.05 & 0.08 & 0.11 \\ 0.23 & 0.21 & 0.15 & 0.08 & 0.10 & 0.10 & 0.13 \\ 0.32 & 0.21 & 0.12 & 0.05 & 0.08 & 0.10 & 0.13 \\ 0,43 & 0.22 & 0.10 & 0.02 & 0.05 & 0.07 & 0.10 \\ 0.40 & 0.19 & 0.10 & 0.03 & 0.07 & 0.09 & 0.12 \end{bmatrix}.$$

For token i, row A_i tells us how much attention token i pays to every token in the sentence, including itself. More precisely, an entry a_{ij} represents the probability that the token at position i attends to the token at position j. This quantifies how strongly the query at position i selects the value vector associated with position j. The key at position j determines whether the token is relevant, while the value at position j contains the information that is actually passed forward.

For example, $a_{21} = 0.53$ means that token 2 ("cat") pulls 53% of its information from the value vector of the first token ("the"). If a_{ii} is large, the model emphasises the token's own meaning. If a_{ij} is small, token i mostly ignores token j.

However, note that because the projection matrices in the example were not trained, they do not encode meaningful linguistic or semantic relationships.

Finally, we compute the attention output (rounded)

$$Z = AV = \begin{bmatrix} 0.61 & 0.15 & 0.06 & 0.01 & 0.03 & 0.06 & 0.09 \\ 0.53 & 0.18 & 0.07 & 0.02 & 0.04 & 0.07 & 0.10 \\ 0.44 & 0.20 & 0.09 & 0.02 & 0.05 & 0.08 & 0.11 \\ 0.23 & 0.21 & 0.15 & 0.08 & 0.10 & 0.10 & 0.13 \\ 0.32 & 0.21 & 0.12 & 0.05 & 0.08 & 0.10 & 0.13 \\ 0,43 & 0.22 & 0.10 & 0.02 & 0.05 & 0.07 & 0.10 \\ 0.40 & 0.19 & 0.10 & 0.03 & 0.07 & 0.09 & 0.12 \end{bmatrix} * \begin{bmatrix} 0.88 & 0.12 \\ 0.91 & 0.15 \\ 0.56 & 0.39 \\ -0.17 & 0.70 \\ 0.02 & 0.61 \\ -0.08 & 0.99 \\ 0.30 & 0.36 \end{bmatrix} = \begin{bmatrix} 0.72 & 0.23 \\ 0.69 & 0.25 \\ 0.65 & 0.28 \\ 0.49 & 0.38 \\ 0.56 & 0.34 \\ 0.66 & 0.28 \\ 0.60 & 0.31 \end{bmatrix}.$$

Each of the seven tokens is transformed from a four-dimensional embedding into a two-dimensional attended representation. The attention mechanism combines information from different tokens based on the similarity of the queries and keys. For instance, if the word "cat" is closely related to "sat", its new vector will contain grammatical and semantic information about the action. The dimensions of Z_i have no direct human meaning, but their relative weights reflect the way in which information from different words is combined. Although these dimensions are abstract – learned by the model – they encode rich linguistic structure. In the full Transformers model, where d lies between 10,000 and 30,000, each dimension represents a small part of a much richer meaning.

The numerical example simply illustrates the mechanics of the computation. It does *not* demonstrate the meaningful patterns that would result from learning.

In summary, self-attention converts an input sequence into contextualised token representations by calculating the relevance of each token to every other token and using these scores to create weighted combinations of the value vectors. This mechanism forms the conceptual centrepiece of the Transformer architecture. The next section builds on this to explain the full multi-head attention mechanism and the broader Transformer layer structure.

5.2.2.2. Multi-head attention

Multi-head attention is an extension of the basic attention mechanism that enables Transformers to capture different types of relationships simultaneously. While single-head attention provides a highly effective method of computing contextualised token representations, it is fundamentally limited by structural constraints. A single attention head learns one set of projections (W_Q, W_K and W_V) and therefore discovers *one* notion of similarity or dependency pattern across tokens. However, natural language exhibits many types of relationships simultaneously, including syntactic, semantic, positional and discourse-level relationships. Using a single attention head would force all these heterogeneous dependencies to be encoded in a single subspace, which would limit the model's ability to distinguish between different linguistic structures.

Multi-head attention is the architectural solution introduced in the Transformer (Vaswani et al., 2017). The idea is straightforward yet extremely powerful. Rather than computing one attention distribution, the model computes multiple attention heads in parallel, each with its own learned projections, and then combines their results.

- **Multi-head extension**. Given an input matrix $X \in \mathbb{R}^{n\times d}$, multi-head attention introduces h independent attention heads. Each head has its own set of learned projection matrices:

$$W_Q^{(i)}, W_K^{(i)}, W_V^{(i)} \in \mathbb{R}^{d\times d_k}, i = 1,2,\dots,h.$$

Each head computes its own attention matrix

$$A^{(i)} = softmax\left(\frac{Q^{(i)}(K^{(i)})^T}{\sqrt{d_k}}\right) \in \mathbb{R}^{n\times n},$$

where $Q^{(i)} = XW_Q^{(i)}$ and $K^{(i)} = XW_K^{(i)}$, and its own attention

$$Z^{(i)} = Attention\left(Q^{(i)}, K^{(i)}, V^{(i)}\right) = A^{(i)}V^{(i)} \in \mathbb{R}^{n\times d_k}.$$

The focus of each head is on a different aspect of the sequence, such as syntax, semantics or positional cues.

- **Concatenation and output projection**. The outputs from all heads are concatenated

$$Z = \text{Concat}\left(Z^{(1)}, Z^{(2)}, \dots, Z^{(h)}\right) \in \mathbb{R}^{n\times(h\cdot d_k)} = \mathbb{R}^{n\times d}.$$

Concatenation means joining the matrices $Z^{(i)}$ along the dimension h

$$Z = \left[\begin{array}{ccc|ccc|c|ccc} & Z^{(1)} & & & Z^{(2)} & & & & Z^{(h)} & \\ \square & \cdots & \square & \square & \cdots & \square & \cdots & \square & \cdots & \square \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & & \vdots & \ddots & \vdots \\ \square & \cdots & \square & \square & \cdots & \square & & \square & \cdots & \square \end{array}\right]$$

The result is a richer representation combining multiple 'views' of the sequence. This parallel structure improves representational capacity without significantly increasing computational depth.

The concatenated output Z is then projected back into the embedding dimension d of the model

$$MHA(X) = ZW_O \in \mathbb{R}^{n\times d},$$

where $W_O \in \mathbb{R}^{(h\cdot d_k)\times d}$ is a learned matrix, and MHA stands for 'Multi-Head Attention'.

In the multi-head formulation, this additional output projection W_O is required to combine information from the different heads. This converts the block-structured concatenation into a

unified representation suitable for subsequent layers. This ensures that the final output has the same dimensions as the hidden size d of the model, enabling it to be fed into successive layers.

Remark. Think of each head as a specialist. One focuses on syntax, one on semantics, and one on positional cues. Concatenation is like gathering all their reports into one big dossier. While the heads specialise, the model needs a way to blend their findings into a coherent representation. This is precisely what the output projection does. It combines head-specific features to create a unified representation, enabling the next layer to develop richer abstractions. The matrix W_O gives the model freedom to up-weight important heads, down-weight or ignore unhelpful heads, and combine heads into higher-level features. Without W_O, the model would be forced to treat all heads equally and independently, which would be a huge limitation. But mixing heads ensures that information discovered by one head can influence all heads in the next layer. This is crucial for deep representation learning.

5.2.2.3. Emergent functional specialisation of Q, K, and V

Up to this point, we have described the roles of queries, keys and values in intuitive terms: queries ask, keys offer, and values deliver. This framing may be pedagogically helpful, but it raises an important conceptual question: *How does the model **learn** these roles?*

After all, the projection matrices W_Q, W_K and W_V are simply trainable linear maps. There is nothing in the raw parameters that explicitly labels one as 'query-like' and another as 'key-like'. So why don't these matrices collapse into the same function? And why does the model reliably discover the division of labour that we attribute to them? It is essential to understand this phenomenon. It helps to explain why attention works at all.

The answer lies not in explicit supervision, but in the structure of the attention mechanism and the gradients that flow through it. The distinct operational roles of the query, key, and value matrices emerge from the structural asymmetry of the attention mechanism and the constraints imposed by gradient-based optimisation.

- **Architectural asymmetry**. The source of differentiated roles is architectural asymmetry. The attention mechanism assigns queries, keys, and values to different positions in the computation graph:

- Queries $Q = XW_Q$ appear only on the left side of the similarity computation

$$S = \frac{QK^T}{\sqrt{d}} = [s_{ij}],$$

where $s_{ij} = \frac{Q_i \cdot (K_j)^T}{\sqrt{d}}$ and $i = 1,2,\dots,n, j = 1,2,\dots,d$.

- Keys $K = XW_K$ appear only on the right side of the same computation.
- Values $V = XW_V$ appear only in the weighted sum

$$Z = AV = [Z_1, Z_2, \dots, Z_n]^T$$

where $A = [a_{ij}], a_{ij} = softmax(s_{ij})$ and

$$Z_i = \sum_{j=1}^{n} a_{ij} V_j.$$

Thus, even though all three projected representations originate from the same hidden state X, they enter the computation in fundamentally different ways. W_Q and W_K determine the attention scores together via dot products, while W_V determines the content aggregated once the attention weights have been computed. This asymmetry is crucial. The parameters W_Q and W_K only indirectly influence the model's behaviour by shaping the similarity function that produces the attention distribution. In contrast, W_V directly controls the information that flows into the output representation after weighting.

Even before training begins, this structural asymmetry ensures that:

- W_Q influences how tokens search for information.

- W_K influences how a token is matched.
- W_V influences what information is transmitted once a match is made.

The model does not need to 'know' these roles in advance; the computation graph enforces them.

- **Gradient flow**. Because queries, keys, and values appear in different positions in the computational graph, their gradients differ systematically. During backpropagation, the loss gradient flows through the attention mechanism in three distinct ways:

- The gradient into W_Q reflects the effect of the query on the similarity scores.
- The gradient into W_K reflects the effect of the key on the similarity scores.
- The gradient into W_V reflects how the value affects the context vector.

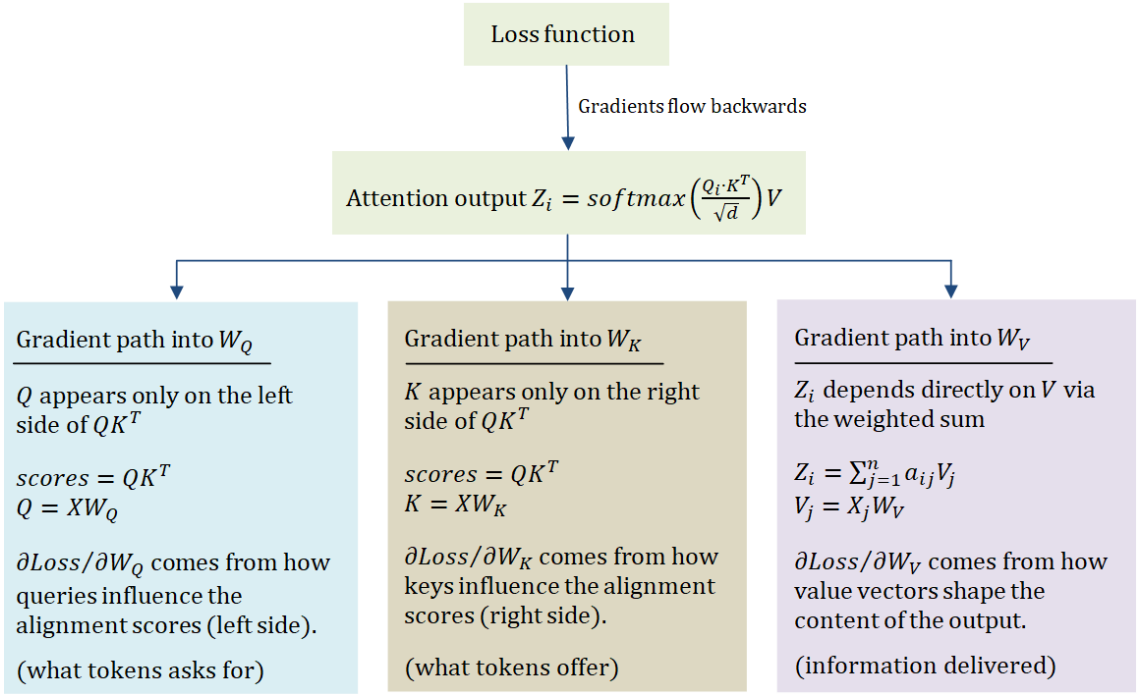
These gradients are not interchangeable. They originate from different parts of the computation graph and represent different optimisation pressures. As a result

- W_Q learns to shape the questions the tokens ask.
- W_K learns to shape the features that the tokens expose for matching.
- W_V learns to shape the information contributed by the selected tokens.

Over the course of training, gradient descent discovers projections that specialise accordingly.

This effect is amplified in multi-head attention. As we know (see Section 5.2.2.2), each head has its own triplet $(W_Q^{(i)}, W_K^{(i)}, W_V^{(i)})$, enabling different heads to specialise in different relational patterns. In practice, some heads capture syntactic dependencies (e.g., subject–verb agreement), while others capture semantic similarity, positional relations or long-range coreference. Therefore, separating the query, key and value subspaces enables role differentiation within a head and relational diversity across heads.

The following diagram illustrates the separate backward gradient flows and highlights how the architecture enforces the functional specialisation of W_Q, W_K and W_V.



- **Optimisation pressure**. If W_Q, W_K and W_V were all the same, the model would be unable to distinguish between searching and offering, produce meaningful alignment scores, or transmit the correct information after a match. This would result in degraded performance, with gradient descent pushing the matrices apart. Therefore, the training objective rewards functional

differentiation. This is analogous to how convolutional filters in a CNN learn different patterns (see Section 4.3.5). Nothing forces them to specialise, but the task makes specialisation advantageous.

- **The role of random initialization**. At initialisation, the three projection matrices are random and independent. Even slight differences in their initial values result in different gradient flows. These differences increase during training, resulting in stable, specialised roles. Consequently, the model does not require explicit instructions, such as 'this matrix is for queries'. A combination of architectural asymmetry, gradient structure and optimisation pressure ensures that these roles emerge naturally.

In summary, the functional separation of queries, keys and values does not occur through manual imposition; rather, it is an emergent property of the attention mechanism. The architecture assigns different computational roles to the three projections, backpropagation provides distinct learning signals, and the optimisation objective rewards specialisation. Consequently, the model reliably learns to use queries to ask, keys to answer and values to deliver. This emergent division of labour is one of the main reasons why attention can capture complex syntactic, semantic and positional relationships within and across sequences.

5.2.2.4. Origins of the Q–K–V formulation in attention

The query-key-value formulation is so central to modern attention mechanisms that it is easy to forget that it had to be invented. There is nothing in the basic structure of neural networks that dictates information retrieval should be expressed through three separate linear projections. So why did Vaswani et al. introduce this structure?

Understanding the historical and conceptual path that led to Q, K and V not only clarifies what attention does, but also why it is designed the way it is.

The Q-K-V formulation introduced in *Attention Is All You Need* ([V2]) did not arise in isolation. Rather, it represents a significant refinement of earlier attention mechanisms developed in sequence modelling, particularly in neural machine translation. The earliest widely adopted attention mechanism was proposed by D. Bahdanau, K. Cho, and Y. Bengio in 2014 ([B34]). In this additive attention framework, the decoder state was compared with each encoder hidden state via a learned scoring function:

$$e_{t,i} = V^T tanh(W_1 S_t + W_2 H_i),$$

where S_t denotes the decoder state at time t and $H_1, H_2, \dots, H_n$ denote the encoder hidden states. The resulting scores were normalised to produce a weighted sum of encoder representations. Although the terms 'query', 'key' and 'value' had not yet been established, the structural roles were already in place: the decoder state acted as a query, the encoder states served as keys, and the same encoder states were also used as values. A learned scoring function determined the alignment. However, these components were not yet expressed as separate representations. Instead, queries, keys and values were simply the hidden states of recurrent networks.

Subsequent work by Minh-Thang Luong et al. in 2015 ([L20]) simplified this mechanism by introducing multiplicative (dot-product) attention and replacing the additive scoring network with direct inner products. This modification reduced the computational cost and revealed a deeper geometric interpretation: attention could be viewed as a form of similarity within a learned representation space. However, the structure remained tied to the encoder-decoder split, with the encoder producing keys and values and the decoder producing queries.

The contribution of Vaswani et al. was to abstract and generalise this structure. The decisive step came with the move to self-attention. Without recurrence, every token must be able to ask, offer, and deliver information. This required three separate learned projections, yielding $Q = XW_Q$, $K = XW_K$, $V = XW_V$. As discussed previously, the architecture positions these projected representations in different areas of the computation. Queries and keys interact via the similarity matrix, whereas values only contribute to the weighted sum that constitutes the

attention output. These structural differences generate distinct gradient flows during training, ensuring that the projections specialise into their respective roles.

The attention mechanism performs a differentiable lookup operation ([180]), retrieving and aggregating information from the parts of the sequence whose learned representations are most relevant to the current query. This makes the retrieval process content-driven rather than location-driven, allowing the model to capture long-range dependencies and context-sensitive relationships within the input flexibly.

The Q-K-V formulation is therefore not arbitrary. Rather, it is the natural generalisation of earlier attention mechanisms to a fully parallel and symmetric setting. Self-attention is asymmetric with respect to the roles that Q, K and V play in the computations, but is fully symmetric at the level of tokens: every position produces a query, a key and a value using the same learned projections.

In summary, the Q-K-V formulation represents the convergence of three areas of development: additive attention in encoder–decoder models; multiplicative, similarity-based scoring; and the growing demand for parallelisable architectures. With hindsight, introducing explicit Q, K and V representations was not just a notational convenience. This decomposition's conceptual clarity contributed significantly to the Transformer framework's architectural simplicity and extensibility.

5.2.3. The Transformer architecture

The Transformer architecture, introduced by Vaswani et al. in 2017 ([V2]), represents a paradigm shift in natural language processing. Unlike earlier sequence models, such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), the Transformer relies solely on attention mechanisms to model dependencies between tokens. This design enables highly parallelisable training and has become the foundation of modern large language models.

A Transformer has two main components: an *encoder*, which converts the input sequence into contextual representations, and a *decoder*, which produces the output sequence. Both the encoder and the decoder consist of repeated layers (sometimes informally referred to as *blocks*), which comprise multi-head self-attention, position-wise feed-forward networks, and residual connections + layer normalisation (often denoted *AddNorm*). In addition, each decoder layer includes a cross-attention sublayer that attends to the encoder's output.

The Transformer processes an input sequence in two stages. First, the encoder stack runs over the source tokens and produces an output matrix M, which is stored as the encoder memory. This matrix contains contextual representations of the entire source sequence, and every decoder layer reuses it through its cross-attention sublayer. Thus, the encoder output matrix M acts as a fixed memory bank for the source sentence ([181]). However, the decoder has its own input: a sequence of target-side tokens taken from the training corpus. During training, these tokens are processed together with the encoder memory M to predict the next target token at each position. Crucially, M is recalculated for each input sequence and reflects the encoder's current parameters at that training step. After passing through all decoder layers, the final decoder representation is projected through a linear layer, followed by a *softmax* function,

---

[180] A differentiable lookup is the continuous analogue of a traditional table lookup. In a discrete lookup, selecting a single memory entry V_i using an index i is not a differentiable operation. In contrast, attention computes a weight vector A_i and returns a weighted combination Z_i. As the weights depend smoothly on Q and K via differentiable operations (matrix multiplication and softmax), the output Z varies continuously with the parameters, enabling gradients to propagate through the retrieval step. Therefore, attention replaces hard selection with similarity-weighted aggregation, preserving the functional concept of memory access while remaining compatible with gradient-based learning.

[181] Strictly speaking, the decoder does not reuse the encoder's output matrix M directly. In each cross-attention sublayer, M is projected linearly onto the keys K and values V. It is therefore the pair (K, V) that constitutes the effective memory, not M itself. Therefore, referring to M as 'encoder memory' is a conceptual simplification. The precise role of M and its relation to the KV cache is discussed in Section 5.2.3.3.

to produce a probability distribution over the vocabulary for the next token. The encoder and decoder parameters are optimised jointly and end-to-end (see Section 5.2.4).

Vaswani et al. originally introduced the full transformer architecture (encoder + decoder) for sequence-to-sequence tasks such as machine translation (e.g. English → German). However, it is not limited to translation. Since then, it has been applied to many other tasks, including summarisation (the encoder processes the document and the decoder generates a summary), question answering (the encoder encodes the question and the decoder generates the answer), and speech recognition (the encoder processes audio features and the decoder outputs text). For many natural language processing tasks, nevertheless, only the encoder or decoder is used (e.g. classification, embeddings and language modelling). Models such as BERT use the encoder stack, whereas GPT-style models use the decoder stack (see Section 5.2.5).

As Section 5.2.1 introduced positional encoding and Section 5.2.2 covered the attention mechanism and multi-head attention, this section focuses on the remaining architectural components: the encoder and decoder structure, feed-forward networks, masking, cross-attention and the AddNorm wrapper.

Further information on this subject can be found in [K20], [R20], and [T18].

5.2.3.1. Encoder structure

The encoder consists of a stack of p identical layers, typically ranging from 6 to 48 in modern LLMs. Each layer processes the entire output of the preceding layer in parallel and passes the result on to the next layer. The first encoder layer takes as input the token matrix $X^{(0)}$, which contains rows of learned token embeddings with positional encoding. The final encoder output $M = X^{(p)} \in \mathbb{R}^{n\times d}$ is the result of the pth layer.

For each $r = 1, 2, \ldots, p$, the rth layer of the encoder includes:

- **Multi-head self-attention (MHA)**. This sublayer enables each token to interact with all the other tokens in the input sequence. It takes the input matrix $X^{(r-1)} \in \mathbb{R}^{n\times d}$, concatenates the outputs of all the heads, and then projects these back into the model dimension. This results in an output matrix $MHA\big(X^{(r-1)}\big) \in \mathbb{R}^{n\times d}$, as detailed in Section 5.2.2.2.

- **Residual connection and layer normalisation (AddNorm) after MHA**. Following the multi-head self-attention sublayer, the Transformer applies a crucial component known as *AddNorm*. This mechanism combines a residual connection with layer normalisation to ensure that information flows smoothly through the network while maintaining numerical stability:

$$X^{(r-1)}_{AN(MHA)} = AddNorm\big(MHA(X^{(r-1)})\big) = LayerNorm\big(X^{(r-1)} + MHA(X^{(r-1)})\big).$$

The term 'residual connection' originates from residual networks (ResNets; He et al. [H4]), in which each layer only learns a residual function $F(X) = H(X) - X$. This ensures that the desired transformation $H(X) = X + F(X)$ is achieved. Rather than learning a complete mapping $H(X)$ from scratch, the network learns the difference between the input and the target transformation. Residual addition $X^{(r-1)} + MHA(X^{(r-1)})$ maintains the original embedding of each token, as well as the attention-based refinement.

After the residual addition $R = X^{(r-1)} + MHA(X^{(r-1)}) \in \mathbb{R}^{n\times d}$, layer normalisation is applied independently to each token vector $R_i = [\rho_{i1}, \rho_{i2}, \ldots, \rho_{id}], i = 1,2, \ldots, n$. First, R_i is normalised

$$\hat{\rho}_{ij} = \frac{\rho_{ij} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}},$$

where

$$\mu_i = \frac{1}{d}\sum_{j=1}^{d} \rho_{ij}, \quad \sigma_i^2 = \sum_{j=1}^{d} \big(\rho_{ij} - \mu_i\big)^2,$$

and ϵ is a small positive constant that prevents division by zero when the variance σ_i^2 is zero ([182]). This means that each token vector is normalised, but the tokens do not interact during the process.

The next step is to apply the following transformation to the matrix $\hat{R} = [\hat{\rho}_{ij}] \in \mathbb{R}^{n\times d}$ to get the output matrix $X_{AN(MHA)}^{(r-1)} = AddNorm\big(MHA(X^{(r-1)})\big) = [z_{ij}] \in \mathbb{R}^{n\times d}$ with

$$z_{ij} = \gamma_j \hat{\rho}_{ij} + \beta_j.$$

The vectors $[\gamma_1, \gamma_2, \dots, \gamma_d]$ and $[\beta_1, \beta_2, \dots, \beta_d]$ are learnable parameters of this sublayer. Using these vectors allows the model to rescale and reshift the normalised values $\hat{\rho}_{ij}$, enabling it to recover any required distribution.

In summary, residual connections retain the original input, even if the sublayer transformation is imperfect. On the other hand, layer normalisation prevents drift across layers by ensuring that the token embeddings are well-scaled and centred. Residual connections and layer normalisation work together to enable the encoder to be built from many repeated layers without vanishing or exploding gradients. Furthermore, each sublayer only needs to adjust the input, rather than having to reconstruct the entire mapping from scratch.

- **Position-wise Feed-Forward Network (FFN)**. After the attention and AddNorm sublayers, each encoder layer applies a two-layer FFN independently to every token representation in the matrix $X_{AN(MHA)}^{(r-1)} \in \mathbb{R}^{n\times d}$. This introduces nonlinearity and feature transformation at the level of individual token embeddings, enabling the model to capture complex relationships that cannot be accounted for by attention alone.

For each token row vector $X_{AN(MHA)}^{(r-1,i)} \in \mathbb{R}^d$ of $X_{AN(MHA)}^{(r-1)}$, $i = 1,2,\dots,n$, the FNN computes

$$FFN\left(X_{AN(MHA)}^{(r-1,i)}\right) = W_2 f\left(W_1 \cdot \left(X_{AN(MHA)}^{(r-1,i)}\right)^T + B_1\right) + B_2 \in \mathbb{R}^d,$$

where $W_1 \in \mathbb{R}^{d_{ff}\times d}$, $W_2 \in \mathbb{R}^{d\times d_{ff}}$, and $B_1 \in \mathbb{R}^{d_{ff}}$, $B_2 \in \mathbb{R}^d$ are bias vectors. All of these objects are trainable parameters of this FFN sublayer. The activation function f is *ReLU* (or *GELU* ([183]) in modern variants), and is applied element-wise. d_{ff} stands for the *feed-forward dimension* (sometimes called the *intermediate dimension*). It is typically much larger than d (e.g., 2048 when $d = 512$).

Consequently, the first linear layer expands the representation, while the second layer then compresses it back again. This 'expansion–contraction' structure significantly increases representational power. In summary, attention mixes information across tokens, but the process remains essentially linear. The FFN complements this by providing local non-linear processing for each token. It creates new features that cannot be obtained through weighted averaging. Therefore, within each encoder layer, the attention mechanism handles the global flow of information, while the FFN performs nonlinear feature extraction for each token.

Since each sublayer still needs to be stabilised and regularised, the FFN output is wrapped in an AddNorm sublayer again before being passed on to the next encoder layer.

- **Residual connection and layer normalization (AddNorm) after FNN**. A second AddNorm sublayer wraps the output $FNN(X_{AN(MHA)}^{(r-1)})$, yielding the input $X^{(r)}$ for the $(r+1)$th layer when $r < p$, or the encoder output matrix $M = X^{(p)}$ otherwise.

---

[182] The typical value of ϵ ranges from 10^{-12} to 10^{-5}, depending on the implementation.

[183] The *Gaussian Error Linear Unit* (GELU) is an activation function defined as $GELU(x) = x \cdot \Phi(x)$, where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution. Unlike *ReLU*, which sharply zeroes out negative inputs, the *GELU* smoothly scales them based on probability. This allows small negative values to contribute and large positive values to pass through almost unchanged ([H17]).

$$X^{(r)} = AddNorm\left(FNN(X^{(r-1)}_{AN(MHA)})\right) = LayerNorm\left(X^{(r-1)}_{AN(MHA)} + FNN(X^{(r-1)}_{AN(MHA)})\right).$$

Although it is mathematically identical to the AddNorm that follows MHA, it plays a slightly different functional role. After the FFN, which introduces position-wise non-linear transformations, the residual connection ensures that the model can preserve the original information while only adding refined adjustments. LayerNorm then prevents undesirable variance growth resulting from the nonlinear expansion–contraction pipeline inside the FFN. In contrast, the first AddNorm sublayer stabilises global, attention-based mixing, while the second one stabilises local, nonlinear refinement. Together, the two AddNorm sublayers maintain numerical stability, support gradient flow, and enable the stacking of an arbitrary number of encoder layers.

Each row vector in the output matrix $M = X^{(p)} \in \mathbb{R}^{n\times d}$ of the final encoder layer provides a detailed contextual representation of the input sequence. This matrix serves as the contextual memory for the cross-attention sublayer of the decoder (see Section 5.2.3.3). It paves the way for the decoder by guiding it to focus on the relevant words in the input sequence during the decoding process.

5.2.3.2. Operational modes of the Transformer model

The Transformer architecture supports two distinct operational modes: *training* and *inference*. While these modes use the same model parameters and underlying architecture, they differ in terms of their purpose, data flow and how input sequences are made available to the decoder. Understanding these modes is essential for interpreting the behaviour of the encoder–decoder system, as well as for accurately describing the decoder's internal mechanisms.

- **Training mode**. The model is optimised using a labelled corpus. During training, both the source sequence (e.g. an English sentence) and the corresponding full target sequence (e.g. its German translation) are available. The decoder receives the entire target sequence at once, shifted by one position (teacher forcing). This enables the model to compute all token predictions in parallel.

Let us take a closer look at the training data flow in the encoder-decoder Transformer, using machine translation English → German process as an example.

In training mode, the Transformer is trained using a parallel corpus consisting of paired sentences (S_{EN}, S_{DE}). For a given example from the English-to-German corpus, the English source sentence S_{EN} is first tokenised into a sequence of token IDs and mapped to learned token embeddings. After adding positional encodings, an input matrix $X \in \mathbb{R}^{n\times d}$ is obtained ([184]), as explained in Section 3.2.1. This matrix is then fed into the encoder to produce the encoder output $M \in \mathbb{R}^{n\times d}$.

The German target sentence S_{DE} is tokenised into a sequence of token IDs, which typically includes the special tokens BOS (*Beginning of Sequence*) and EOS (*End of Sequence*). This sequence is known as the 'ground-truth' sequence and forms the basis of everything the decoder uses during training

$$GTS = [id_{BOS},\ id_1, id_2, \dots, id_m, id_{EOS}].$$

[184] In practical implementations, input is a batch consisting of many pairs. Consequently, rather than dealing with a single $n \times d$ matrix, models work with $b \times n \times d$ tensors, where b is the batch size (i.e. the number of sequences in a batch), n is the sequence length (i.e. the number of tokens per sequence) and d is the model/embedding dimension. A typical batch size is 64–128 sequences. However, for the sake of clarity, the derivations in this section assume a batch size of $b = 1$. This avoids the need to carry an extra leading batch dimension in every tensor, thus keeping the notation readable. All computations can be easily generalised to batched inputs because the operations – matrix multiplications, softmax and concatenations – are applied independently to each element of the batch.

The sequence GTS is used in two different forms. First, a shifted sequence is created by removing the final token. After embedding and positional encoding, this produces $Y_{shifted} \in \mathbb{R}^{(m+1)\times d}$, where d is the model dimension:

$$[id_{BOS},\ id_1, id_2, \dots, id_m] \xrightarrow{embedding + pos.\ encoding} Y_{shifted} = [BOS,\ Y_1, Y_2, \dots, Y_m]^T.$$

The matrix $Y_{shifted}$ serves as the input to the decoder during teacher forcing ([185]). Both BOS and EOS provide the model with semantic and positional signals. The model must learn that the BOS indicates the beginning of a sequence. Tokens should be generated after the BOS signal. Similarly, the model learns to output EOS when appropriate, for example at the end of a sentence in translation or when the answer is complete in dialogue.

Secondly, the ground-truth is used as the prediction targets, which are 'correct next token' at each position (i.e. what the model must output). These targets come from the same ground-truth ID sequence, but shifted left:

$$Y = [\ id_1, id_2, \dots, id_m, id_{EOS}].$$

Unlike $Y_{shifted}$, this sequence is neither embedded nor positionally encoded. Instead, it is compared directly with the decoder's output probabilities (after the final linear-softmax layer) to compute the training loss.

Thus, during training, the decoder receives the pair $(Y_{shifted}, M)$ as its inputs, and the corresponding target token sequence Y is solely used for loss computation.

- **Inference mode**. Training mode takes place offline, with the model being optimised using teacher forcing, backpropagation, and gradient descent on vast amounts of data. When you interact with systems such as ChatGPT or Copilot, the model operates in inference mode. You provide an input prompt, such as a question or a sentence. The Transformer then uses the trained model to generate outputs like translations or text. As the ground truth is not available, the decoder must produce the output sequence itself. In other words, when you ask ChatGPT a question, it is not 'learning' at that moment – it is applying what it learned during training to generate a response.

During inference (i.e. after model deployment), the Transformer generates the output autoregressively (see Section 5.2.3.3), producing one token at a time. Therefore, the data flow differs substantially from the training process. The user provides a source sentence; for example, S_{EN} for an English-to-German translation task. The sentence is tokenised and mapped to embeddings with positional encodings, just as it was during training. If the source sentence consists of n tokens, this yields $X \in \mathbb{R}^{n\times d}$, which is passed through the encoder stack. The encoder output $M \in \mathbb{R}^{n\times d}$ serves as a fixed memory bank for the entire decoding process.

5.2.3.3. Decoder structure

The decoder transforms a partially generated target sequence into the next token distribution, combining information from its previous outputs (via masked self-attention) and from the encoder (via cross-attention). As with the encoder, the decoder comprises a stack of q identical layers, the number of which can range from a few to several hundred, depending on the scale of the model. Each layer contains three sublayers:

- Masked Multi-Head Self-Attention (MMHA)
- Cross-Attention (Encoder–Decoder Attention)
- Position-wise Feed-Forward Network (FFN).

---

[185] Teacher forcing is a training strategy used in Transformer decoders, in which the model is provided with the ground-truth previous target token at every decoding step, rather than its own predicted token. Formally, when predicting the next token y_t the decoder is fed the true sequence $y_{1:t-1}$ from the corpus, not the model's earlier outputs. This stabilises optimisation, accelerates convergence, and ensures consistent gradients during early training.

Each sublayer is wrapped in an AddNorm structure (residual connection + layer normalisation), following the Post-LN formulation ([186])

$$Output = AddNorm(X) = LayerNorm(X + Sublayer(X)).$$

See Section 5.2.3.1 for details – the same construction applies here.

The first layer of the decoder receives as input the matrix $X^{(0)} = Y_{shifted} \in \mathbb{R}^{(m+1)\times d}$, which contains the embedded and positionally encoded target-side sequence, shifted right and beginning with the special token BOS. Each decoder layer s transforms its input $X^{(s-1)}$:

$$X^{(s)} = DecoderLayer^{(s)}\left(X^{(s-1)}, M\right) \in \mathbb{R}^{(m+1)\times d}, \ \ s = 1,2,\dots,q,$$

until the final matrix $X^{(q)}$ is passed to the output classifier (a linear layer + softmax) to produce token probabilities.

- **Masked multi-head self-attention (MMHA)**. The first sublayer allows each target position to attend only to earlier positions within the same target sequence. This is critical because, during the inference process, the model only knows the tokens it has generated so far. For readability, we first present the single-head version. Given an input $X^{(s-1)} \in \mathbb{R}^{(m+1)\times d}$, the decoder computes

$$Q = X^{(s-1)}W_Q, \ K = X^{(s-1)}W_K, \ \ V = X^{(s-1)}W_V,$$

where W_Q, W_Q, $W_Q \in \mathbb{R}^{d\times d}$ are trainable matrices of this sublayer. Note that although these operations are written as matrix multiplications, they are effectively row-wise transformations – see Section 5.2.2.1. The score matrix is computed as

$$S = \frac{QK^T}{\sqrt{d}} \in \mathbb{R}^{(m+1)\times(m+1)}.$$

To enforce the autoregressive constraint, we add the causal mask $M_{causal} = \left[c_{ij}\right]$, where

$$c_{ij} = \begin{cases} -\infty, & i < j \\ 0, & i \geq j \end{cases}, i,j = 1,2,\dots,m+1.$$

Thus, the matrix $M_{causal} \in \mathbb{R}^{(m+1)\times(m+1)}$ has the following form

$$M_{causal} = \begin{bmatrix} 0 & -\infty & -\infty & \cdots & -\infty \\ 0 & 0 & -\infty & \dots & -\infty \\ 0 & 0 & 0 & \dots & -\infty \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}.$$

Why is it this shape? Because the decoder can only attend to positions $\leq i$ (i.e. the past and the current position) and never to positions $> i$ (i.e. the future). This is the *autoregressive constraint*. In practice, $-\infty$ is implemented as a large negative constant, such as -10^9, to ensure that masked scores disappear after the softmax operation.

Consequently,

$$A = softmax(S + M_{causal}) \in \mathbb{R}^{(m+1)\times(m+1)}$$

and

$$Z = Attention(Q,K,V) = AV \in \mathbb{R}^{(m+1)\times d}.$$

In practice, the decoder uses h parallel heads $W_Q^{(l)}, W_K^{(l)}, W_V^{(l)} \in \mathbb{R}^{d\times d_k}, l = 1,2,\dots,h$, $d_k = d/h$. Each head applies the same causal mask, producing $Z^{(l)}$. The outputs are concatenated and linearly projected (see Section 5.2.2.2)

$$MMHA(X^{(s-1)}) = \text{Concat}\left(Z^{(1)}, Z^{(2)}, \dots, Z^{(h)}\right)W_O \in \mathbb{R}^{(m+1)\times d},$$

---

[186] There are two standard variants of the Transformer layer. (1) **Post-LN**: LayerNorm is applied after the residual addition, as described by Vaswani et al. (2017), i.e. $Output = LayerNorm(X + Sublayer(X))$. (2) **Pre-LN** (the version used by GPT-2/3/4/5, LLaMA, PaLM, etc.): LayerNorm is applied before the sublayer, i.e. $Output = X + Sublayer(LayerNorm(X))$.

where $W_O \in \mathbb{R}^{d\times d}$. The matrices $W_Q^{(l)}, W_K^{(l)}, W_V^{(l)}$ and W_O are trained, one set per decoder layer.

In summary, masked self-attention enables each position to generate a context-aware representation of the portion of the target sequence that has already been generated. During training, masking enforces autoregressive generation by suppressing attention to future tokens. During inference, generation proceeds strictly from left to right, meaning that future tokens do not yet exist. Therefore, causality is inherent in the decoding procedure rather than being imposed by masking.

The output

$$X_{AN(MMHA)}^{(s-1)} = AddNorm\big(MMHA(X^{(s-1)})\big) \in \mathbb{R}^{(m+1)\times d}$$

is then used as the input for the next sublayer.

- **Cross-attention (encoder–decoder attention)**. Once the masked self-attention sublayer has produced the contextualised decoder states $X_{AN(MMHA)}^{(s-1)}$, the decoder must incorporate information from the source sequence. This is achieved through cross-attention, whereby each target position queries the encoder's output representations.

Cross-attention uses the same multi-head attention mechanism introduced in Section 5.2.2.2. Each head has its own set of learned projection matrices

$$U_Q^{(l)}, U_K^{(l)}, U_V^{(l)} \in \mathbb{R}^{d\times d_k}, l = 1,2,\dots,h \text{ and } d_k = d/h.$$

The queries come from the decoder

$$Q^{(l)} = X_{AN(MMHA)}^{(s-1)} U_Q^{(l)} \in \mathbb{R}^{(m+1)\times d_k},$$

while the keys and values come from the encoder output matrix $M \in \mathbb{R}^{n\times \mathrm{d}}$

$$K^{(l)} = MU_K^{(l)} \in \mathbb{R}^{n\times d_k},\ V^{(l)} = MU_V^{(l)} \in \mathbb{R}^{n\times d_k}.$$

Therefore, decoder tokens attend to encoder tokens rather than themselves. The outputs $Z^{(l)}$ of all the heads are concatenated $Z = Concat\big(Z^{(1)},\dots,Z^{(h)}\big) \in \mathbb{R}^{(m+1)\times h\cdot d_k}$ and then projected back into the model dimension (see Section 5.2.2.2 for details)

$$CrossAttn\big(X^{(s-1)}, M\big) = ZU_O \in \mathbb{R}^{(m+1)\times d},\ U_O \in \mathbb{R}^{h\cdot d_k\times d}.$$

This is followed by AddNorm, yielding the input for the next sublayer

$$X_{cross}^{(s-1)} = AddNorm\left(CrossAttn\big(X^{(s-1)}, M\big)\right) \in \mathbb{R}^{(m+1)\times d}.$$

The trainable parameters of the cross-attention sublayer are $\left\{U_Q^{(l)}, U_K^{(l)}, U_V^{(l)}\right\}_{l=1}^{h}$ and U_O.

Cross-attention serves as a dynamic alignment mechanism, enabling the decoder to select different source-side information for each target position and generation step.

Remark. The reader may wonder how the encoder's output matrix M, which is derived from the German source sentence, for example, can be useful when the decoder is generating, say, an English translation. After all, M appears to capture German syntax, semantics and word order, so how can an English-language decoder benefit from attending to it?

The key point is that the encoder does not retain the German sentence in its surface linguistic form. Instead, it transforms the entire input into a language-neutral, contextual representation of meaning. Each vector in M encodes not only the local properties of a German word, but also its role in the broader semantic and syntactic structure of the sentence. During decoding, the English-language decoder does not 'read German'; it queries this semantic framework. At each step, the decoder forms a query vector that reflects the partial English translation produced thus far. It then uses cross-attention to retrieve the exact source-side information that is relevant for selecting the next English token. In this way, cross-attention acts

as a dynamic alignment mechanism: the decoder continually asks, "*Given what I have translated so far, which part of the source meaning should I attend to next*?". The encoder's matrix M provides a stable semantic landscape, which the decoder navigates token by token in order to construct a coherent and faithful English translation.

- **Position-wise Feed-Forward Network (FFN)**. This is the final sublayer of each decoder layer, which is identical to that of the encoder (see Section 5.2.3.1). A two-layer fully connected network is applied to each position independently

$$FFN\left(X_{cross}^{(s-1,i)}\right) = W_2 f\left(W_1 \cdot \left(X_{cross}^{(s-1,i)}\right)^T + B_1\right) + B_2 \in \mathbb{R}^d,$$

where $W_1 \in \mathbb{R}^{d_{ff} \times d}$, $W_2 \in \mathbb{R}^{d \times d_{ff}}$, and $B_1 \in \mathbb{R}^{d_{ff}}$, $B_2 \in \mathbb{R}^d$ are bias vectors. All of these objects are trainable parameters of this FFN sublayer. The activation function f is $ReLU$ (or $GELU$). The output is wrapped in a second AddNorm, which provides the input for the next layer

$$X^{(s)} = AddNorm\left(FFN\left(X_{cross}^{(s-1)}\right)\right) = LayerNorm\left(X_{cross}^{(s-1)} + FFN\left(X_{cross}^{(s-1)}\right)\right) \in \mathbb{R}^{(m+1)\times d}.$$

Once the decoder has processed its input through all q layers, it produces the final hidden-state matrix $X = X^{(q)} \in \mathbb{R}^{(m+1)\times d}$, where each row X_i of X represents the model's internal contextual representation at target position i. To generate a predicted token, this representation must be converted into a probability distribution over the vocabulary. The exact procedure depends on the operational mode. To avoid duplication, we first describe the shared final transformation before splitting into two subsections (training and inference).

- **Final linear layer and softmax (shared across both modes)**. The final linear layer maps the contextual hidden state of the decoder into vocabulary space. Softmax then converts these scores into a probability distribution for the next word.

Each final hidden state

$$X_i = [x_{i1}, x_{i2}, \dots, x_{id}] \in \mathbb{R}^d, i = 1,2, \dots, m+1$$

is mapped to vocabulary ([187]) logits ([188]) by a learned transformation

$$Lo_i = [lo_{i1}, lo_{i2}, \dots, lo_{iv}] = X_i W_{out} + B_{out},$$

where $W_{out} \in \mathbb{R}^{d \times v}$, $B_{out} \in \mathbb{R}^v$, and v is the vocabulary size ([189]). $Lo_i \in \mathbb{R}^v$ is the logits vector at position i. The logit lo_{ij} is assigned to token ID j, where $j \in \{1, 2, \dots, v\}$. The logits are normalised with a softmax, giving a categorical distribution over token IDs

$$p(y_i = j \mid X_i) = \frac{exp(lo_{ij})}{\sum_{k=1}^{v} exp(lo_{ik})}.$$

---

[187] In sequence-to-sequence translation models (e.g., Transformer-based neural machine translation (NMT)), the encoder and decoder typically have different vocabularies, each tailored to the language being processed. This means that the decoder's vocabulary corresponds to the target language (e.g. German), while the encoder's vocabulary corresponds to the source language (e.g. English). However, multilingual or dual-language systems (e.g. mBERT and mT5) often use a shared subword vocabulary for many languages. This is possible because subword units often overlap (e.g. 'in', 'tion', 'en', 'die'). Consequently, a single WordPiece/BPE vocabulary can efficiently cover multiple languages. It improves parameter sharing across languages.

[188] Logits are raw, unnormalised scores output by the model before softmax. They may take any real value; only after softmax they become probabilities.

[189] The size of the vocabulary in Transformer-based natural NLP models depends on the tokenisation method, the domain and the model family. Most modern Transformer models use subword tokenisation to enable compact yet expressive vocabularies. Typical sizes are: BERT: 30,000 tokens (WordPiece); GPT-5: 50,000 tokens (Byte-level BPE); PaLM: 32,000 tokens (SentencePiece). A Transformer model's vocabulary is the set of all atomic units (tokens) that the tokeniser can produce. Recall (see Section 5.1.2) that a 'token' does not have to be a word; it can be a subword fragment ('cat', '##title'), a symbol, a punctuation mark, a whitespace character, or even a raw byte (in byte-level BPE). Therefore, the choice of tokeniser defines the vocabulary and directly influences sequence length, model efficiency and representational granularity.

Here, '$y_i = j$ ' means that the token at position i is the vocabulary item with ID j. The X_i condition means that the probability p depends on all the information encoded in the decoder state X_i, i.e. the source sentence (via cross-attention) and the prefix already generated (via masked self-attention).

In summary, the output of this layer is a probability matrix

$$P = softmax(Lo) = [p_{ij}] \in \mathbb{R}^{(m+1)\times v},$$

where $Lo = XW + B$ and $p_{ij} = p(y_i = j \mid X_i)$.

5.2.3.4. Decoder operation during training

Given the decoder architecture defined above, we now specify its training-time operation. While the parameterisation and layer structure remain fixed, the conditioning of decoder states on the target sequence and the resulting information flow differ from those in the inference regime. During training, the decoder operates in parallel mode using the entire ground-truth target sequence from the corpus. Let the target sentence be tokenised into IDs

$$[BOS,\ y_1, y_2, \dots, y_m, EOS],$$

where y_i is an integer token ID in the set $\{1, 2, \dots, v\}$. These IDs are not embedded; they are used directly as labels in the loss function. The decoder uses the right-shifted input

$$[BOS,\ y_1, y_2, \dots, y_m].$$

The goal is to predict the ground-truth ID at each position

$$Y_{target} = [y_1, y_2, \dots, y_m, EOS].$$

Training uses teacher forcing, meaning the decoder is given the correct previous target tokens rather than its own predictions. The model learns to assign high probability to the correct next token at every position in the sequence. That is, at position i, the input always contains

$$[BOS,\ y_1, y_2, \dots, y_{i-1}]$$

not the model's earlier predicted IDs $\hat{y}_1, \hat{y}_1, \dots, \hat{y}_{i-1}$ ([190]). These IDs are not used in training. Instead, the loss function (cross-entropy) compares the predicted distribution with the correct ID y_i

$$L_i = -log\big(p(y_i \mid X_i)\big).$$

Thus, the model learns to assign high probability to the correct next token at every position. For example: the model predicts $\hat{y}_1$ from BOS, then $\hat{y}_2$ from $[BOS,\ y_1]$, and finally predicts EOS from $[BOS,\ y_1, y_2, \dots, y_m]$.

In other words, the objective of the training process is to ensure that the model assigns a higher probability to the correct next token. Gradients, backpropagation, attention weights and embeddings are all just the machinery used to achieve this. Formally, training aims to minimise the average loss.

$$L = \frac{1}{m+1}\sum_{i=1}^{m+1} L_i = -\frac{1}{m+1}\sum_{i=1}^{m+1} log\big(p(y_i \mid X_i)\big).$$

This is equivalent to maximising the likelihood of the correct translation, i.e. producing more accurate and confident predictions from the model.

What is the intuitive meaning of loss values? Each loss value L_i indicates how 'surprised' the model is by the correct next token. If the model assigns a high probability to the correct token

---

[190] What is the predicted token $\hat{y}_1$? Given the probability distribution vector $P_i = softmax(Lo_i)$, the predicted token ID is obtained by selecting one of the vocabulary IDs using a decoding rule. One example is the argmax decoding rule (also known as *greedy prediction*), which chooses the token ID with the highest predicted probability. However, during training, $\hat{y}_1$ is ignored.

(i.e. if L_i is small), the model 'expected' the right answer. Conversely, if the model assigns a low probability to the correct token (i.e. if L_i is large), the model was 'surprised' or 'confused'. Therefore, a loss of 0.1 means that the model was almost certain and correct. A loss of 2.3 means the model 'thought' the correct token was only around 10% likely, while a loss of ≈ 5.0 means the model had no idea. A perfect model would assign probability 1.0 to every correct token

$$p(y_i \mid X_i) = 1 \Rightarrow L_i = -log(1) = 0.$$

The total loss is the average of these per-token losses and therefore reflects the model's overall 'surprise' at the correct translation.

In summary, the loss function indicates how inaccurate the model is. Minimizing the total loss encourages the model to assign higher probability to the correct tokens across the whole sequence, gradually improving its ability to generate coherent and accurate translations.

Example. This example shows how computations are carried out in parallel during training, and how teacher-forced prediction operates. Consider a translation task from English to German, and assume that a training pair (S_{EN}, S_{DE}) consists of the following sentences:

S_{EN}: "The cat sat on the mat."

S_{DE}: "Die Katze saß auf der Matte."

We assume the encoder has already processed the English sentence and its output matrix $M \in \mathbb{R}^{7\times4}$ is

$$M = \begin{bmatrix} 1.0 & 0.0 & 0.5 & 0.0 \\ 0.8 & 0.2 & 0.4 & 0.1 \\ 0.7 & 0.1 & 0.6 & 0.0 \\ 0.6 & 0.0 & 0.3 & 0.2 \\ 0.9 & 0.1 & 0.5 & 0.1 \\ 1.0 & 0.1 & 0.7 & 0.0 \\ 0.2 & 0.0 & 0.0 & 0.0 \end{bmatrix} \in \mathbb{R}^{7\times4}.$$

The rows of matrix M contain the embeddings and positional encodings of the tokens [the], [cat], ..., [.], respectively. This matrix is used to compute the keys and values in the decoder's cross-attention mechanism.

We define a toy target vocabulary of size $v = 9$:

Token	Assigned *ID*
[BOS]	1
[Die]	2
[Katze]	3
[saß]	4
[auf]	5
[der]	6
[Matte]	7
[.]	8
[EOS]	9

The shifted decoder input is

$$Y_{shifted} = [BOS, 'Die', 'Katze', ' saß', ' auf', ' der', ' Matte', ' .'].$$

The German target sentence is

$$Y_{target} = ['Die', 'Katze', ' saß', ' auf', ' der', ' Matte', ' .', EOS].$$

- *Decoder input*. Assume that, after embeddings and positional encoding of $Y_{shifted}$, we get

$$X^{(0)} = \begin{bmatrix} 0.50 & 1.10 & 0.00 & 1.20 \\ 0.45 & 0.15 & 1.05 & 0.25 \\ 0.40 & 0.20 & 0.10 & 0.40 \\ 0.35 & 1.12 & 1.08 & 0.28 \\ 0.30 & 1.10 & 0.05 & 0.20 \\ 0.25 & 0.08 & 1.05 & 0.15 \\ 0.20 & 1.05 & 1.02 & 0.10 \\ 1.05 & 0.02 & 0.01 & 1.03 \end{bmatrix} \in \mathbb{R}^{8\times 4}.$$

These embeddings are fed into the decoder's masked self-attention. After computing $Q = X^{(0)}\mathrm{W}_Q, K = X^{(0)}\mathrm{W}_K$ and $V = X^{(0)}\mathrm{W}_V$ we obtain the matrix ([191])

$$X_{MMHA} = softmax\left(\frac{QK^T}{\sqrt{4}} + M_{causal}\right)V,$$

where $M_{causal} = \left[m_{ij}\right]_{i,j=1}^{8}$ is the causal mask: for row i the entries $j > i$ are set to 10^{-9}. The remaining entries are set to 0.

Through masked self-attention, the ith row of the matrix X_{MMHA} encodes the structure and meaning of the prefix, i.e. of the tokens corresponding to the ground-truth IDs $y_0, \ldots, y_{i-1}$, $i = 1, 2, \ldots, 8$. More precisely, the row i of X_{MMHA} is a vector representation resulting from the embedding of the current token and masked self-attention over all previous tokens. Note that y_0 has no prefix to encode. Instead, it only encodes the BOS token, which is the start-of-sentence token. Here, we can see how the teacher-forcing method works. Each decoder state $(X_{MMHA})_i$, encodes the ground-truth prefix up to position $i-1$. This ensures that the model learns to predict the next token based on the correct prefix, rather than on its own, potentially incorrect, earlier outputs.

- *Parallel cross-attention (all positions at once)*. The queries come from the decoder, while the keys and values come from the encoder output matrix M

$$Q_{cross} = X_{MMHA}U_Q, \quad K_{cross} = MU_K, \quad V_{cross} = MU_V.$$

The next step is to compute cross-attention outputs

$$X_{cross} = AV_{cross} = softmax\left(\frac{Q_{cross}K_{cross}^T}{\sqrt{4}}\right)V_{cross}.$$

- *FFN*. Assume that we have obtained the following matrix of hidden states

$$X_{FFN} = FFN(X_{cross}) = W_2 ReLU(W_1(X_{cross})^T) = \begin{bmatrix} 0.3 & 1.0 & -0.5 & 0.2 \\ -0.2 & 0.1 & 1.2 & 0.0 \\ 0.8 & -0.1 & 0.3 & 0.5 \\ 0.0 & 0.4 & 0.2 & 1.0 \\ 0.5 & 0.2 & 0.0 & -0.3 \\ -0.1 & 0.7 & 0.4 & 0.1 \\ 0.2 & -0.4 & 0.6 & 0.3 \\ 0.0 & 0.0 & 0.0 & 0.8 \end{bmatrix}.$$

Each row i of X_{FFN} is a vector representation resulting from:

- the embedding + positional encoding of the current input token i;
- masked self-attention over all previous tokens;
- cross-attention to the encoder; and
- the feed-forward network.

- *Output logits and probability distribution.* Assume that the resulting full probability matrix is

[191] To keep the example simple, we will not write out all the parameter matrices or include the detailed matrix multiplications.

$$P = softmax(X_{FNN}W_{out}) =$$

$$\begin{bmatrix} 0.124 & 0.250 & 0.056 & 0.112 & 0.092 & 0.092 & 0.092 & 0.092 & 0.092 \\ 0.073 & 0.098 & 0.295 & 0.089 & 0.089 & 0.089 & 0.089 & 0.089 & 0.089 \\ 0.200 & 0.081 & 0.121 & 0.148 & 0.090 & 0.090 & 0.090 & 0.090 & 0.090 \\ 0.088 & 0.132 & 0.108 & 0.238 & 0.088 & 0.088 & 0.088 & 0.088 & 0.088 \\ 0.172 & 0.141 & 0.115 & 0.085 & 0.104 & 0.104 & 0.104 & 0.104 & 0.104 \\ 0.086 & 0.191 & 0.142 & 0.105 & 0.095 & 0.095 & 0.095 & 0.095 & 0.095 \\ 0.121 & 0.067 & 0.181 & 0.134 & 0.099 & 0.099 & 0.099 & 0.099 & 0.099 \\ 0.098 & 0.098 & 0.098 & 0.218 & 0.098 & 0.098 & 0.098 & 0.098 & 0.098 \end{bmatrix}.$$

The maximum probability value in each row is highlighted in blue.

- *Target and loss values*

Position *i*	Target	Probability	Loss$_i$
1	Die	0.250	1.39
2	Katze	0.295	1.22
3	saß	0.148	1.91
4	auf	0.088	2.43
5	der	0.104	2.26
6	Matte	0.095	2.35
7	.	0.099	2.31
8	*EOS*	0.098	2.32
		Total loss	2.03

- *Summary*. In practice, a good model might produce a value of L_i between 0.1 and 0.5 per token. A mediocre model might achieve 1.0 - 2.0, whereas a random model would get around 2.2 - 2.4 (because $-log(1/9)$ is approximately 2.2). Thus, the total loss of 2.03 indicates that the model performs only slightly better than random guessing. This is precisely what we would expect from a model with untrained or randomly initialised parameters: it has not yet learned any meaningful patterns in the data, so it is 'surprised' by the correct translation almost every time.

Assuming that the same probability distributions applied to the inference mode and that we selected the token with the highest probability at each step, the model would produce the following output sequence

"Die Katze *BOS* saß *BOS* die Katze saß".

This would not be a correct translation, of course. This sequence is not grammatically correct, either. However, we have used the probability distributions obtained in training mode. In practice, the distributions would change after each predicted token in the inference mode – see the next section for details.

Please note that there is no end-of-sentence (*EOS*) among the tokens with the maximum probability, so the model would never terminate cleanly; in practice, it would stop at the maximum length.

5.2.3.5. Decoder operation during inference

Unlike the training process, the decoder operates under a causal, autoregressive constraint during inference. At each decoding step, the model only considers previously generated tokens, and representations are computed sequentially rather than in parallel. This change in information availability fundamentally alters the flow of computation in the self-attention mechanism, despite the underlying architecture and parameters remaining unchanged.

In inference mode, the target sentence is unavailable. Instead, the decoder must generate the output sequence one token at a time using only the encoder output $M \in \mathbb{R}^{n \times d}$, which is computed from the user-provided source sequence, and the tokens already generated by the

decoder itself. This procedure is autoregressive, meaning the model predicts $ID = \hat{y}_i$ of the next token based solely on previously generated token IDs

$$[\hat{y}_1, \hat{y}_1, \dots, \hat{y}_{i-1}].$$

Generation always begins with the special BOS token. The decoder embeds this token, adds positional encodings and processes it through a stack of decoder layers, yielding vector X_1. After processing BOS, the decoder produces a probability distribution over the vocabulary

$$p(y_1 = j \mid X_1) = softmax(lo_{1j}).$$

The model then selects a token $ID = \hat{y}_1$, using a decoding strategy such as argmax (*greedy decoding*). The token $w_{\hat{y}_1}$ from the vocabulary with $ID = \hat{y}_1$, is then appended to the BOS token: $[BOS, w_{\hat{y}_1}]$. This sequence becomes the decoder's input at the next step.

At each subsequent position i the decoder receives as input the embeddings of

$$[BOS, w_{\hat{y}_1}, \dots, w_{\hat{y}_{i-1}}].$$

The decoder then computes masked self-attention using only the prefix generated so far, as well as cross-attention to the encoder memory M, and feed-forward transformation and AddNorm layers. This produces the hidden state X_i, which is then used for computing the vocabulary distribution $p(y_i = j \mid X_i)$. The predicted token ID is $\hat{y}_i = Decode(p(\cdot \mid X_i))$, where $Decode$ stands for the selected decoding strategy. The corresponding token $w_{\hat{y}_i}$ from the vocabulary is then appended

$$[BOS, w_{\hat{y}_1}, \dots, w_{\hat{y}_{i-1}}, w_{\hat{y}_i}]$$

and this sequence becomes the decoder's input at the next step.

The generation loop continues until one of the following conditions is met:

- the model emits EOS, which signals the end of the sequence,
- a maximum length limit is reached,
- an external constraint from the application is satisfied (e.g. stopping after generating a full stop '.').

The final output of the decoder is a discrete sequence

$$\hat{Y} = [\hat{y}_1, \hat{y}_1, \dots, \hat{y}_T],$$

where each $\hat{y}_i \in \{1, 2, \dots, v\}$ denotes an index in the target vocabulary. This can be mapped back to human-readable text (e.g. a translated sentence) via the vocabulary, which may include detokenisation or subword merging procedures..

In summary, inference does not use ground-truth tokens since these are unknown. Therefore, unlike in training mode, it cannot use teacher forcing either, meaning it must rely entirely on its own previously generated outputs, which can introduce errors. Furthermore, inference cannot use parallel processing over the target sequence, so tokens must be produced sequentially.

To make these differences concrete, we now examine the internal mechanics of autoregressive decoding at the level of individual decoder layers. Rather than describing inference only in terms of input-output behaviour, the following account traces how queries, keys, and values are computed, cached, and reused across successive decoding steps. This explicit treatment clarifies how masked self-attention is implemented in practice during inference and how the self-attention KV cache enables efficient conditioning on the entire generated prefix.

During inference, the model only processes the current position at each step, while information from all previous positions is accessed exclusively through the key-value (KV) caches inside each decoder layer. Crucially, the decoder does not store the full hidden states of

earlier positions, but rather only their projected keys and values, which contain precisely the information required for attention. The following description outlines the inference process in detail.

- **Step 1: processing the first token**. As we know, the process of inference begins with the special start-of-sequence token BOS. This token is embedded and positionally encoded before being passed as a vector $X_0 \in \mathbb{R}^{1\times d}$ into the first decoder layer. In the masked self-attention sublayer of this layer, the vectors

$$Q_0 = X_0 W_Q, \quad K_0 = X_0 W_K, \quad V_0 = X_0 W_K$$

are computed ([192]). The vectors K_0 and V_0 are stored in the KV cache of this layer. Since only one position exists, the attention score matrix $S = \frac{Q_0 (K_0)^T}{\sqrt{d}}$ consists of a single entry. After passing through all decoder layers, the final hidden state is mapped to vocabulary logits to predict the next token w_1.

- **Step 2: processing the second token**. The predicted token w_1 is embedded and positionally encoded before being passed as $X_1 \in \mathbb{R}^{1\times d}$ into the first decoder layer. In the masked self-attention sublayer, new projected vectors Q_1, K_1 and V_1 are computed from X_1. The previous hidden state X_0 is not reused. The vectors K_1 and V_1 are appended to the layer's KV cache. Attention is computed by combining

- the current query Q_1,
- the cached keys K_0, K_1,
- the cached values V_0, V_1:

$$Attention(Q_1, K, V) = softmax\left(\frac{Q_1 (K)^T}{\sqrt{d}}\right) V \in \mathbb{R}^{1\times d},$$

where $K \in \mathbb{R}^{2\times d}$ is a matrix with rows K_0 and K_1, while $V \in \mathbb{R}^{2\times d}$ is a matrix with rows V_0 and V_1. Note that the score matrix $S = \frac{Q_1 (K)^T}{\sqrt{d}}$ consists of a single row containing two entries. The previous step's query Q_0 plays no role because queries depend on the hidden state of the current position and are never cached.

Once all decoder layers have processed X_1, the resulting hidden state is mapped to logits in order to predict the next token w_2.

- **Step 3 and beyond: continuing autoregressive generation**. At step i ($i \geq 3$), the model processes the newly generated token w_{i-1} in the same way. First, w_{i-1} is embedded and positionally encoded to obtain X_{i-1}. Then, in each decoder layer:

  - Q_{i-1}, K_{i-1} and V_{i-1} are computed from X_{i-1}.
  - K_{i-1} and V_{i-1} are appended to the KV cache of that layer.
  - Attention is computed using the current query Q_{i-1}, all cached keys $K_0, K_1, \dots, K_{i-1}$, and all cached values $V_0, V_1, \dots, V_{i-1}$.

The resulting hidden state is then passed through all layers to obtain the final hidden state, which is mapped to logits in order to predict w_i.

This process continues until the model emits the end-of-sequence token EOS.

---

[192] In inference mode, the Transformer decoder's masked self-attention effectively becomes causal self-attention over the prefix generated so far. Masking becomes redundant because autoregressive decoding ensures that future tokens are never present. Therefore, attention is computed solely over past tokens. In this sense, masking does not affect computation during inference; it is implicitly satisfied by the autoregressive generation process. Nevertheless, the mask remains part of the model's definition to ensure that the same attention operator is valid during training.

- **Cross-attention during inference**. In encoder-decoder Transformers, the keys and values for the decoder's cross-attention sublayers are not generated during autoregressive decoding. Instead, they are computed once immediately after the encoder has processed the output matrix M. Each decoder layer then projects M into cross-attention keys and values

$$K_{cross} = MU_K \in \mathbb{R}^{n\times d}, V_{cross} = MU_V \in \mathbb{R}^{n\times d},$$

where U_K and U_V are trainable $d \times d$ matrices of the cross-attention sublayer (see Section 5.2.3.3). The matrices K_{cross} and V_{cross} are then stored in a dedicated cross-attention $(KV)_{cross}$ cache for each decoder layer. Crucially, this cache remains unchanged throughout the entire inference process: the encoder output does not change, and no new, encoder-derived keys or values are ever added when decoding.

The following mathematical explanation illustrates how the decoder's query interacts with the encoder's keys in order to identify the source-side information that is most relevant for producing the next target token. For simplicity, we will consider one-head attention with $d_k = d$.

During autoregressive inference, the decoder generates one target token at a time. At decoding step i, the decoder has already produced the prefix $w_0, w_1, \dots, w_{i-1}$, where $w_0 = BOS$. Assume that the current decoder input vector $X_{i-1} \in \mathbb{R}^d$ corresponds to the most recently generated token w_{i-1}. For a given cross-attention layer, let $Z_{i-1} \in \mathbb{R}^d$ be the output vector from the immediately preceding masked self-attention sublayer. The decoder computes a query vector from Z_{i-1}

$$Q_{i-1} = Z_{i-1}U_Q \in \mathbb{R}^d.$$

Note that, although the decoder only processes the current token, the self-attention mechanism incorporates the entire prefix into the current hidden state Z_{i-1}, via the KV cache of the self-attention sublayer. Consequently, cross-attention uses this prefix-aware state to extract the relevant semantic information from the encoder's representation. In other words, the query Q_{i-1} expresses „*what the decoder currently needs* “ from the source sequence, conditioned on the translation prefix $w_0, w_1, \dots, w_{i-1}$.

In order to retrieve the relevant source information, the decoder compares the current query with all the encoder keys

$$s_{(i-1)j} = \frac{Q_{i-1}\cdot K_j}{\sqrt{d}}, j = 1,2, \dots, n,$$

where K_j is the jth row of the matrix K_{cross}. These alignment scores indicate which parts of the source sentence are most relevant for producing the next token, given the translation prefix encoded in Z_{i-1}. Now, the alignment scores are normalised using a softmax

$$a_{(i-1)j} = softmax\big(s_{(i-1)j}\big), j = 1,2, \dots, n.$$

This yields a probability distribution over encoder positions, indicating where the decoder should 'look' in the source sentence at step i. Finally, the decoder computes a weighted sum of the value vectors

$$Attention(Q_{i-1}, K_{cross}, V_{cross}) = softmax\left(\frac{Q_{i-1}(K_{cross})^T}{\sqrt{d}}\right)V_{cross} \in \mathbb{R}^{1\times d}.$$

The resulting vector is a distilled semantic representation extracted from the encoder and tailored to the decoder's current generative context. This vector is then fed into subsequent layers to produce the next English token.

Having outlined the step-by-step process of autoregressive inference, we will now examine several structural features of the decoder that explain why this procedure works. These include the role of self-atention KV caches, the necessity of backwards-looking self-attention, and the importance of accessing all previously stored value vectors.

- ***KV* caches in autoregressive decoding**. A key feature that enables efficient autoregressive decoding is the system of KV caches maintained within each decoder layer. These caches store

the key-value representations of all previously processed positions, thereby eliminating the need to recompute self-attention over the entire prefix at every step. Recall that each decoder layer has two KV caches: one stores the keys and values produced by the masked self-attention sublayer and the other stores the fixed, encoder-derived keys and values used by the cross-attention sublayer.

- **Why self-attention must look backwards**. The decoder's self-attention mechanism must always look backwards across the entire generated prefix because predicting the next token requires information from all earlier positions, not just the most recent one.

When computing the self-attention scores at position $i \geq 1$, the query vector Q_{i-1} is not only compared with the key K_{i-1} of the current position, but also with all the keys $K_0, K_1, \dots K_{i-2}$ stored in the KV caches of the decoder layers. This design is essential for the Transformer's expressive power. The next token w_i, must be predicted based on the entire prefix, $w_0, w_1, \dots, w_{i-1}$ rather than merely the most recent token. Each key K_j encodes the information that token j 'offers' to future positions, while the query Q_{i-1} expresses the information required at the current position. By comparing Q_{i-1} with all past keys, the model retrieves precisely the value vectors V_{i-1} that are relevant for the current prediction. Restricting attention to K_{i-1} alone would collapse the conditioning context to the immediately preceding token, effectively approximating a bigram language model in which predictions depend only on local token-to-token transitions. Such a restriction would severely limit the model's ability to capture long-range dependencies, resolve references, and maintain semantic and syntactic coherence, rendering it similar to a bigram-like predictor. Attention therefore acts as a content-based retrieval mechanism over the entire prefix, and the KV cache provides the persistent memory that makes this retrieval possible.

- **Why all past values are required**. As each value vector encodes the contextualised semantic contribution of its corresponding token, the decoder must retain and access all past values during the attention process to retrieve the precise information needed for next-token prediction that is both semantically coherent and context-sensitive.

In masked self-attention, the query vector Q_{i-1} is compared with all keys $K_0, K_1, \dots K_{i-1}$ to determine which past positions are relevant for the current prediction. Once these relevance weights have been computed, the model retrieves information from the corresponding value vectors $V_0, V_1, \dots V_{i-1}$. The use of all values is essential: each V_j contains the contextualized semantic content of token j, including its meaning, syntactic role, and contribution to the discourse. Keys merely indicate where relevant information may be found; values contain what must be retrieved. Restricting the model to V_{i-1} alone would force it to behave like a bigram model, unable to recall long-range dependencies, resolve references, or maintain coherence across the sequence. The KV cache therefore serves as a distributed memory of the entire prefix, and attention provides a content-based retrieval mechanism that allows the model to extract precisely the information needed at each step of autoregressive generation.

Example. This example demonstrates how the decoder operates in inference mode, generating tokens autoregressively. Consider the translation task again from English to German of the following sentence:

S_{EN}: "The cat sat on the mat."

Let us assume the same settings as in the previous example. Thus, we will use the same encoder output M, but this time the decoder will run in inference mode, feeding back its own predictions. To keep the example simple, we will again not write out all the parameter matrices or include detailed matrix multiplications.

- *Inference step 1.* The only input to the decoder is the token BOS, which is embedded and positionally encoded. Assume that the decoder produces the following hidden state

$$X_{FFN} = [0.1 \quad 1.5 \quad 0.3 \quad -0.2].$$

Applying the softmax function gives us

$$P = [0.087 \quad 0.351 \quad 0.106 \quad 0.064 \quad 0.078 \quad 0.078 \quad 0.078 \quad 0.078 \quad 0.078].$$

Since $ID = 2$ has the highest probability, the prediction is $\hat{y}_1 = 2$ ('Die').

- *Inference step 2*. The decoder input becomes

$$[BOS, \text{'Die'}].$$

Both tokens are embedded and positionally encoded to produce a 2×4 matrix, which is then fed into the decoder. Assume that after processing, the second step yields the following probability distribution

$$P = [0.076 \quad 0{,}093 \quad 0.379 \quad 0.069 \quad 0.076 \quad 0.076 \quad 0.076 \quad 0.076 \quad 0.000].$$

Consequently, the prediction is $\hat{y}_2 = 3$ ('Katze').

- *Inference step 3*. The decoder input becomes

$$[BOS, \text{'Die'}, \text{'Katze'}].$$

After embedding and positional encoding, we get a 3×4 matrix, which is then fed into the decoder. Assume that the third step yields the following probability distribution

$$P = [0.066 \quad 0.090 \quad 0.109 \quad 0.329 \quad 0.081 \quad 0.081 \quad 0.081 \quad 0.081 \quad 0.081].$$

Consequently, the prediction is $\hat{y}_3 = 4$ ('saß').

- *Inference step 4*. The decoder input becomes

$$[BOS, \text{'Die'}, \text{'Katze'}, \text{'saß'}].$$

After embedding and positional encoding, a 4×4 matrix is produced and fed into the decoder. Assume that this step generates the following probability distribution

$$P = [0.075 \quad 0.092 \quad 0.102 \quad 0.112 \quad 0.124 \quad 0.124 \quad 0.124 \quad 0.124 \quad 0.124].$$

Columns 5–9 are all tied at 0.124 thus by convention we pick $\hat{y}_4 = 9$ (EOS).

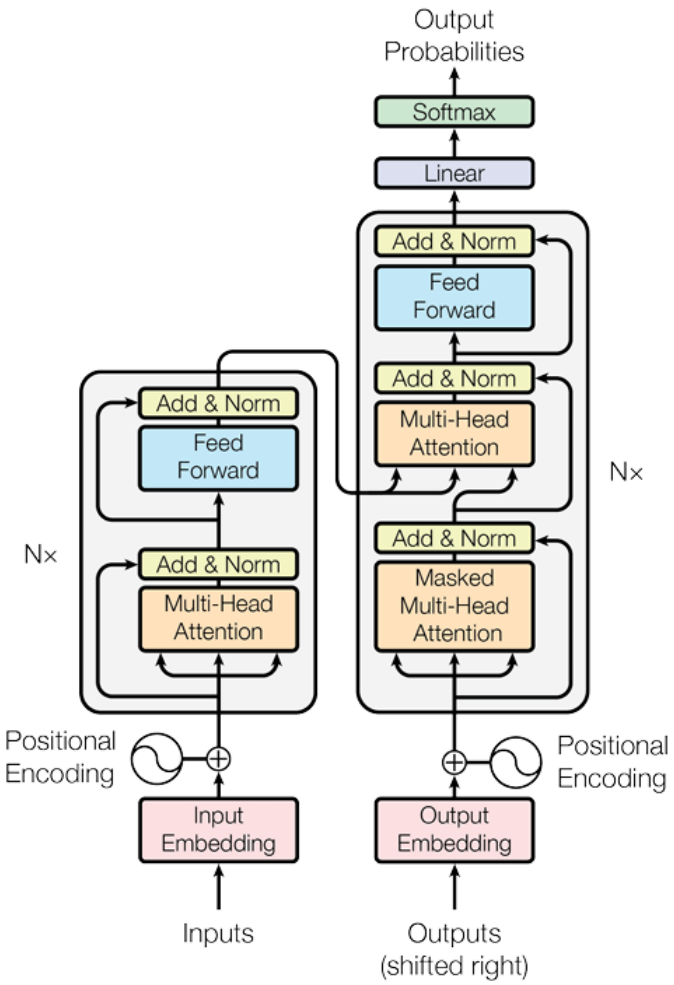
Once EOS is generated, inference terminates. The final output is therefore

"Die Katze saß EOS".

In summary, during inference, the decoder generates the translation one token at a time, using its own previous predictions as input each time. This distinguishes inference from training, in which the decoder receives the correct previous token (teacher forcing). As the model in this example is not trained, its predictions reflect weak or random preferences rather than true translation ability. Nevertheless, the inference mechanism itself is exactly the same as in a real Transformer model.

Having examined the encoder and decoder individually, the overall structure of the Transformer becomes easier to appreciate, as shown in the following figure ([193]).

[193] The original Transformer architecture figure is reproduced under the Creative Commons CC-BY 4.0 license from Vaswani et al. (2017) ([V2]).



Remark. Early sequence models, such as RNNs, LSTMs and GRUs, implemented information mixing and nonlinear transformation within a single recurrent update. However, this coupling imposed strict sequential dependencies and limited parallelism, thereby constraining the model's ability to scale to long sequences. CNN-based architectures partially decoupled these roles, but their mixing operations were still restricted to fixed-size local receptive fields. This meant that deep stacks of layers were required to approximate global context. The Transformer, however, introduced a clean architectural factorisation: self-attention performs global, content-dependent mixing, while the position-wise feed-forward network (FFN) performs non-linear transformation independently at each position. This separation proved critical for two reasons. Firstly, it enabled full parallelisation across sequence positions, eliminating the recurrence bottleneck and enabling training on much larger datasets. Secondly, it enabled the model to be scaled in both depth and width while maintaining stable optimisation dynamics since attention and nonlinearity are handled by distinct, modular components. This architectural decoupling is a key factor in why Transformers exhibit predictable scaling laws and remain the basis of modern, large-scale language models.

5.2.4. How large language models learn

Large Language Models (LLMs) are not programmed with explicit linguistic rules, grammatical formalisms or symbolic representations of meaning. Instead, they acquire their capabilities through a data-driven learning process that optimises the straightforward yet powerful objective of predicting tokens from context. Despite this objective's apparent simplicity, repeatedly applying it to massive datasets and highly expressive model architectures gives rise to complex linguistic, semantic and reasoning-like behaviours.

While LLMs can perform a variety of tasks, including translation, question answering, summarisation, reasoning and dialogue, the underlying learning principle remains remarkably consistent. This section explains how LLMs learn from data. First, we clarify the foundational principle underlying almost all modern LLMs: learning as predictive modelling. We then discuss the training objectives used in practice and the data they require. Finally, we turn to the concrete mechanics of training Transformer-based architectures, including the interaction between the encoder and decoder components, loss functions, and optimisation. As almost all contemporary LLMs are based on Transformer variants, understanding how the Transformer is trained offers insight into how LLMs acquire linguistic structure, world knowledge and task-specific behaviour.

5.2.4.1. The core idea: learning as predictive modelling

At the heart of every modern LLM is the simple yet powerful concept that language learning can be framed as a prediction problem. The model is trained to estimate the probability of the next token, based on a sequence of preceding tokens and statistical regularities observed in large text corpora.

Let a text sequence be represented as a sequence of token IDs

$$[y_1, y_2, \dots, y_m],$$

where each $y_i \in \{1, 2, \dots, v\}$ corresponds to an element of a fixed vocabulary V.

Depending on the tokenisation scheme (see Section 5.1.2), tokens may represent words, subwords, characters, punctuation marks, or special symbols. From the model's perspective, language is not treated as sentences, phrases or syntactic trees, but rather as ordered sequences of discrete symbols.

LLMs are typically trained under an autoregressive factorisation of the joint probability of a sequence

$$p(y_1, y_2, \dots, y_m) = \prod_{i=1}^{m} p(y_i | y_1, y_2, \dots, y_{i-1}).$$

This decomposition does not make any explicit assumptions about syntax or semantics. Instead, all structure must be inferred implicitly from data through repeated exposure to sequences and their continuations. In this setting, learning consists of estimating the conditional distributions $p(y_i | y_1, y_2, \dots, y_{i-1})$ for all positions i, across a very large and diverse corpus. In other words, the training objective is to maximize the probability of the correct next token at every position in the sequence. This objective does not require labelled data or task-specific annotations. Any sufficiently large text corpus provides millions or billions of training examples simply by sliding a window over the sequence. Over time, this process aligns the model's predictions with the statistical patterns of the training data. Crucially, no explicit supervision about meaning, syntax, or correctness is provided beyond the raw text itself.

Although the objective operates at the level of individual tokens, repeatedly applying it across long sequences forces the model to capture increasingly abstract regularities:

- short-range dependencies (e.g. grammatical agreement, co-occurrence of words)
- long-range dependencies (e.g. cross-reference, topic continuity).
- semantic constraints (e.g. plausibility and coherence)
- pragmatic patterns (e.g. discourse structure and style).

These properties are not encoded explicitly, but rather emerge because accurate prediction of the next token requires internal representations that capture them.

Other paradigms follow the same predictive principle but modify the way the input is presented. In masked language modelling, for example, some tokens are replaced with a mask symbol, and the model must predict the missing content. In span corruption objectives, entire spans are removed, and the model must reconstruct the missing text (see Section 5.2.4.2). Although these approaches still rely on predictive modelling, they allow the model to utilise both

the left and right contexts, producing bidirectional representations that are useful for classification and retrieval tasks.

As discussed in Section 5.2.3, sequence-to-sequence models expand the predictive framework to tasks where the input and output vary, such as translation or summarisation. The model receives an input sequence (e.g. an English sentence) and is trained to predict the corresponding output sequence (e.g. the German translation). The decoder still performs next-token prediction, but is now conditioned on both the previously generated tokens and the encoded input. The learning principle remains unchanged: the model continues to predict the next piece of text.

In summary, the underlying mechanism is always the same, regardless of whether an LLM is trained to translate, answer questions, summarise documents or follow instructions: the model learns to predict missing, masked, or yet to come. This unifying perspective provides the conceptual foundation for understanding the more specialised training paradigms that will be introduced in the next sections.

Remark. The predictive learning paradigm of LLMs can be loosely compared to human language acquisition. Like humans, LLMs acquire language primarily through exposure to sequences of linguistic input rather than explicit instruction in grammatical rules. Effective language use also relies on expectations about what comes next in an ongoing expression. However, this similarity is purely conceptual – human language learning is rooted in perception, social interaction and robust cognitive inductive biases, whereas LLMs learn solely from textual data through large-scale statistical optimisation. Therefore, predictive modelling should be understood as a computational principle rather than a model of human cognition.

5.2.4.2. Training objectives and their data requirements

Large language models are trained in multiple stages, each with a different training objective and type of data. The first stage is pretraining, where the model acquires a general understanding of language and broad knowledge of the world from raw, unlabelled text using self-supervised objectives such as predicting the next token or masked language modelling. This creates a pretrained model – a general purpose foundation model that is not yet specialised for any particular task. Subsequent stages – which are referred to as 'training' in the broader sense – adapt this pretrained model to specific tasks, domains or behaviours ([194]). These stages include supervised sequence-to-sequence learning, instruction tuning, fine-tuning, and reinforcement learning from human feedback (RLHF).

Although all large language models ultimately learn by predicting text, the way in which this predictive signal is constructed varies depending on the training paradigm. Each objective imposes its own requirements on the training data, which strongly influence the tasks that a model can learn. Some objectives rely on raw text alone, while others require carefully aligned input-output pairs, labelled data or human-curated preferences ([195]). Understanding these differences is essential to appreciating why translation, question answering, summarisation and instruction processing are trained in fundamentally different ways. The following subsections provide a detailed description of the training objectives and their associated data requirements.

- **Autoregressive next-token prediction (GPT-style)**. Autoregressive language modelling is the simplest and most scalable training paradigm (see Section 5.2.3). The model receives a

[194] 'Training' is a generic term that can refer to any optimisation process, such as pre-training, tuning, domain adaptation, or continual updating. In simple terms, it means adjusting model parameters using gradient descent. So, while all pretraining is training, not all training is pretraining.

[195] Labelled data contains explicit target outputs and is used for supervised learning tasks such as classification, summarisation and translation. By contrast, curated data refers to text that has been selected, filtered or organised for quality or relevance, regardless of whether it contains labels. A dataset may be curated without being labelled, labelled without being curated, or both. Fine-tuning typically relies on curated, labelled data, whereas continual pretraining often uses curated, unlabelled text.

sequence of tokens and is trained to predict the next one. Crucially, this objective requires no labelled data. Any sufficiently large body of text provides billions of training examples simply by sliding a window over the text. Data requirement:

- unlabelled raw text
- no input–output pairs
- no task-specific annotation.

GPT-style models use a decoder-only Transformer. There is no separate encoder module as in BERT or encoder-decoder architectures such as T5. Instead, the single Transformer stack simultaneously represents the input context and predicts the next token. During training, the model always receives the true previous tokens as input (teacher forcing). Even though GPT-style training uses no human-annotated labels, the next token in the raw text serves as the automatically generated 'ground truth'. Only at inference time does the model feed back its own predictions to generate text autoregressively.

The training data includes articles, books, dialogues, forum discussions, Q&A websites, explanations and conversations, as well as many other sources. Within this extensive collection, the model identifies natural patterns, such as:

- questions followed by answers
- instructions followed by completions
- explanations
- reasoning chains
- dialogue turns
- problem-solution patterns
- cause-and-effect patterns
- definitions
- summaries

and so on.

Therefore, the model learns how humans typically respond to questions and instructions in general rather than memorising specific question-answer pairs. One of the most surprising aspects of GPT-style training is that it enables models to perform tasks for which no explicit 'ground truth' labels exist. These tasks include open-ended question answering, brainstorming, creative writing and explaining a concept – tasks that do not have a single correct answer in the way that translation or classification tasks do. For example, there is no single correct answer to prompts such as "*Explain transformers to a child*" or "*Give me 10 ideas for a birthday gift*". Yet GPT-style models learn to perform these tasks remarkably well.

The key point is that next-token prediction does not require predefined labels. Instead, the model learns from continuations that occur naturally within large text corpora. By predicting the next token in billions of contexts, the model internalises the statistical patterns involved in human communication, such as asking, answering, explaining, arguing and creating.

- **Masked language modelling (BERT-style)**. Models such as BERT (see Section 5.2.5) use raw text. However, rather than predicting the next token, the model predicts the masked tokens. This masking is synthetic, meaning that there do not need to be any actual gaps in the training data. Data requirement:

- unlabelled raw text
- masking is generated automatically
- no paired examples are needed.

This model produces representations that are useful for classification, retrieval, and embedding tasks. However, it does not directly support generative tasks, such as translation or long-form text generation.

Example. Suppose we take a raw sentence from a text corpus:

"Neural networks are powerful models for natural language processing."

During BERT-style training, some of the tokens are randomly masked. For example:

"Neural networks are powerful models for natural [MASK] processing."

The model's task is to predict the missing token ("language") using the context on the left and right. This differs from autoregressive models, which only consider the left context.

In order to predict the token "language", the model must understand that "natural ___ processing" is a common phrase, that "language" fits syntactically and semantically, and that "processing" is often paired with "language" in technical texts. This enables the model to internalise linguistic structure without any labelled supervision.

This example illustrates the following key points:

- Masked LM uses only raw text – no labels and no paired examples.
- The model learns bidirectional representations.
- The training signal is synthetic (the mask is artificially inserted).

This paradigm is fundamentally different from supervised tasks such as translation.

- **Span corruption and denoising (T5-style)**. Denoising objectives generalise masked language modelling by removing entire sections of text and asking the model to reconstruct them. This 'span corruption' training objective is used in models such as Google Research's T5 (Text-to-Text Transfer Transformer), in which contiguous random sequences of input text (spans) are replaced by a single unique mask token (known as a 'sentinel token'). The model is then trained to reconstruct the original masked spans ([G28]). As with previous paradigms, this approach relies entirely on raw text. Data requirement:

- unlabelled raw text
- synthetic corruptions
- no human annotation.

Since the model learns to map corrupted inputs to clean outputs, it can be naturally extended to sequence-to-sequence tasks like summarisation and question answering.

- **Supervised sequence-to-sequence learning (e.g. translation)**. Machine translation is a fundamentally different paradigm. The model is trained using paired sequences, such as an English sentence and its German translation (see Section 5.2.4). This is a classic supervised learning setup, whereby the model must learn to map one sequence onto another. Data requirement:

- parallel corpora (aligned input–output pairs)
- human-produced translations
- high-quality alignment between the source and target.

This data is expensive to produce and limited in availability. Importantly, translation data is not suitable for training open-ended question answering systems, as QA does not consist of aligned sentence pairs in the same way. A translation model that is only trained on parallel corpora would not learn to answer questions, summarise documents or engage in dialogue.

- **Instruction tuning**. This involves training the model using pairs of instructions and their desired responses, which provides an additional form of supervision. In this approach, the model is trained using sequences that explicitly include a natural-language input written by a human,

alongside plain text. This instruction then becomes part of the model's input context, just like any other token. Data requirement:

- (instruction, response) pairs
- human-written
- broad task coverage.

This paradigm teaches the model to follow user instructions, even for tasks that it was not explicitly trained on during pretraining. Instruction tuning relies on teacher forcing during training, whereby the model is conditioned on the instruction and the ground-truth prefix of the response in order to learn to predict the next token. However, while teacher forcing is a general training technique, instruction tuning is a specific approach for teaching models to follow natural-language instructions. When a user interacts with the model at inference time, the model only receives instructions and optional input text. It must then generate the response from scratch, without being shown the ground-truth answer.

Example. During instruction tuning, the model is trained with teacher forcing: the instruction, the input text, and the ground-truth prefix of the response are concatenated into a single sequence:

- **Instruction**: *Summarise the following paragraph in one sentence*.
- **Paragraph**: *Neural networks are computational models inspired by the brain. They consist of layers of interconnected units that transform input data into increasingly abstract representations. Modern architectures such as Transformers have enabled significant advances in natural language processing*.
- **Response** (prefix only): *Neural networks learn layered representations of data, enabling modern advances in NLP.*

The model is then trained to predict the next response token.

- **Fine-tuning**. Fine-tuning involves adapting a pretrained language model to a specific task, domain or style by training it on a tailored dataset of input-output examples. Unlike GPT-style pretraining, which uses only raw text, fine-tuning introduces task-specific supervision. The model is presented with an input, such as a document, question or prompt, and the desired output, such as a label, summary, extraction or answer. During training, the model processes the input using the same decoder-only Transformer stack as in pretraining. It is then conditioned – via teacher forcing – on the ground-truth output prefix to learn to predict the next token. Data requirement:

- (input, output) pairs
- task-specific or domain-specific
- human-curated or automatically generated.

Fine-tuning teaches the model to reliably perform a specific task, even if that task was rare or absent in the pretraining corpus. This process can specialise the model for specific domains, such as law, medicine, finance or customer support, or for particular tasks, such as classification, summarisation, extraction or structured generation. As with instruction tuning, teacher forcing is simply the training mechanism. The distinctive feature of fine-tuning is the task-aligned supervision provided by the input-output pairs. At inference time, the model only receives the input (e.g. a document or question) and must generate the output from scratch without access to the correct answer.

- **Reinforcement Learning from Human Feedback (RLHF)**. This learning method introduces a different kind of supervision. Rather than providing correct answers, humans indicate their preferences between model outputs. A reward model is then trained using these preferences, after which the LLM is optimised to produce outputs that humans prefer. Data requirement:

- human preference comparisons

- no explicit 'correct answer'
- used to shape behaviour, not core knowledge.

This paradigm is essential for aligning LLMs with human expectations of helpfulness, safety and conversational quality.

In summary, different tasks require different training objectives, which in turn require different types of data. Modern LLMs therefore combine multiple paradigms – pretraining on raw text, supervised fine-tuning on labelled examples and alignment through human feedback – to achieve broad and reliable capabilities.

5.2.4.3. Training the Transformer

Because the Transformer serves as the foundational architecture for nearly all current LLMs, its training procedure provides a canonical example of the learning mechanisms employed in large-scale language models. However, despite relying on standard backpropagation, the Transformer's training dynamics differ in several respects from those of earlier neural architectures. These differences are due to the model's dual-stack structure (encoder and decoder), its use of self-attention and the autoregressive nature of the decoder. The following focuses on the mechanisms specific to Transformer training.

- **Joint training of encoder and decoder**. Training a Transformer differs from classical neural networks primarily due to its jointly optimised encoder-decoder structure, and the presence of attention-based parameter couplings across all positions and layers. Given an input sequence X and a target sequence Y, the encoder produces a sequence of contextual representations (see Section 5.2.3.1)

$$M = Encoder(X; \boldsymbol{\theta}_E),$$

which are then consumed by the decoder to define a conditional distribution

$$p(\, Y \mid X; \boldsymbol{\theta}_E, \boldsymbol{\theta}_D) = \prod_{i=1}^{m} p(y_i | y_1, y_2, \dots, y_{i-1}, M; \boldsymbol{\theta}_D),$$

where $\boldsymbol{\theta}_E$ and $\boldsymbol{\theta}_D$ represent the trainable parameters of the encoder and decoder, respectively. Therefore, the loss function (typically cross-entropy) is a function of the parameters of both the encoder and the decoder

$$L(\boldsymbol{\theta}_E, \boldsymbol{\theta}_D) = -\sum_{i=1}^{m} log[p(y_i | y_1, y_2, \dots, y_{i-1}, M; \boldsymbol{\theta}_D)].$$

This implies that the gradients computed at the decoder output propagate backwards not only through the decoder stack, but also through the cross-attention sublayers into the encoder representations M.

In other words, during training, the loss is only computed from the decoder's output distribution over the target tokens. but the gradient flows:

- from the output layer,
- through all decoder layers,
- through the cross-attention connections,
- into all encoder layers,
- and finally into the source embeddings.

This means that the encoder is optimised not to reconstruct the input sentence, but rather to produce representations that are as useful as possible for the decoder's prediction task. The entire architecture is therefore trained as one large computational graph.

- **Teacher forcing and prefix encoding**. During training, the decoder receives the ground-truth prefix at every step (teacher forcing – see Section 5.2.3.3). This has two significant implications for backpropagation:

- Each decoder hidden state encodes the correct prefix

$$[BOS,\ y_1, y_2, \ldots, y_{i-1}],$$

rather than the model's own predictions.

- The loss at position i depends only on the model's distribution for the next token, but the gradient flows backwards through the entire prefix-encoding computation.

Consequently, the decoder learns to map the correct prefixes to the correct next-token distributions, thereby stabilising training and accelerating convergence.

- **Backpropagation through self-attention**. Self-attention introduces a unique gradient flow pattern. Each attention head computes attention scores, softmax weights, and weighted sums of value vectors. During backpropagation, gradients flow through the softmax into the attention weights and then into the query, key and value representations (see Section 5.2.2.3). They ultimately reach all tokens that contributed to the attention context. This means that every token can influence the gradient of every other token, which differs fundamentally from RNNs (which have sequential dependency) and CNNs (which have local receptive fields). The global connectivity of attention is one reason why Transformers learn long-range dependencies so effectively.

- **Backpropagation through cross-attention**. Cross-attention is the mechanism that connects the decoder and the encoder. During training, the decoder queries the encoder's output matrix M and attention weights determine which source positions are relevant. Gradients then flow back from the decoder to the encoder through these attention links. This is why the encoder and decoder cannot be trained independently – the decoder's loss directly shapes the encoder's representations.

- **Parallel backpropagation across sequence positions**. Unlike RNNs, the Transformer processes all positions simultaneously. This parallelism also applies to backpropagation. All token-level losses are computed simultaneously and gradients flow backwards through all layers at once. There is no sequential dependency in the gradient path. These features make Transformer training highly efficient on modern hardware, which is one of the reasons why the architecture scales so well.

- **Layer normalization and residual paths**. Residual connections and layer normalisation introduce additional gradient pathways. The residual paths allow for direct gradient flow around attention and FFN sublayers. The parameters of layer normalisation are trained jointly with the rest of the model, since gradients must pass through normalisation statistics, which affects the dynamics of optimisation. Together, these components help to prevent vanishing gradients in deep Transformer stacks.

Ultimately, the parameter updates are performed jointly

$$(\boldsymbol{\theta}_E, \boldsymbol{\theta}_D) \leftarrow (\boldsymbol{\theta}_E, \boldsymbol{\theta}_D) - \alpha \frac{\partial L}{\partial (\boldsymbol{\theta}_E, \boldsymbol{\theta}_D)},$$

reflecting the fact that the Transformer is trained as a single coupled system, rather than as independently optimised encoder and decoder components.

5.2.4.4. Continual updating and post-training adaptation

Large language models are usually trained using extensive, static corpora collected at a specific point in time. While this pretraining phase provides the model with a broad understanding of language and extensive knowledge of the world, it does not ensure that the model will remain up to date, adaptable or aligned with evolving user needs. After pretraining, models often undergo continual updating – a set of techniques that enables them to incorporate new information, specialise in new areas or refine their behaviour without being retrained from scratch. Continual updating is therefore not part of the model's initial learning paradigm, but rather a mechanism for extending and maintaining its capabilities over time. However, it is important to distinguish continual updating from *continuous learning*, whereby models dynamically update their parameters in real time based on new data or interactions.

Continual updating addresses several challenges:

- *Temporal drift*. Facts, events and conventions change after the pre-training corpus has been collected.
- *Domain adaptation*. Users may require expertise in specialised fields that are not well represented in the original data.
- *Task evolution*. New tasks or interaction styles emerge that were not part of the original training distribution.
- *Safety and alignment maintenance*. Models must be updated periodically to reflect new safety guidelines, ethical considerations and user expectations.

Continual updating ensures that a model remains useful, reliable and aligned long after its initial training. This is not a single technique, but rather a family of approaches that modify the model or its behaviour after pretraining.

- **The static nature of LLM parameters**. Once an LLM has been trained and deployed, its parameters are usually fixed. Inference then occurs using this fixed set of parameters, and the model's generative capacity is determined by the learned distribution. Unlike human learning, LLMs do not perform continuous gradient updates after deployment. This means that the model's parameters are not updated or modified directly based on user interactions or incoming data streams. The model's behaviour is therefore determined by the pre-existing weight configuration, and any adjustments require offline retraining or fine-tuning processes.

- **Periodic retraining and fine-tuning**. Although LLM parameters remain static during inference, they can be periodically retrained or fine-tuned using new data. In this scenario, model updates occur offline in a controlled environment where new datasets, often derived from real-world interactions or domain-specific corpora, are introduced. The model is then retrained by optimising the loss function with respect to the updated data and adjusting the parameters accordingly. This process is typically referred to as 'fine-tuning' and does not occur continuously. Once training is complete, the updated model is redeployed, reflecting the most recent adaptation to the data. While this retraining process is periodic, it does not constitute continuous learning, since it necessitates retraining on batch data and often involves a comprehensive re-evaluation of the model.

- **Parameter-efficient adaptation**. When model size and retraining time are prohibitive, parameter-efficient adaptation techniques have been developed. These methods enable LLMs to 'adapt' without updating the entire model. Techniques such as low-rank adaptation (LoRA), adapters and prefix tuning enable efficient updates by introducing small, task-specific parameters while keeping the main model 'frozen'. These mechanisms modify only a small portion of the model parameters – often the intermediate layers – thus enabling the model to adapt to new tasks or domains with minimal computational cost ([196]). Although these approaches allow for flexible adaptation, they still necessitate a degree of controlled offline adjustment, and the core model parameters remain unmodified. This further highlights that LLMs do not engage in continuous updating in a fully online, real-time manner.

- **Non-parametric adaptation**. In addition to direct parameter updates, LLMs may exhibit adaptive behaviour through non-parametric methods, such as in-context learning and *retrieval-augmented generation* (RAG). Rather than modifying the model's parameters, these methods

[196] Adapters, LoRA and prefix tuning are all parameter-efficient fine-tuning methods that only modify a small subset of a model's parameters. Adapters use bottleneck layers to provide a compact mechanism for modifying a large model; the pretrained weights remain frozen while the small bottleneck modules learn task-specific adjustments efficiently without destabilising the base model. LoRA (Low-Rank Adaptation) learns low-rank update matrices that are added to the model's weight projections during training, enabling efficient adaptation with minimal additional parameters. Prefix tuning prepends a set of learned key-value vectors to the model's attention mechanism, effectively steering the model's behaviour without altering its internal weights. These techniques reduce computational costs and prevent the overwriting of the model's general abilities, while still enabling domain- or task-specific adaptation.

adjust the context provided to the model during inference. In-context learning enables the model to incorporate new information provided at runtime, while RAG retrieves relevant documents or data from external sources to guide the generation process. In these cases, while the model's internal weights remain fixed, the output is influenced by dynamic, context-specific information provided at runtime. This form of adaptation enables the model to respond flexibly to novel queries or tasks, eliminating the need for retraining or parameter updates.

- **Continuous learning**. The adaptation mechanisms discussed above – retraining, parameter-efficient updates and non-parametric methods – are often mistakenly referred to as 'continuous learning'. However, these mechanisms are distinct from 'true' continuous learning, in which the model parameters are updated online as new data arrives and the system continuously adjusts its internal weights to accommodate new patterns. One challenge of this approach is catastrophic forgetting, whereby the model may lose knowledge from earlier experiences when trained on new data. Currently, continuous learning is not used in major deployed LLMs. It remains an active area of research, rather than a production technique.

Taken together, these approaches demonstrate the various methods by which modern LLMs are maintained, adapted and expanded beyond their initial pre-training. The following table summarises the main adaptation strategies and provides examples of representative models for each one.

Adaptation Strategy	Meaning	Examples	Production use?
Periodic retraining & fine-tuning	Batch updates, new versions	GPT-5, GPT-4.1, Claude3, Gemini, LLaMA-3	✓ mainstream
Parameter-efficient adaptation	Small trainable modules (LoRA, Adapters)	LLaMA-2/3 LoRA, T5 Adapters, Mistral LoRA	✓ very common
Non-parametric adaptation	External memory/retrieval; model weights stay fixed	GPT-4 with retrieval, RETRO, Cohere RAG, Microsoft Copilot	✓ standard in production
Continuous learning	Continuous parametric updates without forgetting	Gato, PaLM-E, academic models	× mostly research

5.2.5. Transformer-based pre-trained language models

Since the introduction of the Transformer architecture, a diverse ecosystem of pre-trained language models has emerged, each exploring different architectural choices, pre-training objectives and adaptation strategies. While all these models are based on the same fundamental attention-based framework, their creators have introduced targeted modifications to address limitations of the original architecture or better exploit large-scale pretraining. These innovations have shaped the evolution of modern NLP and laid the conceptual foundations for today's large language models.

This section examines representative examples of four major classes of Transformer-based pre-trained language models: encoder-only models (e.g., BERT), encoder–decoder models (e.g., BART and T5), decoder-only models (e.g., GPT-5), and mixture-of-experts decoder architectures (e.g., DeepSeek-V3). It subsequently discusses Microsoft Copilot as a large-scale application instantiated on top of such model architectures.

BERT and BART exemplify the autoencoding and denoising paradigms, demonstrating how bidirectional context and reconstruction objectives can generate robust representations for understanding tasks. T5 extends the encoder–decoder architecture into a unified text-to-text framework, showing how a single model can handle a wide range of NLP tasks via a consistent interface. In contrast, the GPT family embodies the autoregressive paradigm, revealing how next-token prediction at scale can produce powerful generative models capable of reasoning, following instructions, and generating open-ended text. DeepSeek-V3 represents a recent evolution of this autoregressive lineage, combining large-scale Transformer training with a

mixture-of-experts architecture to achieve strong performance while improving parameter efficiency and inference scalability. Copilot belongs to the broad family of Transformer-based systems. However, the exact details of its architecture are not publicly available.

As the canonical Transformer architecture has already been described in detail earlier in Section 5.2.3, the following sections focus specifically on the innovations at the model level that distinguish these families, including their historical context, architectural modifications, pretraining objectives and limitations or challenges arising from design choices. Together, these models demonstrate how minor architectural decisions and carefully selected training objectives can result in significantly different capabilities, behaviours, and downstream applications.

5.2.5.1. BERT (2018) and BART (2019)

BERT (*Bidirectional Encoder Representations from Transformers*) was introduced in 2018 by Jacob Devlin et al. at Google AI Language ([D12]). This marked a decisive shift in NLP, demonstrating that large-scale bidirectional pretraining could produce far superior contextual representations to earlier unidirectional or shallow models. BERT quickly became the dominant foundation for classification, question answering (QA) and sentence-level tasks, and it triggered the 'pretrain-then-fine-tune' paradigm that defined NLP from 2018 to 2020.

BART (*Bidirectional Auto-Regressive Transformers*) was introduced by Mike Lewis et al. ([L17]) at Facebook AI Research (now Meta AI Research) in 2019. Building on the BERT concept, it incorporates a denoising encoder–decoder architecture that combines a BERT-like encoder with a GPT-like decoder. Designed to combine the strengths of masked language models (strong understanding) and autoregressive decoders (strong generation), BART is ideal for summarisation, translation, and text generation tasks.

Together, BERT and BART represent the autoencoding ([197]) and denoising branches of Transformer pretraining ([198]).

- **Architecture and modifications relative to the Vaswani Transformer**. BERT only adopts the encoder part of the Transformer architecture introduced by Vaswani et al. ([V2]). This reuse is possible because the Vaswani encoder already employs full bidirectional self-attention with no causal mask. This enables every token to attend to all others, allowing deep contextualisation. Consequently, BERT's encoder is architecturally very similar to the original design, with only a few targeted modifications. Firstly, special tokens and segment embeddings are used: $[CLS]$ for classification and $[SEP]$ for sentence-pair tasks. This supports downstream tasks without requiring architectural changes. The second modification is increased depth and scale. BERT uses 12–24 encoder layers, whereas the original Transformer uses 6. The objective of BERT is to generate rich, bidirectional representations to facilitate task comprehension. As the Vaswani encoder already provides bidirectional attention, BERT can reuse this and concentrate on the pretraining objective rather than the architecture. In summary, BERT contains the Vaswani encoder, but not a decoder. As an encoder-only model, BERT cannot generate text autoregressively, since its bidirectional attention disrupts the left-to-right generative process.

---

[197] Recall that autoencoding refers to training a model to reconstruct the original input from a transformed or corrupted version of that input. It is a training objective rather than an architectural feature: the model learns an internal representation by attempting to recover the clean data. BERT uses a partial autoencoding objective through masked language modelling, while BART employs a full denoising autoencoding objective using an encoder–decoder Transformer.

[198] Both models were released as open source by their respective creators (Google for BERT, Meta AI for BART). This allows anyone to inspect, modify and build upon their code and pre-trained weights. The original BERT repository is available on GitHub. Various versions of the pre-trained models (e.g. bert-base-uncased and facebook/bart-large) are hosted on community platforms such as the Hugging Face Model Hub, which makes them easy to implement without needing to train them from scratch. These models are integrated into popular libraries such as the Hugging Face transformers library, which simplifies their use for researchers, developers and hobbyists with a basic understanding of Python and deep learning concepts.

BART uses the same full encoder-decoder structure as the original Transformer, but with different roles for each part. The encoder is a BERT-like bidirectional encoder similar in structure to the Vaswani model, but it is trained using denoising objectives ([199]) rather than translation objectives. The decoder is a Vaswani-style autoregressive decoder that utilises causal masking in both self- and cross-attention over the encoder's outputs. This mirrors the original Transformer decoder used for machine translation. Additionally, BART introduce a flexible set of input corruption strategies that the decoder must reconstruct.

- **Differences in training**. BERT is trained using two techniques: masked language modelling (MLM) and next sentence prediction (NSP). During MLM pretraining, 15% of tokens are selected for prediction. Of these, 80% are replaced with the special token [$MASK$], 10% with a random vocabulary token and 10% remain unchanged. This corrupted sequence is then fed into a standard Vaswani-style encoder, which processes it using full bidirectional self-attention. The encoder itself contains no corruption layers; all corruption is applied externally during data preprocessing, rather than within the Transformer architecture. NSP is implemented as an auxiliary classification task using the [CLS] embedding; however, this is also a training objective rather than an architectural modification. The role of the encoder is to produce contextualised representations from which the model can predict the original, uncorrupted tokens. The key difference from the standard Transformer training approach is that the Vaswani decoder uses causal masking and next-token prediction, whereas the BERT encoder uses bidirectional MLM, creating a pretraining-inference mismatch.

BART generalises BERT's MLM into a full denoising autoencoding framework. As with BERT, corruption is implemented as a set of external data preprocessing functions that are applied before the input reaches the encoder. During pretraining, the original text is transformed through masking, deletion, span filling, or sentence permutation. This corrupted sequence is then passed to the BERT-like bidirectional encoder. The BART decoder is a Vaswani-style autoregressive decoder that is trained to reconstruct the original, uncorrupted text one token at a time. Neither the encoder nor the decoder contains any corruption layers; the denoising behaviour emerges entirely from the training objective, not from architectural changes.

- **Differences in inference**. The way BERT makes inferences differs fundamentally from the original Vaswani Transformer model because BERT only contains the encoder stack and therefore does not perform any autoregressive decoding. As with all Transformer models, the actual input to the encoder is the sum of token embeddings and positional encodings rather than raw text. BERT also introduces segment embeddings to indicate whether each token belongs to the first or second sentence of a sentence pair input ([200]). These three embedding components are added together to form the encoder's input representation. During inference, BERT processes the entire input sequence in a single, parallel forward pass using full, bidirectional, self-attention. The purpose of this processing is to understand the input, not to generate new text. The BERT output comprises contextualised token representations rather than new text. These representations are then fed into task-specific heads – small neural modules added to the encoder for downstream tasks such as classification, token labelling or question answering. These heads are not related to multi-head attention; rather, they are lightweight output layers that are trained during fine-tuning. As BERT does not generate text, its inference procedure is

[199] BART and diffusion models (see Section 3.2.17) are linked conceptually through their use of denoising objectives; both models learn to reconstruct clean data from corrupted input. However, they belong to different generative paradigms. BART performs discrete, single-step denoising within a Transformer architecture, whereas diffusion models learn to reverse a continuous, multi-step noising process and generate data through iterative refinement. This makes them conceptually related, yet architecturally and operationally distinct.

[200] In BERT, many tasks involve processing two text segments simultaneously in a single forward pass. This is known as sentence-pair input. BERT uses segment embeddings – also referred to as token type embeddings – to identify which token belongs to which segment. One example is *Semantic Textual Similarity* (STS), i.e. the question "*How similar are two sentences?*". In this case, the input to BERT would be: (Segment A) "*The cat sat on the mat.*" and (Segment B) "*A cat is sitting on a rug.*". BERT will output a similarity score.

non-autoregressive, encoder-only and parallel. This is in contrast to the original Transformer's autoregressive encoder-decoder inference.

Although BART's inference procedure is similar to the original Vaswani Transformer, it differs in the nature and purpose of its inputs. As with all Transformers, the encoder receives token embeddings and positional encodings. However, unlike BERT, BART does not use segment embeddings because its encoder is designed to process a single input sequence rather than pairs of sentences. The purpose of the input differs, too. In BERT, the input is processed to produce representations for understanding tasks, whereas in BART, the input is processed to provide the conditioning context for text generation. During inference, the BART encoder produces a bidirectional, contextualised representation of the input sequence. The decoder then generates output tokens autoregressively using causal masking, in the same way as the Vaswani Transformer. Each newly generated token is fed back into the decoder (again as token embeddings + positional encodings), and standard decoding strategies such as greedy decoding are applied. Therefore, the output of BART is generated text, produced one token at a time. This is in contrast to BERT's non-generative contextual embeddings. Although task-specific heads can be added for fine-tuning on downstream tasks, in generative settings, the decoder's output projection layer serves as the primary output mechanism.

- **Key applications**. As BERT is an encoder-only model that creates contextualised representations rather than generating text, it is primarily used for tasks that require deep semantic understanding, classification or token-level labelling. Typical use cases include:

- *Text classification*. Sentiment analysis, topic classification, and document categorisation rely on BERT's ability to encode entire sequences into rich semantic representations.
- *Sentence-pair tasks*. Natural language inference (NLI) ([201]), semantic textual similarity (STS) and question-answer matching benefit from BERT's segment embeddings and bidirectional context modelling.
- *Token-level prediction*. Named entity recognition (NER), part-of-speech tagging and other sequence-labelling tasks use BERT's contextual token embeddings.
- *Extractive question answering*. BERT identifies answer phrases within a passage by predicting their start and end positions, using its bidirectional attention.
- *Information retrieval and ranking*. BERT-based bi-encoders and cross-encoders ([202]) are widely used in search engines, semantic retrieval and document ranking.
- *Embedding generation*. BERT produces high-quality sentence and token embeddings for downstream processes such as clustering, recommendations, and semantic searches.

These applications all exploit BERT's strengths as a non-generative, high-fidelity encoder that excels at understanding and representing text.

BART combines a bidirectional encoder with an autoregressive decoder, making it well-suited to tasks that require both understanding and generation. Its denoising pre-training objective enables it to handle a wide range of sequence-to-sequence applications:

- *Abstractive summarisation*. BART is widely used to summarise long documents, news articles and reports, producing fluent and coherent summaries.

---

[201] Natural language inference (NLI) involves determining the logical relationship between two pieces of text, usually a premise and a hypothesis. It is also referred to as *Recognising Textual Entailment* (RTE).

[202] A bi-encoder (also called a dual encoder or Siamese encoder) is an architecture in which two input sequences X_1 and X_2 are encoded independently by a shared Transformer, yielding representations H_1 and H_2. Their interaction is computed only at the representation level, typically via a similarity function $sim(H_1, H_2)$ (e.g., cosine similarity or dot product). This factorization enables efficient large-scale retrieval because representations can be precomputed and indexed. In contrast, a cross-encoder jointly processes the concatenated pair (X_1, X_2) within a single Transformer, allowing full token-level cross-attention across both sequences.

- *Paraphrasing and text rewriting*. Its encoder–decoder structure enables BART to generate alternative phrasings while preserving meaning.
- *Dialogue and conversational agents*. BART can generate contextually appropriate responses based on previous exchanges.
- *Machine translation (general purpose)*. Although not specifically trained for translation, BART's architecture supports translation tasks when fine-tuned.
- *Data-to-text generation*. BART can generate natural language descriptions from structured inputs, such as tables or records.
- *Long-form content generation*. Its autoregressive decoder enables coherent multi-sentence generation when conditioned on prompts or source text.
- *Text infilling and controlled editing*. BART can fill in missing text or rewrite sections, reflecting its denoising pre-training.

These applications make use of BART's two main strengths: deep contextual understanding from the encoder and fluent text generation from the decoder.

Although BERT and BART are foundational models in modern NLP, their architectures and training paradigms introduce constraints that influence how they can be applied in practice and how effectively they can perform. Understanding these limitations is essential for evaluating their suitability for specific tasks, anticipating performance bottlenecks and motivating the development of more advanced architectures. The following section outlines the key limitations and challenges associated with each model.

- **Limitations and challenges**. Although BERT excels at understanding language, its applicability is constrained by several structural and methodological limitations:

- ✕ *Inability to generate text*. BERT has no decoder or autoregressive mechanism. Consequently, it cannot produce free-form text, complete prompts, or perform sequence-to- sequence tasks. Its outputs are contextual embeddings, not generated sequences.
- ✕ *Pre-training–inference mismatch*. BERT is trained using masked language modelling, whereby the model is presented with corrupted input. At inference time, however, the input is uncorrupted. This mismatch can lead to suboptimal representations, particularly for rare or ambiguous tokens.
- ✕ *Limited handling of long contexts*. Like the original Transformer encoder, BERT's performance decreases as the sequence length increases due to full self-attention. This restricts practical input lengths and makes processing long documents computationally expensive.
- ✕ *Sentence-pair bias from NSP*. The Next Sentence Prediction objective introduces biases towards sentence-pair tasks and may not improve performance for all downstream applications. Later models (e.g. RoBERTa) removed NSP entirely.
- ✕ *Computational cost of fine-tuning*. BERT's large number of parameters makes fine-tuning expensive, especially in environments with limited resources. Memory requirements can be prohibitive for large batch sizes or long sequences.
- ✕ *Static input length and positional encoding*. BERT relies on absolute positional encodings with a fixed maximum length. Extending beyond this limit requires either architectural modification or retraining.

Although BART's encoder-decoder structure enables powerful generative capabilities, it also introduces its own challenges.

- ✕ *High computational cost*. BART combines a full bidirectional encoder with an autoregressive decoder. This makes both training and inference significantly more expensive than models with only an encoder.

- ✕ *Exposure bias in autoregressive decoding*. Like all autoregressive models, BART suffers from exposure bias. During training, the decoder sees the true tokens, but during inference, it must base its predictions on itself. This can lead to error accumulation in long sequences.
- ✕ *Limited control over output structure*. Although BART can produce fluent text, controlling style, structure, or factuality remains challenging. The model may produce inaccurate information or overly wordy outputs, particularly when the input is ambiguous.
- ✕ *Dependence on denoising pretraining*. BART's encoder is trained on corrupted inputs, yet receives clean inputs at inference time. This mismatch can affect performance on substantially different tasks.
- ✕ *Quadratic attention complexity*. Both the encoder and the decoder's self-attention mechanisms scale quadratically with sequence length. This limits BART's ability to handle long documents efficiently, particularly for summarisation tasks.
- ✕ *Difficulty with highly structured outputs*. Tasks that require strict formatting, such as code generation and structured data extraction, can be challenging because the BART decoder is optimised for natural language fluency rather than structural precision.

5.2.5.2. T5: Text-to-Text Transfer Transformer (2020)

The Text-to-Text Transfer Transformer (T5) was introduced in 2020 by Colin et al. in the influential Google Research paper, "*Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*" ([C14]). T5 emerged from a systematic investigation of transfer learning in NLP ([203]) to unify a wide range of tasks under a single text-to-text formulation. This unification was motivated by the observation that NLP tasks often require specialised architectures or output heads, which complicates transfer learning and model reuse. T5 proposes a single model that treats every NLP task – classification, translation, summarisation, question answering, and more – as a sequence-to-sequence text generation problem ([204]).

- **Architecture and modifications relative to the Vaswani Transformer**. The T5 model retains the encoder-decoder structure of the original Vaswani Transformer, but introduces several important modifications designed to enhance scalability, stability and the range of tasks it can perform. The key architectural differences are:

- *Relative positional encodings*. T5 replaces absolute positional encodings with relative position biases ([205]) to improve generalisation to longer sequences and enable more flexible handling of variable-length inputs.

---

[203] *Transfer learning* in NLP refers to the practice of pre-training a model on a large, general-purpose corpus to learn reusable linguistic representations, and subsequently adapting the same model to specific downstream tasks – often with limited task-specific data – by fine-tuning or prompt-based conditioning.

[204] The T5 model and its variants (such as Flan-T5 and mT5) are released under the Apache 2.0 licence, which is permissive. This generally allows for both non-commercial and commercial use, subject to certain standard legal conditions. The pre-trained models and fine-tuning scripts can be accessed via platforms such as the Hugging Face Transformers library and the original Google Research GitHub repository. While the model is accessible, using it effectively does require some technical knowledge. Typically, users need programming skills (e.g. Python), an understanding of machine learning concepts and access to sufficient computational resources (e.g. GPUs or TPUs) for fine-tuning or running large-scale inference.

[205] Absolute positional encodings are added to token embeddings at the input layer to provide information about their fixed position in the sequence (see Section 5.2.1). In contrast, relative position biases do not modify the input representations; rather, they adjust the attention scores by incorporating a learned bias term that depends on the distance between the query and key positions. Thus, absolute encodings provide position information independently of pairwise interactions. Relative biases directly modulate attention as a function of token-to-token distance, which often improves generalisation across varying sequence lengths.

- *Layer normalisation placement*. T5 uses pre-norm (LayerNorm before each sublayer) rather than the post-norm configuration of the original Transformer. This improves training stability, particularly for very deep models.
- *Feed-forward activation*. T5 uses Gated GELU (GeGLU) activations in the feed-forward layers, which have been shown to outperform ReLU ([S34]).
- *Unified text-to-text interface*. Unlike BERT (encoder-only) or GPT (decoder-only), T5's architecture is designed to support a single interface. Both the input and the output are text. This simplifies task formulation and enables broad transfer learning.
- *No segment embeddings*. As T5 treats all tasks as text-to-text, it does not require BERT-style segment embeddings. Task distinctions are encoded through task prefixes (e.g. "*Translate English to German*: …").

These modifications were introduced to improve scalability to very large model sizes (up to 11 billion parameters in the original paper) and to support a unified task format, thereby reducing architectural fragmentation. They also enhance generalisation across tasks and sequence lengths.

- **Differences in training**. There are several important differences between the training pipeline for T5 and the Vaswani Transformer.

- *A unified text-to-text objective*. Every task is presented as a text-to-text mapping. Examples:
  - "Summarise: [article]" → [summary]
  - "Translate from English to French: [sentence]" → [translation]
  - "Classify sentiment: [text]" → 'Positive'.

  This eliminates the need for task-specific output heads.
- *Span-corruption pretraining (denoising)*. T5 uses a span-corruption objective called *span-masked language modelling*. Random spans of tokens are replaced with a single sentinel token ([206]) (e.g. $< extra\_id\_0 >$). The target sequence is then the concatenation of the missing spans, each one preceded by its respective sentinel token. This differs from BERT's token-level masking and BART's flexible corruption functions. T5's span corruption encourages the model to learn long-range dependencies, thereby improving generative performance.
- *Massive multi-task training*. T5 is pretrained on a mixture of supervised and unsupervised text-to-text tasks constructed from diverse datasets. Unsupervised tasks are derived from the C4 corpus ([207]) via span corruption, whereas supervised tasks are created from datasets focusing on translation, summarisation, question answering and classification, each of which is reformulated into a unified text-to-text format. This mixture of tasks is central to T5's generalisation ability.
- *Scaled training*. T5 was trained at multiple parameter scales (approximately 60M, 220M, 770M, 3B, and 11B parameters), exhibiting systematic performance improvements as model capacity increased.

---

[206] A sentinel token is a special mask token used in T5 to indicate the start and end of masked or corrupted text during self-supervised pre-training.

[207] The C4 corpus (*Colossal Clean Crawled Corpus*) is a large, systematically cleaned subset of Common Crawl designed to provide high-quality English text for pre-training. It removes boilerplate content, non-English text, duplicates and low-quality pages, and is the main source of unsupervised training for T5's span-corruption objective. The term 'boilerplate' refers to non-content text that appears on web pages, such as navigation menus, headers, footers, adverts and template elements. Common Crawl is operated by the Common Crawl Foundation, which is a non-profit organization based in the United States.

- **Differences in inference**. Although the T5 model retains the encoder-decoder structure of the original Vaswani Transformer model, its inference procedure differs in several important ways that reflect its unified text-to-text design.

  - *Unified text-to-text inference*. The Vaswani Transformer was originally designed for machine translation, and its inference pipeline is tied to this specific task. T5 generalises this approach by treating every NLP task as a text-to-text problem. The input is a textual prompt that specifies the task, and the output is a textual sequence that is generated autoregressively. This means that classification, summarisation, translation, question answering and regression are all performed through the same decoding mechanism, without task-specific output heads.
  - *Task prefixes instead of architectural changes*. While the Vaswani Transformer requires different output layers, or even different architectures, for different tasks, T5 simply uses task prefixes. These prefixes guide the decoder during inference, enabling a single model to perform a wide range of tasks without architectural modification.
  - *Autoregressive decoding across all tasks*. Like the Vaswani Transformer, T5 generates output tokens one at a time using causal masking. However, the purpose of decoding is broader:
    - For summarisation, it generates a multi-sentence abstract.
    - For classification, it generates a label such as 'positive'.
    - For question answering, it generates a short answer.
    - For translation, it generates a sentence in the target language.

    Thus, T5 uses the same decoding process for tasks that the original Transformer would handle using different architectures.
  - *There are no task-specific output heads*. A major difference from both the Vaswani Transformer and BERT is that T5 uses no task-specific heads. The decoder's output projection layer is the only output mechanism. This simplifies inference and ensures that all tasks share the same generative interface.

- **Key applications**. Due to its unified text-to-text formulation, T5 is one of the most versatile models in NLP. Key applications include:

  - *Abstractive summarisation*. T5 achieves state-of-the-art performance on the CNN/Daily Mail summarisation benchmark and others.
  - *Machine translation*. With the correct task prefixes, T5 can compete with specialised translation models.
  - *Question answering*. Both extractive and generative QA tasks can be naturally expressed in text-to-text form.
  - *Paraphrasing and text rewriting*. T5 can generate alternative phrasings or stylistic transformations.
  - *Classification tasks*. Sentiment analysis, topic classification and NLI are handled by generating textual labels.
  - *Data-to-text generation*. T5 can verbalise structured data, tables or records.
  - *General-purpose prompting*. As all tasks are text-to-text, T5 can be prompted in a GPT-like manner for many generative tasks.

- **Limitations and challenges**. Despite its versatility, the T5 model has several structural and practical limitations.

  ✕ *High computational cost*. T5's encoder-decoder architecture is expensive to train and deploy, particularly at large scales (3-11 billion parameters).

- ✕ *Autoregressive decoding latency*. Like BART and GPT, T5 generates text one token at a time, which limits its use in real-time applications.
- ✕ *Sensitivity to task prefixes*. Performance can degrade if task prefixes are poorly chosen or inconsistent. This introduces an additional layer of complexity in terms of prompt engineering.
- ✕ *Exposure bias*. As with all autoregressive models, T5 is susceptible to exposure bias ([208]) during generation, which can result in error accumulation in long outputs.
- ✕ *Limited handling of very long sequences*. Although relative position biases help, T5 still inherits the quadratic attention cost of the Transformer.
- ✕ *Potential for hallucination*. T5 can produce fluent but inaccurate text, particularly in open-ended tasks.
- ✕ *Pretraining data quality*. T5 relies heavily on the C4 corpus, which, despite being cleaned, contains noise, biases and web-scale artefacts ([209]) that can propagate into model behaviour.

5.2.5.3. GPT-5 (GPT lineage: 2018–2025)

GPT-5 belongs to the *Generative Pretrained Transformer* (GPT) family developed by OpenAI. This family began with GPT in 2018, followed by GPT-2 in 2019, GPT-3 in 2020, GPT-3.5 in 2022, GPT-4 in 2023, and finally GPT-5.2 in 2025.

These models popularised large-scale autoregressive pretraining and demonstrated that increasing the size of the model, the amount of data and the computing power leads to predictable improvements in performance across a wide range of tasks. GPT-5 continues this lineage as a decoder-only Transformer, trained using next-token prediction and alignment techniques such as supervised fine-tuning and reinforcement learning from human feedback (RLHF), on large-scale text corpora. Its design reflects the broader evolution of large language models towards general-purpose reasoning, instruction following and multimodal integration ([210]).

- **Architecture and modifications relative to the Vaswani Transformer**. GPT-5 retains the decoder-only architecture of the original GPT models, which itself is derived from the Transformer decoder introduced by Vaswani et al. However, there are several important modifications that distinguish GPT-style models from the original architecture. Key architectural differences:

- *Decoder-only stack*. GPT-5 uses only the Transformer decoder, eliminating the need for an encoder. Consequently, the model does not use cross-attention and relies exclusively on masked self-attention. This simplifies the architecture, focusing all capacity on autoregressive generation.

---

[208] Recall that exposure bias is the discrepancy between training and inference in autoregressive models, caused by training the decoder on ground-truth histories while requiring it to generate from its own predictions at inference time.

[209] Web-scale artefacts are systematic distortions that arise from large, heterogeneous web corpora. Examples include duplication, templated content, SEO-driven phrasing (i.e. patterns optimised for search engine ranking), platform-specific conventions and residual markup. These artefacts reflect the properties of the web ecosystem rather than the underlying linguistic or semantic structure. They can influence generation patterns, calibration and stylistic biases, even in downstream tasks that are far removed from web content.

[210] The GPT-5 model is available to all users, including free users, via the main ChatGPT website and API. It is now the default model for all users. Free users can use the core GPT-5 model on the ChatGPT web interface and mobile apps, and a lighter 'GPT-5 Mini' version once daily limits are reached. Paid users (Plus, Pro and Team) receive significantly higher usage limits and access to more powerful variants, such as 'GPT-5 Thinking' and 'GPT-5 Pro', which are designed for complex tasks. The GPT-5 models (GPT-5, GPT-5 Mini and GPT-5 Nano) are available to developers via the OpenAI API for integration into their own applications.

- *Causal self-attention*. As with all GPT models, attention is strictly causal, with each token attending only to previous tokens. This enforces left-to-right generation.
- *Pre-norm residual layers*. LayerNorm is applied before each sublayer (attention or feed-forward), improving training stability for deep networks.
- *Rotary positional embeddings (RoPE)*. Like many modern decoder-only Transformers, GPT-style models use RoPE rather than the sinusoidal encodings of the original Transformer. This technique incorporates positional information into the query and key vectors via a learned rotation in a complex plane. RoPE retains the advantages of sinusoidal embeddings while facilitating better extrapolation to long contexts and enabling more stable autoregressive inference ([S33]).
- *Sparse or optimised attention mechanisms*. GPT-5 incorporates attention optimisations (e.g. FlashAttention and layer-sparse attention [D13]) to reduce memory and computing costs.
- *Large-scale parameterisation*. GPT-5 follows the scaling laws established by earlier GPT models, using very large hidden dimensions, feed-forward widths and attention heads.

- **Differences in training**. The original Vaswani Transformer was trained in a supervised manner for machine translation. In contrast, GPT-5, like all GPT models, uses a different training paradigm.

- *Autoregressive next-token prediction*. GPT-5 is trained to predict the next token based on all previous tokens. This objective is simple yet highly versatile, facilitating broad transfer across tasks.
- *Massive unsupervised pretraining*. GPT-style models are trained on large, diverse corpora containing books, web text, code and other types of text. This contrasts with the task-specific parallel corpora used for the original Transformer.
- *Instruction tuning*. After pretraining, GPT-5 undergoes supervised fine-tuning on instruction-following datasets to improve usability and alignment.
- *Reinforcement learning from human feedback (RLHF)*. Human preference data is used to shape the model's behaviour, thereby improving safety and conversational quality.
- *Continued scaling*. GPT-5 adheres to the empirical scaling laws discovered by OpenAI. These laws state that the performance of LLMs improves in a predictable power-law relationship as the size of the model, the size of the dataset, and the amount of computing power used for training increase ([K13]). The GPT-5 model is trained on more data, with more parameters, and uses more computing power than earlier models.

- **Differences in inference**. The way GPT-5 makes inferences differs from the original Transformer in several fundamental ways.

- *Decoder-only inference*. The Vaswani Transformer uses an encoder-decoder pipeline for translation. GPT-5, however, uses only the decoder, thus simplifying the interface.
- *Unified prompting interface*. GPT-5 uses a single interface for all tasks. The input is a prompt, and the output is generated text. No task-specific heads or architectural changes are required.
- *Long-context inference*. GPT-style models support context windows that are much longer than the original Transformer's limits. This enables document-level reasoning, multi-turn dialogue, and retrieval-augmented generation (RAG).
- *Advanced decoding strategies*. GPT-5 uses modern decoding techniques, such as:
    - *Nucleus sampling*. In nucleus sampling, the next token is selected from the smallest set of tokens whose cumulative probability exceeds a given threshold. This ensures diversity while avoiding words with extremely low probability.

- *Temperature scaling*. This technique adjusts the model's logits to produce a sharper (low temperature) or more random (high temperature) output distribution, thereby controlling the balance between creativity and determinism.
- *Beam search* (less common). Beam search maintains several candidate sequences ('beams') simultaneously and expands the most promising ones. This approach aims to identify a high-probability output rather than making a greedy local choice.
- *Contrastive decoding*. This technique selects tokens that are likely to be included in the model and that differ from common, high-probability continuations. This reduces repetition and improves fact accuracy.
- *Speculative decoding for speed*. Speculative decoding speeds up the generation process by using a smaller 'draft' model to suggest several tokens at once, which the larger model then verifies or corrects in one step.

These methods are more advanced than the simple beam search used in the original Transformer.

- **Key applications**. Like its predecessors, GPT-5 is a general-purpose generative model. Its applications span almost all areas of NLP and beyond.

- *Conversational agents and assistants*. Natural dialogue, reasoning, and instruction following.
- *Content generation*. Articles, summaries, explanations, and creative writing.
- *Code generation and analysis*. Autocompletion, debugging, refactoring and documentation.
- *Question answering and reasoning*. Open-domain QA, multi-step reasoning, and chain-of-thought prompting.
- *Translation and multilingual tasks*. High-quality translation without task-specific training.
- *Data-to-text generation*. Structured data verbalisation and report generation.
- *Retrieval-augmented workflows*. Retrieval-augmented generation (RAG) is a technique used by LLMs to connect to external knowledge bases and retrieve relevant information before generating a response. This integration with external knowledge sources helps to ensure factual accuracy.
- *Tool use and agentic behaviour*. Planning, calling APIs and orchestrating multi-step tasks.

- **Limitations and challenges**. Despite its capabilities, GPT-5 inherits several limitations characteristic of large autoregressive models.

✕ *Exposure bias*. As with all autoregressive decoders, GPT-5 is susceptible to training–inference mismatch, which leads to error accumulation in long sequences.

✕ *Hallucination*. GPT-style models may produce fluent yet factually inaccurate statements, particularly in open-ended tasks where the model is required to generate free-form text with no constraints on length, format or content.

✕ *High computational cost*. Training and inference require substantial computing power, memory and specialised hardware.

✕ *Latency from autoregressive decoding*. Token-by-token generation limits real-time applications.

✕ *Sensitivity to prompting*. Performance can vary significantly depending on the phrasing of the prompt, the length of the context and the formatting.

✕ *Limited interpretability*. The internal representations of large decoder-only models are still hard to analyse or control.

✕ *Long-context degradation*. Even with extended context windows, performance may deteriorate for very long inputs due to attention scaling and memory limitations. 'Very

long' refers to the point at which the model can technically accept the input, but its ability to maintain coherence, recall earlier content or attend uniformly across the sequence begins to weaken. ([211])

5.2.5.4. DeepSeek-V3 (2024)

DeepSeek-V3 is the latest large-scale language model developed by DeepSeek-AI, a Chinese artificial intelligence company based in Hangzhou, Zhejiang. The company is formally registered as Hangzhou DeepSeek Artificial Intelligence Basic Technology Research Co., Ltd.

Introduced in late 2024, DeepSeek-V3 was formally documented in the *DeepSeek-V3 Technical Report* ([D14]), which was authored by a large team of researchers, including Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang and many others. Building on the architectural innovations of DeepSeek-V2, the model aims to deliver performance comparable to that of leading closed-source models while maintaining cost-efficient training and inference ([212]).

- **Architecture and modifications relative to the Vaswani Transformer**. DeepSeek-V3 remains fundamentally a Transformer-based model, but retains only the decoder part of the original Transformer introduced by Vaswani et al. Moreover, several major architectural innovations distinguish DeepSeek-V3 models from the original design.

A DeepSeek-V3 decoder layer comprises three tightly integrated sublayers: Multi-Head Latent Attention (MLA), a Mixture-of-Experts (MoE) feed-forward module and residual-norm stabilisation, which makes training more efficient at scale. These components enable DeepSeek-V3 to support extremely large parameter counts, long-context reasoning and efficient inference.

- *Multi-Head Latent Attention (MLA)*. The MLA forms the attention mechanism in DeepSeek-V3, replacing the standard masked self-attention mechanism in every decoder layer. Its design enables the model to process information more efficiently, performing attention computations in a shared, low-dimensional latent space. During training, the MLA behaves like a fully parallelisable masked attention mechanism, making it compatible with the standard Transformer training pipeline. During autoregressive inference, MLA serves as a drop-in replacement for conventional masked self-attention, offering substantially reduced computational and memory overhead due to its compact key-value representations.

  MLA reduces computational overhead while maintaining high accuracy, making it ideal for large-scale language models. The term 'latent' refers to the fact that the attention mechanism operates in a learned, lower-dimensional hidden space, which compresses the key and value representations. This approach minimises computation and memory usage while preserving essential structure.

  Let an input sequence be tokenised and embedded as a matrix $X \in \mathbb{R}^{n\times d}$, where n is the sequence length and d is the model dimension ([213]). In standard multi-head attention (MHA), each head has its own full projections

$$W_Q^{(i)}, W_K^{(i)}, W_V^{(i)} \in \mathbb{R}^{d\times d_k}, i = 1,2,\dots,h,$$

  where h is the number of attention 'heads', i.e. $d_k = d/h$. These transformations are applied to the input matrix X yielding the set of $n \times d_k$ matrices: $Q^{(i)} = XW_Q^{(i)}$, $K^{(i)} = XW_K^{(i)}$ and $V^{(i)} = XW_V^{(i)}$, $i = 1,2,\dots,h$ (see Section 5.2.2.2). This results in a high parameter count and a large KV cache during inference.

---

[211] Although the context window length of GPT-5 has not been publicly disclosed, it is expected to be substantially larger than that of previous GPT models. It is likely to be in the range of hundreds of thousands or millions of tokens, reflecting ongoing advances in long-context attention mechanisms.

[212] The DeepSeek V3 platform is widely available to all users via several channels. These include a free, full-featured web chat, an open-source download for local use and a pay-as-you-go API for developers. 'Pay-as-you-go' is a usage-based service model in which developers access computational resources billed according to actual usage (e.g., the number of requests, tokens processed, or compute time).

[213] X does not contain positional encodings. Instead, positional information is introduced inside the MLA mechanism.

MLA reduces computational and memory costs by projecting token representations into a single shared low-dimensional latent space, which is computed once per layer and reused by all attention heads. Instead of computing separate full-dimensional key and value projections for each head directly from X, MLA factorizes these projections through a common latent basis. Formally, the shared latent representation is computed as

$$L = XW_L \in \mathbb{R}^{n \times d_l},$$

where $W_L \in \mathbb{R}^{d \times d_l}$. The matrix L is shared across all heads and serves as a compressed basis for subsequent head-specific key and value projections. The latent dimension d_l is chosen such that $d_l \ll d$, ensuring that the projection from the model space into the latent space is computationally efficient. Although d_l may be similar in scale to the per-head dimension d_k, the essential reduction occurs relative to the full model dimension d. DeepSeek-V3 does not publicly disclose the latent dimension d_l used in its MLA mechanism. However, an analysis of the available model configuration files suggests that $d_l \approx 64 - 128$.

While the latent factorisation is structurally identical in both regimes, its computational implications differ between training and inference. During training, the full sequence is processed in parallel, and no KV cache is maintained, meaning that MLA behaves as a structured, low-rank reparametrisation of masked multi-head attention. In contrast, during autoregressive inference, previously computed representations are cached, and the latent formulation significantly reduces memory complexity. The following sections analyse the MLA mechanism in training and inference modes separately.

During training, the model processes all tokens in the sequence simultaneously. For the input matrix $X \in \mathbb{R}^{n \times d}$, a single low-dimensional latent representation $L = XW_L \in \mathbb{R}^{n \times d_l}$ is computed. Prior to the attention computation, the keys and values for each attention head i are reconstructed from the latent representation using upward projection matrices

$$K^{(i)} = LU_K^{(i)} \in \mathbb{R}^{n \times d_k}, V^{(i)} = LU_V^{(i)} \in \mathbb{R}^{n \times d_k},$$

where $U_K^{(i)}, U_V^{(i)} \in \mathbb{R}^{d_l \times d_k}$ and $i = 1,2, \dots, h$. This effectively factors the original MHA projection matrices

$$W_K^{(i)} = W_L U_K^{(i)}, \quad W_V^{(i)} = W_L U_V^{(i)},$$

which corresponds to a low-rank decomposition with rank at most d_l. Queries are usually calculated directly from the full hidden state

$$Q^{(i)} = XW_Q^{(i)}, \; W_Q^{(i)} \in \mathbb{R}^{d \times d_k}.$$

As queries do not require caching during autoregressive decoding, they do not create the same memory bottleneck as keys and values.

DeepSeek-V3 then applies RoPE to $Q^{(i)}$ and $K^{(i)}$ to encode positional information ([214])

$$Q_r^{(i)} = RoPE(Q^{(i)}), \; K_r^{(i)} = RoPE(K^{(i)}).$$

For each head, attention proceeds as usual

$$A^{(i)} = softmax\left(\frac{Q_r^{(i)}\left(K_r^{(i)}\right)^T}{\sqrt{d_k}}\right) \in \mathbb{R}^{n \times n}.$$

[214] Rotary Positional Embeddings (RoPE) encode the position of each token by applying a rotation, dependent on position, to each query and key vector. More precisely, for a given head, RoPE rotates each 2D pair of Q and K components by a position-dependent complex phase whose angle increases with the token index. This allows the dot product between two RoPE-transformed vectors to depend only on their relative distance, giving the model an intrinsic sense of relative position (see [S33] for more information on RoPE).

Then the attention weights are applied to the values

$$Z^{(i)} = A^{(i)}V^{(i)} \in \mathbb{R}^{n\times d_k}.$$

This is the only stage where token representations are mixed. The outputs from all heads are concatenated

$$Z = \text{Concat}\left(Z^{(1)}, Z^{(2)}, \dots, Z^{(h)}\right) \in \mathbb{R}^{n\times d}.$$

After concatenation, MLA applies the same output projection as standard multi-head attention

$$MLA(X) = ZW_O \in \mathbb{R}^{n\times d},$$

where $W_O \in \mathbb{R}^{d\times d}$. Thus, the downstream computation is the same as standard MHA, with only the parameterisation of $K^{(i)}$ and $V^{(i)}$ being different.

In summary, in training mode, MLA can be viewed as a low-rank factorisation of the key and value projection matrices with a shared latent basis. This reparameterisation preserves the standard attention computation. While MLA is primarily driven by the need to reduce inference-time memory, in training mode it behaves mathematically as a structured low-rank decomposition of multi-head attention.

During the inference process, tokens are generated one at a time and the KV cache becomes essential. The crucial innovation here is that MLA never stores high-dimensional keys or values in the KV cache. This is why the MLA's KV cache is so small. As discussed in Section 5.2.3.3, there is an asymmetry between queries and keys/values in autoregressive decoding. When processing the current token, the model only uses the query values that depend on it; the previous queries are not reused. Therefore, queries are computed afresh at every decoding step and are not cached. This process begins with a low-rank projection onto the latent space. For each newly generated token X_{i-1}, the layer computes

$$L_{i-1} = X_{i-1}W_L,$$

where $W_L \in \mathbb{R}^{d\times d_l}$. The latent vector L_{i-1} is appended to the KV cache. Attention is computed using the current query Q_{i-1}, all keys $K_0, K_1, \dots, K_{i-1}$, and all values $V_0, V_1, \dots, V_{i-1}$, which are reconstructed from the KV cache using upward projection matrices ([215]).

In summary, the MLA alters the construction of Q, K and V. However, once the per-head outputs $Z^{(i)}$ have been computed, the rest of the pipeline is functionally equivalent to that of the standard MHA. Therefore, from an autoregressive generation perspective, the MLA is a drop-in replacement for the standard decoder's masked MHA, offering a much more efficient internal representation.

- *Mixture-of-Experts Feed-Forward Network (MoE-FFN)*. DeepSeek-V3 replaces the standard dense feed-forward network (FFN) with a mixture-of-experts feed-forward network (MoE-FFN), enabling the model to increase its number of parameters without proportionally raising the computational cost. Rather than applying a single FFN to every token, the MoE-FFN contains a set of p independent expert networks, with a learned router selecting a small subset of these experts for each token. Only the selected experts are executed, resulting in sparse activation of the layer (see [B32], [D15], and [S33] for more details).

[215] In optimised MLA implementations, the head-specific 'up-projections' do not need to be explicitly materialised as full per-head matrices. Instead, the factorised structure can be incorporated into the subsequent attention computation. This avoids the need to construct full $K^{(i)}$ and $V^{(i)}$ matrices in memory, thereby achieving the intended reduction in memory bandwidth and activation footprint, while preserving mathematical equivalence to the unfused formulation.

The MoE-FFN consists of a router R (also known as a *gating network*) and a set of experts $\{E_1, E_2, \dots, E_p\}$. Each expert E_j is a standard two-layer feed-forward network

$$E_j(\cdot) = W_2^{(j)} f\left(W_1^{(j)}(\cdot)^T + B_1^{(j)}\right) + B_2^{(j)},$$

with a hidden dimension d_{ff} and a nonlinearity f, which is typically *GELU* (see Section 5.2.3.1). The router is implemented as a single-layer feed-forward neural network, parameterized by a weight matrix $W_R \in \mathbb{R}^{p \times d}$, followed by a softmax, producing routing probabilities over experts.

The input to the MoE-FFN is the hidden-state matrix $Y = MLA(X) \in \mathbb{R}^{n \times d}$, where the row $Y_i \in \mathbb{R}^d$ $(i = 1, 2, \dots, n)$ corresponds to the token at position i. The router computes a vector of expert scores for Y_i

$$Router(Y_i) = W_R Y_i + B_R \in \mathbb{R}^p,$$

followed by

$$R_i = softmax\big(Router(Y_i)\big) = \big[r_{i1}, r_{i2}, \dots, r_{ip}\big],$$

where r_{ij} is the routing weight that the router assigns to expert E_j for the token representation Y_i. Rather than using all experts, DeepSeek-V3 employs *top-k sparse routing* (with $k \ll p$), selecting the index set

$$J_i = TopK(R_i; k) \subseteq \{1, 2, \dots, p\}.$$

DeepSeek-V3 typically uses $k = 2$ (i.e. J_i has only two elements), meaning that precisely two experts process each token. These two experts have the highest routing weights ([216]).

The MoE-FFN output is then

$$MoE(Y_i) = \sum_{j \in J_i} r_{ij}\, E_j(Y_i),$$

where only the selected experts are evaluated.

Sparse routing can lead to an imbalance of experts, whereby only a small number of experts receive the majority of tokens. To mitigate this issue, DeepSeek-V3 uses an additional load-balancing loss function to encourage the use of all experts uniformly. Other methods include applying capacity constraints to limit the number of tokens per expert, and noisy routing, which involves adding small noise to logits during training. These mechanisms ensure that all experts receive sufficient gradient updates and that the model remains stable during large-scale training.

In summary, the MoE-FFN replaces a single dense feed-forward network with a mixture of sparsely activated expert networks. A learned router selects the top k experts for each token, computes their outputs and combines them using routing weights. This design dramatically increases model capacity while keeping per-token computation almost the same ([217]).

- *Residual connections and normalisation*. As with modern Transformer variants, each sublayer is wrapped in residual connections and normalisation layers to stabilise

---

[216] In practice, this 'highest-weight' selection is conditional, since each expert has a finite capacity per batch. Therefore, if one of the top two experts is already at capacity, the token is either routed to the next-highest-weight expert, or is dropped for that expert (depending on the implementation).

[217] Although GPT-5 also employs a Mixture-of-Experts architecture, its use of MoE differs from that of DeepSeek-V3. DeepSeek-V3 activates only a small subset of its experts per token to minimise compute and memory cost, whereas GPT-5 uses a more compute-intensive MoE configuration with adaptive routing designed to maximise model performance rather than efficiency. As a result, both models rely on sparse expert activation, but they optimise the mechanism for different objectives: DeepSeek-V3 for cost-efficient scalability, GPT-5 for frontier-level accuracy. The internal architecture of Microsoft Copilot has not been publicly disclosed. Although MoE architectures are a plausible scaling strategy for large-scale deployment, there is currently no evidence to confirm that Copilot uses MoE.

optimisation at scale. DeepSeek-V3 uses a pre-norm configuration, applying normalisation before attention and MoE-FFN computations. This improves gradient flow in extremely deep networks and reduces training instability.

- **Differences in training**. The training pipeline of DeepSeek-V3 diverges from that of the original Transformer. Since only a small number of experts are active per token, training requires more complex optimisation due to the uneven flow of gradients across the experts. To guarantee dense gradient paths, DeepSeek-V3 requires careful router initialisation and shared experts. Therefore, training MoE models is significantly more challenging than training dense Transformers. In summary, the Vaswani Transformer optimises a dense, static computation graph, whereas DeepSeek-V3 optimises a sparse, input-dependent computation graph whose structure is learned jointly with the model parameters.

- **Differences in inference**. Although both models perform autoregressive or encoder-decoder inference using self-attention, their inference dynamics differ fundamentally due to factors such as sparsity, routing and system-level execution. Notably, the non-trivial control flow and communication overhead present in DeepSeek-V3 inference are absent in the original Transformer. Furthermore, although DeepSeek-V3's autoregressive decoding involves per-token routing decisions that dynamically select a subset of experts, these decisions do not constitute online learning. The model parameters remain fixed during inference, and no gradient-based updates occur. In summary, the Vaswani Transformer's inference process entails evaluating a fixed dense network, whereas DeepSeek-V3's inference process consists of dynamically instantiating a sparse subnetwork for each token via learned routing.

- **Key applications**. The DeepSeek-V3 is best understood as a general-purpose, frontier-scale MoE language model. Its applications stem directly from its characteristics of high capacity, strong reasoning and efficient inference.

  - *Long-context reasoning and document understanding*. The latent-space attention and RoPE-based positional encoding of DeepSeek-V3 enable efficient processing of very long sequences. This makes it well-suited to tasks such as legal analysis, scientific literature review and multi-document synthesis, where models must track dependencies across tens or hundreds of thousands of tokens.

  - *Code generation and software engineering*. The model's mixture-of-experts architecture provides significant effective capacity without additional computational cost, enabling it to recognise a variety of programming patterns, libraries, and idioms. This makes DeepSeek-V3 particularly effective at code completion, refactoring and multi-file reasoning.

  - *Mathematical and symbolic problem solving*. The DeepSeek-V3 model can handle multi-step derivations, algebraic transformations and structured symbolic reasoning more reliably than dense models with comparable computing power.

  - *Knowledge-intensive question answering*. Thanks to its large effective parameter count and long-context capabilities, the model is well-suited to retrieval-augmented generation, open-domain QA and tasks requiring the synthesis of information from multiple sources.

  - *Domain-specific adaptation (e.g., finance, medicine, law)*. As only a subset of experts are activated per token, fine-tuning enables certain experts to be specialised without degrading general performance. This allows for efficient adaptation to specialised domains with relatively small datasets.

- **Limitations and challenges**. Despite its strong performance and highly efficient design, the DeepSeek-V3 also has several structural and practical limitations arising from its architecture, training regime and deployment constraints.

  ✕ *Complexity of the MoE architecture*. DeepSeek-V3 relies on a large-scale mixture-of-experts system comprising hundreds of experts and top-k routing. While this design dramatically improves parameter efficiency, it also introduces significant engineering challenges. Expert imbalance, routing instability and capacity saturation must be carefully

managed using auxiliary losses and capacity constraints. Without these mechanisms, the model risks experiencing expert collapse, degraded specialisation or unstable training dynamics

- ✕ *Latent-space attention adds architectural overhead*. The Multi-Head Latent Attention (MLA) mechanism reduces KV-cache size and improves long-context efficiency, but it also introduces additional projection layers and latent-space transformations. These components increase architectural complexity and require careful tuning to prevent information bottlenecks. In particular, compressing keys and values into a shared latent space can limit expressiveness if the latent dimension is set too restrictively.
- ✕ *High total parameter count*. Although only a small number of parameters are active per token, the complete model comprises over 600 billion parameters. This creates significant storage and distribution requirements during training and inference. Efficient partitioning, placement of experts, and communication strategies are essential, and the model is difficult to deploy on smaller clusters or off-the-shelf hardware.
- ✕ *Training stability and optimisation complexity*. DeepSeek-V3 integrates multiple training objectives, such as next-token prediction and multi-token prediction. Balancing these objectives requires careful scheduling and hyperparameter tuning. Combining MoE, MLA and multi-objective training increases the risk of optimisation instabilities, particularly at very large scales.
- ✕ *Limited transparency of expert specialisation*. Although MoE architectures encourage emergent specialisation, the resulting expert behaviours are often opaque. It remains an open research problem to understand which experts handle which linguistic, semantic, or reasoning patterns. This lack of interpretability complicates debugging, safety evaluation and targeted fine-tuning.

5.2.5.5. Microsoft Copilot: application-level system (2023)

Introduced by Microsoft in 2023, Copilot is an application-level AI companion built on advanced Transformer-based language models ([218]). An AI companion is an application-level system that integrates large language models, safety mechanisms and interaction design to provide a conversational, adaptive and collaborative user experience. Copilot builds on the broader evolution of large-scale language models and conversational AI by integrating advances in natural language processing. Unlike research-oriented models such as BERT or T5, Copilot has been specifically developed as a user-facing assistant focused on clarity, adaptability, and collaborative problem-solving.

- **Architecture and modifications relative to the Vaswani Transformer**. Although Copilot is built on state-of-the-art large language models, no technical details have been publicly disclosed about its internal architecture, model size or training data. This includes information on whether the underlying model uses a decoder-only architecture, an encoder-decoder architecture, a hybrid architecture, or a modified Transformer variant. What we do know is that Copilot is powered by a combination of Microsoft technologies and advanced large language models, including those from OpenAI. Microsoft's Prometheus framework integrates Bing search signals with these models to provide more grounded, up-to-date, contextually relevant responses. In any case, Copilot inherits the general Transformer lineage, incorporating several design principles that are tailored for real-world interaction:

- *Advanced reasoning capabilities*. Copilot is designed to support multi-step thinking, explanation and synthesis.

[218] Copilot is publicly available on all major platforms, including Windows, macOS, the web, iOS, Android and Edge. It can be accessed for free by anyone who chooses to install or enable it. In addition to this free version, Microsoft offers paid tiers, such as Microsoft 365 Copilot, which integrates advanced AI capabilities into the Microsoft 365 productivity suite, and Copilot Pro, which provides enhanced features for individual users.

- *Instruction-following behaviour*. Optimised to interpret user intent and respond helpfully across diverse tasks.
- *Safety-aligned conversational design*. Guardrails ([219]) are integrated to ensure responsible behaviour, factual grounding and user well-being.
- *Multimodal integration*. Depending on the platform, Copilot can work with text, images, voice and other modalities.

These modifications are designed to enhance usability, reliability and collaborative interaction rather than optimising for a single benchmark task.

- **Differences in training**. Although the specific training procedures behind Copilot remain undisclosed, its conceptual training paradigm differs significantly from that of the original Vaswani Transformer.

- *Copilot is not trained for a single task*. While the original Transformer was trained for machine translation, Copilot is designed to support a broad range of tasks and conversational settings.
- *Instruction-tuned behaviour*. Copilot is optimised to follow natural language instructions, explain reasoning and adapt to user goals.
- *Safety-aligned training*. It incorporates alignment techniques to ensure responsible behaviour and avoid harmful outputs.
- *Continually updated knowledge via integrated search*. Copilot can retrieve up-to-date information through web searches, enabling it to provide responses that are grounded in current data.

- **Differences in inference**. The Copilot inference process differs from the original Transformer model in several important ways.

- *Conversational inference*. It maintains context across turns, adapts to user preferences and supports multi-step reasoning.
- *Instruction-following decoding*. Responses are shaped by user intent, safety constraints and conversational norms rather than solely by probability maximisation.
- *Integrated web search*. It can retrieve fresh information when needed, which the original Transformer was not designed to do.
- *Multimodal inference*. Depending on the platform, Copilot can interpret or generate images or respond to voice input.
- *Adaptive response style*. It adjusts the tone, structure and depth of its responses based on the user's goals and communication style.

Therefore, inference is not just sequence generation – it is interactive, contextual and user-centred.

- **Key applications**. Copilot has a variety of applications in different fields, primarily focusing on increasing productivity, automating tasks and boosting creativity. Designed as a general-purpose AI companion, it supports:

- *Knowledge synthesis and explanation*. Clarifying complex ideas, summarising information, and providing structured insights.
- *Creative and analytical writing*. Drafting, editing, refining and structuring text across genres and domains.

---

[219] Guardrails for LLMs are safety mechanisms and rule-based controls that restrict the behaviour of AI systems. Operating as a layer between the user and the model, they ensure that outputs are safe, reliable, ethically aligned and compliant with specific policies.

- *Problem-solving and reasoning*. Breaking down complex tasks, exploring alternatives and supporting decision-making.
- *Educational support*. Guiding learning, generating examples and helping users to understand difficult concepts.
- *Multimodal tasks*. Interpreting and generating images and interacting through voice (depending on the platform).
- *Conversational interaction*. Engaging in meaningful dialogue that adapts to the user's style and goals.
- **Limitations and challenges**. Despite its versatility, Copilot has some significant limitations and challenges related to accuracy, data security, memory usage, and general usability. Users need to be aware of these issues to use the tool effectively.

✕ *There are no publicly disclosed architectural details*. The internal model architecture, size and training data are not revealed.

✕ *No access to personal data unless explicitly provided*. Copilot cannot recall or access private information outside the current conversation unless the user enables memory.

✕ *Not a substitute for professional advice*. Copilot cannot diagnose medical, legal or financial issues.

✕ *Contextual understanding*. It may struggle with nuanced or ambiguous queries, leading to off-target responses if the user prompt is not specific and clear.

✕ *Hallucinations*. Like all LLMs based generative AI tools, it occasionally produces responses that sound confident and authoritative but are factually incorrect, misleading, or entirely fabricated. Users report instances of it failing simple math tasks in Excel or providing wrong information, necessitating human review and fact-checking for all high-stakes outputs.

✕ *Dependence on user intent*. Copilot is most effective when the user provides context, goals, or constraints.

✕ *Safety constraints*. Copilot avoids generating harmful, unsafe or inappropriate content, which may limit its responses in certain domains. These limitations are intentional safeguards designed to prevent harm.

✕ *Compliance concerns*. Organizations in regulated industries must conduct thorough risk assessments to ensure Copilot's data handling aligns with laws like GDPR.

✕ *Memory limitations*. Copilot's memory is often short-term; it may not retain context across different chat sessions.

For clarity and ease of comparison, the following table synthesises the main differences and shared features of BERT, BART, T5, GPT-5, DeepSeek-V3, and Copilot across the dimensions of architecture, training, inference, applications, and limitations.

Model	Architecture	Training Objective	Inference Characteristics	Typical Applications	Key Limitations	Knowledge cutoff
BERT (2018)	Encoder-only Transformer	Masked Language Modelling (MLM) + Next Sentence Prediction (NSP)	Bidirectional encoding; no generative decoding; used mainly for feature extraction	Classification, QA, NER, sentence similarity, embedding extraction	Not generative; limited to fixed-length inputs; NSP later shown to be suboptimal	2018
BART (2019)	Encoder–decoder (seq2seq) combining BERT-like encoder and GPT-like decoder	Denoising autoencoding with multiple corruption schemes	Flexible seq2seq generation; strong for text reconstruction and summarisation	Summarisation, translation, paraphrasing, text generation	More complex training; heavier inference than encoder-only models	2018
T5 (2020)	Unified encoder–decoder "text-to-text" Transformer	Span corruption + task prefixing; everything cast as text-to-text	Highly flexible; all tasks framed as generation; strong generalisation	Translation, summarisation, QA, reasoning, multi-task learning	Large computational footprint; performance sensitive to prompt formulation	2019
GPT-5.2 (2025)	Decoder-only Transformer (autoregressive)	Next-token prediction at scale	Strong long-range generation; multi-step reasoning; conversational inference	Open-ended generation, coding, reasoning, dialogue, content creation	Context-length degradation; hallucination risk; opaque training details	August 31, 2025
DeepSeek-V3 (2024)	Decoder-only Transformer, MoE + MLA (latent attention)	Next-token prediction, multi-token prediction, and distillation	Extremely efficient MoE; latent-space attention enables long contexts	Long-context reasoning, code, multilingual tasks, efficient large-scale inference	Requires careful routing balance; latent attention adds architectural complexity	2024
Copilot (2023)	Application-level system built on advanced LLMs (architecture not publicly disclosed)	Not a pre-training paradigm; uses underlying LLMs + Microsoft orchestration and safety layers	Conversational, adaptive, multimodal; integrates web search; instruction-following with safety constraints	Knowledge synthesis, writing, reasoning, planning, multimodal tasks; integrated productivity assistance	Safety constraints in medical, legal, financial, political, and high-risk domains; no disclosed architecture; cannot replace professional advice	Not disclosed; uses real-time search

A knowledge cutoff is the point in time after which a model's training data no longer includes any new information. This restricts the model's ability to account for events or facts that occurred after that date. However, Copilot integrates real-time Bing search via Microsoft's Prometheus framework. This means that it does not rely exclusively on a static training corpus, but can supplement its responses with information retrieved from external sources. Similarly, modern GPT-5 series models, including GPT-5.2, can operate in Deep Research mode, dynamically accessing and synthesising up-to-date web sources during inference. In such configurations, the practical limitation imposed by the training-time knowledge cutoff is mitigated by retrieval-augmented generation. Nevertheless, the underlying model parameters themselves remain fixed and are not updated online.

5.2.6. Prompt engineering

Prompt engineering is the systematic design of input instructions (prompts) for large language models (LLMs) to produce reliable, high-quality, and contextually appropriate outputs. Unlike classical natural language processing (NLP) pipelines, where behaviour is largely determined by explicitly programmed architectures and training objectives ([220]), LLM-based systems expose a significant portion of their functional behaviour through the prompt itself. The prompt acts as a soft interface between the user and a fixed, pre-trained (or lightly fine-tuned) model. Consequently, prompt engineering bridges the gap between human intent and model behaviour, enabling users to shape outputs without modifying the model's parameters. In this sense, prompt engineering can be considered a type of zero-shot or few-shot programming, where natural language instructions serve as a high-level interface to a complex generative system.

[220] In classical NLP pipelines, such as CRF-based sequence taggers, TF–IDF classifiers with linear models, or task-specific neural architectures like BiLSTM-CRF systems, model behaviour is largely fixed by the chosen architecture, feature design, and training objective. These pipelines have little or no capacity for inference-time behavioural control.

Although LLMs such as GPT-5 and Microsoft Copilot are trained on extensive data sets and demonstrate robust generalisation capabilities, their behaviour is essentially conditional. The model calculates a probability distribution for potential continuations based on the prompt. Consequently, the structure, clarity and constraints encoded in the prompt directly influence the model's internal inference trajectory. From a practical perspective, the key lies in asking questions in a way that aligns the model's learned representations, inductive biases, and decoding behaviour with the user's intent.

This subchapter introduces the conceptual foundations of prompt engineering, offering practical guidance on how to interact effectively with LLMs, such as ChatGPT and Microsoft Copilot.

5.2.6.1. General principles of prompt engineering

LLMs do not "understand" user intent in the same way as humans do; they infer intent from linguistic cues. Even small changes in phrasing can alter the model's internal representation, thereby changing its output distribution.

Formally, an autoregressive LLM defines a conditional probability distribution

$$p(y_1, y_2, \dots, y_m \mid x_1, x_2, \dots, x_n) = \prod_{j=1}^{m} p\left(y_j \mid x_1, x_2, \dots, x_n;\ y_1, y_2, \dots, y_{j-1}\right)$$

where $[x_1, x_2, \dots, x_n]$ is the input prompt and $[y_1, y_2, \dots, y_m]$ is the generated output. Prompt engineering consists of constructing $[x_1, x_2, \dots, x_n]$ such that the induced conditional distribution assigns high probability mass to sequences consistent with the desired task, style and constraints. Crucially, the prompt not only specifies what the model should do but also implicitly determines how it reasons. Because transformers attend to the entire prompt via self-attention, the structure, ordering, and explicitness of instructions can significantly affect internal activations and downstream token probabilities.

From a mathematical perspective, the prompt acts as a conditioning prefix that shapes the model's hidden states. In a Transformer, it is the prompt that determines the initial sequence of key, query and value vectors. This process is entirely driven by the token embeddings and positional encodings applied at the input of each Transformer layer. These, in turn, influence the attention weights, which determine how the model integrates information across tokens. Consequently, a well-designed prompt can steer the model's internal computations by altering the structure of its attention matrix.

In summary, a prompt does not set queries, keys and values directly; rather, it determines them indirectly through token embeddings, which are the model's input. As we know, each prompt token is mapped to an embedding vector, augmented with positional information, forming the initial hidden states. At every attention layer, these hidden states are linearly projected via fixed matrices into queries, keys and values. Therefore, prompt engineering shapes the initial representation space from which all subsequent Q, K and V vectors are derived, thereby influencing attention patterns and behaviour at inference time without modifying model parameters (see [C15], [J8], [L18], [P19], and [T10] for more details on prompt engineering).

5.2.6.2. Key aspects of prompt engineering

There are various reasons why prompt engineering is important:

- **Parameter immutability**. In most deployments, users cannot modify the model weights, so the prompt becomes the main control mechanism.

- **Task generality**. LLMs are trained on diverse objectives. Prompts clarify which latent capability (e.g. summarisation, reasoning or translation) should be activated.

- **Error reduction**. Well-designed prompts reduce hallucinations, ambiguity and irrelevant content.

- **Efficiency**. Better prompts often reduce the need for lengthy conversations, retries or post-processing.

From an information-theoretic perspective, a prompt minimises uncertainty by limiting the model's posterior distribution of outputs. Poor prompts correspond to under-specified conditioning contexts, resulting in high entropy and unstable generation.

Several technical aspects recur across effective prompts.

- **Role and instruction specification**. Explicitly assigning a role and clearly stating the task instructions biases the model towards a specific region of its learned conditional distribution. From a modelling perspective, role statements (e.g. "*You are an expert in numerical analysis*" or "*Act as a scientific editor*") serve as high-level priors, activating representations associated with specific discourse styles, domains and norms acquired during instruction tuning. Clear task instructions further reduce ambiguity by constraining the space of admissible outputs and lowering the entropy of the model's posterior distribution over responses. In practice, prompts that combine an explicit role with unambiguous instructions produce more stable, coherent and goal-aligned outputs than prompts that rely on implicit assumptions or underspecified intent.

- **Chain-of-thought (CoT) prompting**. This encourages the model to generate explicit intermediate reasoning steps before providing a final answer. Rather than modelling the conditional distribution $p(Y \mid X)$ directly, the prompt induces an auxiliary reasoning sequence R, yielding

$$p(Y, R \mid X) = p(R \mid X) \cdot p(Y \mid X, R)$$

From a Transformer perspective, the reasoning tokens in R form part of the conditioning context on which self-attention operates. This shapes the hidden states and attention patterns that influence subsequent token predictions. In this way, CoT biases the model towards multi-step inference patterns learned during pre-training and instruction tuning. This technique is particularly effective for tasks involving compositional structure, arithmetic or logical dependencies, where premature commitment to an answer is a common cause of failure.

- **Task decomposition**. Complex tasks can be broken down into steps, encouraging the model to allocate attention sequentially and reducing error propagation. This improves reasoning quality (a principle related to CoT prompting).

- **Contextual grounding**. Contextual grounding involves explicitly including definitions, assumptions, background information or reference material within the prompt, thereby anchoring the model's generation to a well-defined semantic context. In formal terms, grounding augments the conditioning sequence X with task-relevant context C, restricting the space of plausible outputs. From a Transformer perspective, grounded context tokens fully participate in self-attention, influencing hidden-state representations. This reduces reliance on implicit assumptions and mitigates hallucinations, particularly in technical or domain-specific tasks. Thus, effective contextual grounding serves as an inference-time alignment mechanism that constrains generation by explicitly specifying the informational basis on which the output should be constructed.

- **Output constraints**. These constraints refer to the explicit specification of structural, stylistic or formal requirements for the model's generated response, such as length limits, formatting rules or required components. Constraints expressed in the prompt are encoded as tokens that influence attention patterns and hidden-state representations. As an inference-time control mechanism, output constraints guide the decoding process by explicitly defining the acceptable form of outputs, thereby aligning probabilistic generation with task-specific expectations.

- **Examples (few-shot prompting)**. Including input-output pairs approximates Bayesian conditioning on a task-specific dataset, which often yields significant performance improvements. Few-shot prompting can be interpreted as augmenting the prompt with

empirical evidence. This shifts the model towards a conditional distribution that more closely matches the mapping demonstrated by the provided examples.

- **Iterative refinement**. Prompt engineering is an inherently interactive process in which users refine prompts based on model outputs.

- **Language of the prompt**. While prompt engineering is often discussed in terms of structure, clarity and task decomposition, the language in which the prompt is written is an equally important but sometimes overlooked factor. Modern LLMs are trained on multilingual corpora, meaning that their representations capture patterns across numerous languages. Consequently, multilingual training does not generate distinct vocabularies or probability distributions for each language. Instead, everything runs through one shared tokeniser and vocabulary, with the model computing the next-token distribution over the entire shared vocabulary at every step.

However, these corpora are not distributed uniformly across languages. Thus, the choice of prompt language can have a significant impact on the model's outputs. Models tend to demonstrate superior reasoning quality, stylistic fluency, factual accuracy and reliability in languages that dominate the training data, but may struggle with low-resource languages or dialects.

To examine this phenomenon more closely, we need to understand how multilingual training imbalances shape internal representations, how prompts interact with these learned structures, and why the behaviour of prompts differs significantly between high- and low-resource languages.

- *Uneven language representation in training data*. Most contemporary LLMs are trained on heterogeneous corpora comprising web-scale text, digitised books, code repositories, and multilingual datasets whose language distributions are markedly imbalanced. English constitutes a dominant share of the training signal, while a small set of widely used languages (e.g., Spanish, French, Chinese, and German) receive moderate coverage. The majority of the world's languages, particularly those with limited digital presence, are represented sparsely and are therefore effectively low-resource.

- *Alignment with learned representations*. From a representation-learning perspective, such an imbalance creates systematic asymmetries in the learned parameter space. LLMs rely on statistical associations learned during pre-training. High-resource languages are linked to denser, more varied and more redundant training signals. This enables the model to learn stable semantic structures, well-calibrated syntactic regularities, and richer associations with factual and world knowledge. By contrast, low-resource languages are learned under data-scarce conditions, resulting in higher variance estimates, weaker embedding geometry, and less reliable grammatical and factual generalisation. Consequently, multilingual competence in LLMs is inherently non-uniform, even when all languages are formally supported by the model.

- *Prompts are not translated into English*. Even though English is the dominant language, multilingual LLMs do not explicitly translate non-English prompts into English before processing them. Instead, text from all supported languages is mapped directly into a shared latent representation space that is learned during pretraining. While this space is language-agnostic at the architectural level, it is statistically structured by the distribution of the training data. As English usually dominates large-scale pretraining corpora, the latent space implicitly centres around English semantic, syntactic and world knowledge patterns. This representation alignment explains why performance tends to degrade gradually rather than abruptly across languages. The apparent 'English-first' behaviour is actually the result of statistical bias in representation learning rather than an underlying translation process.

- *Cross-lingual prompting*. Representation alignment is also the reason why LLMs can accept prompts in one language and generate responses in another without the need for

explicit translation. For example, if you write "*Explain X in English, then give the final answer in German*", the phrase "*in German*" becomes part of the context. The model has learned that such a linguistic cue precedes German output. Once the model generates the first German token, the hidden states shift into a region of representation space associated with German morphology, syntax, subword embeddings and German co-occurrence statistics. This creates the following self-reinforcing loop:

$$German\ token \rightarrow German\text{-}like\ state \rightarrow higher\ probability\ of\ German\ tokens.$$

Therefore, after the first German word, the model naturally continues in German unless instructed otherwise. But even when answering in German, the model still computes:

$$p(next\ token)\ over\ all\ tokens\ in\ all\ supported\ languages.$$

However, since the context contains 'in German', the probability mass shifts towards German subwords, character sequences and grammatical rules. This is why the model 'sticks' to the requested language. Note that the model does not reason about languages; it simply follows a learned mapping.

$$Instruction \rightarrow linguistic\ mode \rightarrow token\ distribution\ shift.$$

We should point out that if the prompt is ambiguous, the model may mix languages, starting in English before switching to another language. It may also produce text in one language but with syntax influenced by another language. This is particularly common in low-resource languages, where the model's internal representations are weaker.

- *Prompting in low-resource language*. When an LLM is prompted in a low-resource language, its behaviour reflects the limited and uneven training signal that the language received during pretraining. Sparse data leads to weaker lexical, syntactic and discourse representations, which are often indirectly aligned via higher-resource languages rather than being learned from numerous native examples. Consequently, generation may exhibit reduced fluency, lower stylistic and grammatical accuracy, and an increased hallucination rate, particularly in tasks requiring precise terminology or factual grounding.

5.2.6.3. A compositional structure for effective prompts

While the discussed prompt aspects emphasise specific mechanisms, effective prompts can be systematically described using a small set of compositional elements that jointly influence the model's behaviour at inference time. A generic prompt can be broken down into five distinct components:

$$Who \rightarrow What \rightarrow With\ what\ information \rightarrow How \rightarrow In\ what\ form.$$

The following prompt items correspond to these five dimensions, with each one controlling a distinct aspect of the model's inference-time behaviour:

1. Role specification → *who the model should be*

It defines the assumed identity, expertise or perspective that the model should adopt (e.g. domain expert, editor or tutor), thereby ensuring that the generation is biased towards domain-appropriate terminology, reasoning norms and discourse style.

2. Task definition → *what the model should do*

It specifies the objective to be accomplished (e.g. explaining, deriving, summarising or classifying), activates task-relevant capabilities, and clarifies which of the model's latent competencies should govern the response.

3. Contextual grounding (optional but recommended) → *what information the model should rely on*

It provides all the necessary background, assumptions, definitions and source material, ensuring that generation is placed within a clearly defined informational context.

4. Instructions and constraints → *how the model should do it*

It constrains the reasoning process, the level of detail, the methodological choices, and the permissible operations (e.g. step-by-step reasoning, the use of equations, and the avoidance of speculation), thereby shaping the structure of the generated output.

5. Output specification → *in what form the answer should appear*

It specifies the formal properties of the response, such as its length, format, structure or style (e.g. bullet points, equations only, tables or lists, or JSON schemas ([221])). This restricts the range of possible outputs, thereby improving consistency and usability.

Together, these five components provide complementary information that reduces uncertainty in the model's conditional generation process.

Example. A well-structured prompt (with annotated prompt components):

[**Role:**] *You are a mathematics lecturer specialising in topology and real analysis.* [**Task:**] *Explain why every compact metric space is separable.* [**Context:**] *Assume the reader is familiar with metric spaces, open covers, and basic point-set topology, but has not encountered this result before.* [**Instructions:**] *Provide a clear and logically structured argument, emphasizing the key ideas of the proof and avoiding unnecessary generalizations beyond metric spaces.* [**Output:**] *Present the explanation as a concise mathematical exposition in prose, using standard notation and limiting the length to approximately one page*.

This prompt explicitly defines the role of the model, setting clear expectations regarding rigour, notation and pedagogical style. The task definition is precise and narrowly scoped, clearly indicating the mathematical claim to be explained. The explicit contextual grounding calibrates the reader's assumed background knowledge, enabling the model to adjust the level of detail without providing excessive or insufficient explanation of prerequisite concepts. The instructions and constraints guide the reasoning process, emphasising logical structure and focus while limiting unnecessary digressions. Finally, the output specification restricts the form and length of the output, thereby improving consistency, reproducibility and suitability for direct inclusion in a technical or educational setting.

The importance of this structured approach becomes particularly clear when contrasted with the following underspecified prompt, which addresses the same mathematical question.

Example. A poorly structured prompt: *Why is every compact metric space separable?*

This prompt lacks several key features of effective prompt design. Firstly, it lacks role specification, providing no indication of the expected level of rigour or pedagogical approach (e.g. informal intuition versus lecturer-style exposition). Secondly, the task definition is vague: while the theorem is named, it is unclear whether a full proof, a proof sketch or a high-level explanation is required. Thirdly, there is no contextual grounding, forcing the model to guess the reader's mathematical background and increasing the risk of either excessive abstraction or unnecessary exposition. Fourthly, the prompt provides no instructions or constraints regarding structure, emphasis or scope (for example, whether the argument should be restricted to metric spaces or placed in a broader topological context). Finally, the absence of an output specification leaves the length, format and notation unconstrained, resulting in high variability across responses.

5.2.6.4. How to ask LLMs effectively (best practices)

This section provides additional prompt examples of how to interact effectively with LLMs such as ChatGPT and Microsoft Copilot.

---

[221] JSON (JavaScript Object Notation) is a simple, lightweight, text-based data format. JSON Schema, an IETF standard, provides a format for specifying the required JSON data for a given application and how to interact with it.

- **Be explicit about the task.**

  ✓ Good:

  Summarise the following article in 150 words, focusing on the economic implications. Use bullet points.

  A good prompt specifies the length, focus and format.

  × Bad:

  Summarise this.

- **Provide context.**

  ✓ Good:

  I am writing a lecture on differential vector calculus for graduate students. Explain the curl of a 3-vector field at a level appropriate for readers who are familiar with ordinary differentiation and linear algebra, but who are new to vector fields.

  This prompt clearly specifies the topic, level of detail and background knowledge required, enabling the model to tailor its response accordingly.

  × Bad:

  Explain the curl of a 3-vector field.

  This is under-specified, meaning the model is forced to guess the intended level and focus.

- **Define constraints.**

  ✓ Good:

  Generate a Python function that computes cosine similarity. Do not import any external libraries.

  Constraints prevent undesired behaviour.

  × Bad:

  Write code for cosine similarity.

- **Decompose complex tasks.**

  ✓ Good:

  First, provide a formal definition of a Hilbert space. Then explain why it is important. Finally, provide two concrete examples with explanations.

  This mirrors structured reasoning and often improves correctness.

  × Bad:

  Write everything about Hilbert spaces.

  This prompt invites verbosity without coherence.

- **Use few-shot examples when style matters.**

  ✓ Good:

  You are both a technology and marketing expert. Rewrite the following product description to match the style of the examples below. The target style is: customer-centric, benefit-driven and emotionally engaging, written in a modern, consumer-tech marketing voice.

  Examples:

  - *Meet the wireless earbuds that move with you. With all-day comfort and crystal-clear sound, they turn every commute into a private concert*.

  - *Take control of your fitness journey. Our smart scale provides real-time insights so you can celebrate progress, not just numbers*.

Description to rewrite:

This smartwatch uses advanced sensors to track heart rate, sleep cycles and daily activity.

× Bad:

Rewrite the following product description in the style of the examples below.

Example:

Our product is great.

Description to rewrite:

This smartwatch uses advanced sensors to track heart rate, sleep cycles and daily activity.

Why is it bad?

- The example is too short to convey any marketing tone, lacking voice, rhythm and persuasive structure.
- The example is generic and does not reflect real marketing language, lacking both a value proposition and emotional appeal.
- The style of the example is incompatible with that of the target text (banal vs. technical).
- The model receives no guidance regarding the audience, brand voice or desired emotional impact.
- The prompt fails to demonstrate how marketing copy should sound.

- **"Let's think about this" prompt.**

The formula for this prompt is simply the phrase "Let's think about this", followed by a topic or question.

✓ Good:

Prompt: *You are a legal scholar specialising in contract law. Analyse whether the following scenario constitutes a valid contract under the general principles of common law.* ***Let's think about this*** *step by step, explicitly examining offer, acceptance, consideration and possible defences in detail, but avoid jurisdiction-specific doctrines unless strictly necessary. Present your analysis as a structured sequence of legal elements, followed by a brief conclusion.*

Scenario: *Company A emails Company B, offering to sell 1,000 units of a component at a fixed price. Company B replies, "Agreed, subject to final approval by our board," but later refuses to proceed after the board declines approval.*

Here, the reasoning cue is used for a task that inherently requires an element-by-element legal analysis. The prompt clearly defines the role, specifies the doctrinal framework (offer, acceptance and consideration) and constrains the scope to general common-law principles. It also limits the form of the output. Therefore, the phrase 'Let's think about this step by step' structures the legal reasoning process rather than merely encouraging verbosity.

× Bad:

Prompt: *Is this a contract? Let's think about this.*

Scenario: *Company A emails Company B, offering to sell 1,000 units of a component at a fixed price. Company B replies, "Agreed, subject to final approval by our board," but later refuses to proceed after the board declines approval.*

Why this fails? In this case, the reasoning cue is not supported by structure. The prompt lacks role specification and provides no legal framework for analysis or

constraints on the depth or form of the reasoning. Consequently, the model may produce analysis that is unfocused or inconsistent in terms of jurisdiction, mixing intuition, doctrine, and speculation. The cue "Let's think about this" increases output length without reliably improving legal correctness or clarity.

- **Select high-resource prompt language.**

  ✓ Good:

  Summarize the following paragraph in clear English, using concise, formal language: ….

  The prompt language matches a high-resource language the model is strongest in, and the task instructions are explicit.

  × Bad (low-resource language):

  Resuma este párrafo (Spanish prompt for a model predominantly trained on English).

  The model may have limited training data in Spanish, leading to lower fluency, increased hallucinations, or incomplete responses.

  × Bad (mixed language):

  Summarize the following paragraph in English.

  Example 1: Input: "Ceci est un exemple." Output: "This is an example."

  Example 2: Input: "Dies ist ein Test." Output: "This is a test."

  Mixing languages in few-shot examples can confuse the model, as the statistical patterns for reasoning and translation differ across languages, potentially reducing coherence and correctness of the output.

Practical recommendations:

- For complex reasoning, mathematics, scientific explanation, or program synthesis, prefer English prompts.
- For low-resource languages, consider a two-step approach:
  - Prompt in English (or another high-resource language).
  - Request translation or adaptation into the target language.

Example. The prompt can be written in English, specifying:

Translate the following summary into German: …

or

Answer the following question in German: …

The model leverages its cross-lingual representations learned from multilingual corpora to produce output in the target language. Even if English dominates the training data, the model has typically seen enough German examples to generate fluent text. When in doubt, test both languages and compare the quality of the output.

5.2.6.5. How NOT to ask LLMs (common pitfalls)

While prompt engineering provides fine-grained control over LLMs, many failures in practice are not due to limitations in the models themselves, but rather to poorly designed prompts. This section highlights common pitfalls in prompt design that can systematically degrade performance. It illustrates how seemingly harmless choices, such as vagueness, implicit assumptions or open-ended reasoning prompts, can undermine accuracy and alignment. Therefore, understanding how not to ask is a necessary complement to best practices in effective prompt engineering.

- **Overly vague prompts**. Vague prompts shift the interpretation task onto the model, which increases the risk of irrelevant or superficial answers.

Avoid:

Tell me something interesting about AI.

- **Conflicting or implicit constraints**. Conflicting constraints lead to degraded outputs because the model cannot satisfy all conditions at once.

Avoid:

Give a very short but extremely detailed explanation with all equations.

- **Assuming hidden context**. LLMs do not reliably infer unstated assumptions.

Avoid:

As before, do the same thing but better.

Unless the relevant context is explicitly included, performance will degrade.

- **Treating the model as a deterministic oracle**. LLMs produce probabilistic outputs. Repeated queries with weak prompts may produce inconsistent results.

Avoid re-asking the same underspecified question and expecting identical results.

- **Problematic use of a reasoning cue**. Use “Let's think about this” only when the task requires structured, multi-step reasoning, and when the legal framework and scope of analysis have been explicitly specified. For tasks with low intrinsic reasoning depth, using this prompt unnecessarily expands the output space, increasing generation length and variance without providing additional constraints on correctness.

Avoid:

State the definition of a continuous function on a metric space. Let’s think about this step by step.

The task is purely definitional and does not involve compositional inference or logical case analysis. Introducing an explicit reasoning cue leads to unnecessary expansion rather than sharpening the constraints on the generated response.

- **Assuming the model knows your intent**. LLMs do not have access to a user's unstated goals, prior reasoning or external context beyond the prompt itself. If intent, scope or evaluation criteria are not explicitly stated, the model has to infer them from weak or ambiguous signals, which increases uncertainty in the conditional generation process.

Avoid:

Is this medical treatment appropriate?

This prompt assumes a shared understanding of critical contextual factors that remain implicit. These include the patient's health condition, the level of diagnostic certainty, and how the answer will be used (for an educational overview or medical treatment). Without explicit guidance, the model may generate a response that is overly generic, lacks caution, or is inappropriate in terms of the level of medical rigour, despite appearing fluent and authoritative.

- **Asking for impossible or unsafe tasks**. LLMs are probabilistic with no direct access to the physical world, private data or real-time verification. They are also subject to safety constraints that prohibit harmful or unethical behaviour. Prompts that implicitly or explicitly request actions beyond these limits, such as guaranteeing factual accuracy or making medical or legal decisions without context, create a fundamental mismatch between user expectations and model capabilities. In such cases, failure modes range from refusal and partial compliance to outputs that are superficially plausible yet unreliable.

Avoid:

Tell me which stock will outperform the market next month and guarantee that this strategy will not lose money.

This prompt requests an impossible guarantee regarding the inherently stochastic behaviour of markets and treats the model as a predictive oracle rather than an analytical tool. Financial

markets are influenced by uncertain factors that cannot be forecast with certainty, so any claim of guaranteed performance would be misleading. Such prompts, therefore, force a mismatch between user expectations and model capabilities, often resulting in overconfident yet unreliable outputs.

5.2.6.6. Differences in effective prompting: ChatGPT vs. Microsoft Copilot

Although ChatGPT and Microsoft Copilot are both built on Transformer-based LLMs, they operate in significantly different environments. These differences influence how users should formulate prompts to obtain high-quality outputs. ChatGPT is primarily a general-purpose conversational model, whereas Copilot is an integrated productivity assistant embedded across the Microsoft ecosystem, including Windows, Office, Edge, and other applications. Consequently, effective prompting strategies differ in terms of context requirements, task formulation, and expected output structure.

- **Interaction context and information flow**.
- *ChatGPT: Self-contained dialogue*. ChatGPT operates in a purely conversational setting. It has no inherent access to external documents, applications or user context. Therefore, all relevant information must be provided within the prompt. Effective prompts tend to be self-contained, richly specified and often multi-step.

  Implication: Users must articulate the task, audience, constraints and examples directly in the prompt.
- *Copilot: Context-Aware Assistance.* Copilot interacts with users inside applications such as Word, Outlook, PowerPoint and Edge. It can interpret words such as 'this document', 'this slide', or 'the selected text' because it can use the surrounding environment to provide contextual grounding.

  Implication: Prompts can be shorter and more action-oriented, relying on the application context to supply missing details.
- **Role specification**.
- *ChatGPT*. Explicit role specification (*'You are a mathematician ... '*) can improve the quality of the output by activating domain-specific discourse patterns that were learned during the tuning of the instructions.

  Implication: Role specification is a primary control level.
- *Copilot*. The effective role is frequently implicitly determined by the host application (e.g. an editor in Word, an analyst in Excel or an assistant in Outlook). Explicit role instructions may be redundant or ignored if they conflict with the intent of the application.

  Implication: The role specification is a secondary or optional control level.
- **Task orientation and output expectations**.
- *ChatGPT: Open-ended reasoning and exploration*. ChatGPT performs well in conceptual explanation, creative generation, and analytical reasoning. It responds best to prompts that specify the structure, the reasoning steps, and the output format.

  Effective pattern: *Explain concept X to audience Y, following steps 1–3. Use format Z*.
- *Copilot: Productivity-focused execution*. Optimised for tasks such as document summarisation, email drafting, text rewriting, slide content generation and key point extraction. It requires concise, directive prompts that clearly correspond to productivity actions.

  Effective pattern: *Summarise this document for a technical audience*.

- **Prompt length and explicitness**.
- *ChatGPT*. It benefits from long, explicit, highly structured prompts. Few-shot examples can significantly improve stylistic control. Chain-of-thought or stepwise instructions can enhance the quality of reasoning.
- *Copilot*. It benefits from shorter, more directive prompts. Few-shot examples are helpful, but not usually necessary. Chain-of-thought prompting is usually unnecessary unless the task is analytical.
- **Prompt language**.
- *ChatGPT: Broad multilingual robustness, but uneven across languages*. Although ChatGPT-style models are trained on very large multilingual corpora, the distribution of languages is highly skewed. English dominates, followed by a few high-resource languages such as German, French, Spanish and Chinese. Many languages are represented only sparsely. Consequently, ChatGPT generally exhibits robust reasoning capabilities in English and other high-resource languages, whereas performance in low-resource languages may deteriorate, resulting in shorter reasoning chains, increased hallucinations, and diminished stylistic fidelity.

  Implication: For reasoning-heavy or technical tasks, prompts should ideally be written in English or another language with a large amount of resources available. The final output can then be requested in the target language. Although ChatGPT is more tolerant of multilingual prompting than Copilot, the quality gap between languages remains noticeable.
- *Copilot: English-centred optimisation and weaker multilingual robustness*. Copilot is also multilingual, but its optimisation is heavily influenced by its intended use cases, such as productivity workflows, coding, document editing and reasoning within Microsoft applications. The majority of these domains rely on English-language corpora (documentation, APIs, technical writing and business communication). Consequently, Copilot's reasoning performance is most reliable in English. High-resource European languages perform well for everyday tasks, but deep reasoning, formal analysis and code-related tasks deteriorate more quickly than in ChatGPT when the prompt is written in a low-resource language.

  Implication: The choice of prompt language has a stronger effect on Copilot's output quality. English should therefore be used for analytical, technical or multi-step reasoning tasks. Prompts in other languages work well for summarisation, rewriting or stylistic tasks, but not for complex reasoning.
- **Retrieval-augmented inference and knowledge access**.
- *ChatGPT: Conditional retrieval capability*. In its base configuration, ChatGPT relies on parametric knowledge, which is subject to the training-time knowledge cutoff. However, when operating in Browsing or Deep Research mode (e.g. GPT-5.2), it can dynamically retrieve and synthesise up-to-date information from external web sources during inference. In such cases, retrieval must typically be explicitly invoked or prompted, and the model may present cited sources.
- *Implication:* Prompts should specify whether up-to-date or source-backed information is required. Users may also need to instruct the system to consult external sources or verify claims.

Example. To trigger a deep research answer, simply state this explicitly in the prompt:

Provide a Deep Research answer with
- *multiple independent sources*
- *explicit citations*
- *a clear statement of publication dates*
- *a short critical evaluation of the reliability of the sources*.

- *Copilot: Integrated retrieval by design*. It incorporates real-time information retrieval through Microsoft's search infrastructure and application-layer context (e.g. enterprise documents, emails and calendar data) and often operates without requiring explicit user instructions. Retrieval is tightly coupled with the host environment and may operate implicitly in the background.

  Implication: Prompts can access current information and contextual data within the Microsoft ecosystem. This shifts the user's focus from triggering retrieval to constraining the scope and relevance of the retrieved material.

- **Style and tone control**.
- *ChatGPT*. It is highly responsive to detailed stylistic constraints. It performs well when given explicit tone descriptors or stylistic exemplars.
- *Copilot*. It defaults to a professional, concise, business-oriented tone. It requires explicit instruction to adopt alternative styles (e.g. humorous, casual or narrative).

The table below summarises the conceptual and practical differences across several dimensions.

Dimension	ChatGPT	Microsoft Copilot
Primary mode	Conversational reasoning	Productivity assistance
Context	User-provided	Application-integrated
Prompt length	Longer, explicit	Shorter, action-oriented
Best for	Explanation, analysis, creativity	Summaries, rewrites, document tasks
Few-shot examples	Highly effective	Helpful but not essential
Tone control	Fully flexible	Defaults to professional tone
Interaction style	Dialogue-driven	Task-driven
Reference scope	Only what user provides	Active document or app context

In a nutshell, effective prompting in ChatGPT emphasises the explicit specification of role, reasoning and output. In contrast, effective prompting in Microsoft Copilot emphasises the concise articulation of goals that leverage the context and constraints provided by the application.

Although prompt engineering offers effective methods for directing today's LLMs and generating high-quality behaviour from existing architectures, it also reveals the limitations of what current models can reliably accomplish. These practical constraints naturally motivate broader enquiries into the structural, architectural and theoretical challenges that will define the next phase of LLM research. The following subchapter explores these future directions and considers the open questions that will influence the development of the field.

5.2.7. Outlook: future directions and open challenges in LLM research

The preceding sections have outlined the conceptual foundations, architectural principles, learning mechanisms and inference-time control strategies that currently define large language models. Together, these developments have established LLMs as a central paradigm in contemporary natural language processing. However, they also expose a set of structural tensions that limit the scope of current approaches and will drive future research. The most significant of these are the reliance on next-token prediction as a unifying training objective, the dominance of the Transformer architecture as the default inductive bias and the increasing reliance on prompt-based conditioning as the primary mechanism for behavioural control during inference.

This section examines a selection of representative research frontiers that address these tensions. Rather than providing an exhaustive overview, it emphasises several significant trends currently shaping the discourse in LLM research, such as efforts to move beyond pure next-token prediction, innovations aimed at expanding context and enhancing efficiency, data-centric training methods, and emerging architectural alternatives to the Transformer. The chapter concludes by discussing deeper theoretical questions that extend beyond current paradigms and may define progress over the next decade.

The purpose of this section is twofold. Firstly, it identifies active research areas that aim to expand or reconsider existing paradigms in model architecture, training and data utilisation. Secondly, it identifies a small set of significant questions that lie beyond current methodological frameworks yet are likely to influence the long-term evolution of LLM-based NLP.

5.2.7.1. Main areas of research (representative selection)

Current research on large language models is characterised by a shift in focus from demonstrating empirical capabilities to addressing the structural limitations that emerge as the size and scope of the models increase. Rather than constituting a single, unified agenda, this work spans several interrelated research areas, each of which is motivated by a specific shortcoming of the current approach. The following discussion presents a selection of these areas, focusing on their motivations, key technical concepts, and initial findings.

- **Beyond next-token prediction: toward reasoning and world models**. Despite their impressive performance across a wide range of tasks, contemporary LLMs are essentially trained to minimise next-token prediction loss. While this approach produces rich linguistic representations, it offers no explicit motivation to learn structured reasoning processes or coherent internal models of the world. Consequently, LLMs often demonstrate fragile reasoning, sensitivity to prompt phrasing and limited robustness in tasks requiring multi-step inference or hypothetical reasoning.

Research in this area explores training objectives and architectures that extend beyond the maximisation of pure autoregressive likelihood. Suggested approaches include

- *Latent reasoning*. At its core, latent reasoning is the concept that an LLM can perform implicit internal reasoning steps that are not evident in its output tokens. These steps occur within the model's hidden (latent) representations rather than within the text produced. This process does not rely on explicit natural language chains-of-thought or explicit supervision at the level of individual tokens.

  Imagine the model is simulating chains-of-thought internally, compressing them into dense vectors, and outputting only the final answer. Therefore, the model may 'think' more than it 'says'. Just as humans do not always rely on language for their cognitive processes, LLMs allocate most of their processing power to the latent space. Even when they provide a brief response, their hidden layers encode intermediate logical or arithmetic states. Consequently, explicit chain-of-thought is not the only mode of reasoning. Models can reason silently and sometimes perform better when not forced to verbalise steps. This latent chain-of-thought (Latent CoT) involves reasoning in continuous internal representations, often via recurrent mechanisms within the model. Thus, Latent CoT offers richer expressivity and access to non-linguistic reasoning paths, potentially unlocking new frontiers in model reasoning.

  There are models explicitly designed for Latent CoT reasoning, in which intermediate inference states are represented without being expressed as natural language text. These approaches introduce latent variables or hidden scratchpads that influence the generation process while remaining unobserved during the inference stage. This aims to decouple the internal reasoning process from the surface-level explanations. Examples include variational latent language models, latent transformer architectures and models trained with hidden reasoning traces or auxiliary planning objectives. Empirical results suggest improved robustness and reasoning performance on compositional tasks. However,

challenges remain in stabilising training, preventing posterior collapse and interpreting the learned latent representations (see [C16], [L19], [W13], and [Z11] for more details).

- *Predictive world modelling*. Another prominent area of research that builds on next-token prediction is incorporating predictive world modelling into language models. Rather than treating text as merely a sequence of symbols, this approach interprets linguistic observations as partial projections of an underlying latent world state which evolves according to structured dynamics. Formally, these approaches posit latent states s_t and transition operators $s_{t+1} = f(s_t, a_t)$, with language tokens being generated as observations conditioned on these states ([222]). The objective is to learn not only surface-level statistical regularities, but also implicit models of causality and counterfactual structure ([223]). Unlike standard autoregressive training, predictive world models aim to support planning, multi-step reasoning and robustness under distribution shift by linking prediction to latent state dynamics.

  While early empirical results suggest improvements in tasks requiring temporal consistency or causal inference, significant challenges remain in defining appropriate state representations, learning stable transitions from purely linguistic data and integrating such models with large-scale pre-training regimes. Therefore, predictive world modelling is a promising but still exploratory departure from traditional language modelling objectives (see [W17] and [W18] for more details).

- **Efficient and long-context architectures**. As model sizes and context windows increase, the quadratic computational cost of attention has become a central bottleneck. Furthermore, even with large context windows, models often struggle to maintain coherence over long documents or retrieve relevant information efficiently. These challenges have driven research into architectures and mechanisms that support longer contexts, more efficient memory usage, and scalable inference. This research frontier explores architectural modifications that either reduce attention complexity or replace it altogether. The goal is not only to process longer sequences, but also to maintain stable representations over extended contexts. Technical ideas under exploration ([D16], [H18]):

  - *Sparse and linear-time attention mechanisms*. The core idea is to restrict or approximate the attention operation so that each token only considers a structured subset of other tokens, or to reformulate attention so that it can be computed in linear time. Sparse attention schemes impose inductive biases, such as locality, global summary tokens or predefined attention patterns, to preserve functionality while dramatically reducing computation cost. Examples of this include layer-sparse, sliding-window and hierarchical designs. By contrast, linear-time approaches reinterpret attention through kernelisation, low-rank factorisation, or state-space-inspired recurrences. This enables the computation of context representations in $O(n)$ time (i.e. execution time grows proportionally to the input size n) without forming the full attention matrix explicitly.

    These methods have enabled models to process long contexts ranging from tens of thousands to millions of tokens, shifting the focus from a bottleneck to a controllable design choice. However, they also reveal key trade-offs between efficiency, stability and representational fidelity, raising questions about the most effective forms of sparsity or approximation.

  - *Memory-augmented architectures*. Memory-augmented architectures extend standard Transformer models by incorporating explicit mechanisms for storing, retrieving and

---

[222] a_t denotes an action or intervention variable that influences the evolution of the latent world state. Its precise interpretation depends on the modelling framework, but the core idea is that state transitions are not purely autonomous.

[223] Counterfactual reasoning is the cognitive process of imagining alternative scenarios to reality. It involves asking "what if" or "if only" questions about events that did not happen in order to understand causes, learn from mistakes and make better decisions in the future.

updating information beyond the fixed-length context window. This addresses limitations in long-term dependency modelling and context persistence. Rather than relying solely on self-attention over the current token sequence, these models introduce an external or semi-persistent memory, such as a key–value store, recurrent state vector, or database, that can be accessed through learned read and write operations. This design enables the model to accumulate information across long documents, multi-turn interactions or training and inference episodes while keeping per-step computation limited.

From a modelling perspective, these architectures shift some of the representational workload from attention to memory management. This raises new challenges regarding the stability of stored representations and the selection of memory. Nevertheless, memory-augmented models are a significant step towards scalable, long-term reasoning and the integration of more persistent forms of knowledge in large language models.

- **Data-centric and curriculum-based learning**. The prevailing emphasis on scaling models has increasingly exposed data as a bottleneck. Large-scale web corpora tend to be noisy, redundant and unevenly distributed across domains and languages. Furthermore, training uniformly over vast datasets can mask rare yet semantically significant patterns and hinder systematic generalisation.

Data-centric research reframes performance improvements as a function of data quality, composition and ordering, rather than model size alone. This includes filtering and weighting datasets, generating synthetic data, employing curriculum learning strategies and using adaptive sampling schemes ([224]) to align training data with desired capabilities. Some approaches draw inspiration from human learning, emphasising staged exposure and task progression ([B35], [X2]).

Empirical studies indicate that carefully curated or dynamically sampled datasets can significantly improve efficiency and downstream performance, sometimes matching the gains achieved through parameter scaling. Nevertheless, principled methods for curriculum design and data valuation remain underdeveloped, and results often depend strongly on the specific task and domain.

In summary, the research areas discussed in this section demonstrate the range of ongoing efforts to overcome the structural limitations of contemporary LLMs. They reflect a shift from brute-force scaling towards more principled approaches that incorporate new objectives, architectures and data strategies. The next section will examine architectural innovations beyond the Transformer in greater detail, focusing on emerging alternatives and the debates surrounding them.

5.2.7.2. Architectural innovations beyond the Transformer

The Transformer architecture has dominated large language models for almost a decade, enabling significant progress in natural language understanding, generation, and multimodal integration. Its scalability, parallelism and expressive power have established it as the de facto foundation for contemporary LLMs. However, as models continue to increase in size and context length, the limitations of the Transformer architecture have become more apparent. These include the quadratic computational cost of attention, difficulties in modelling very long sequences and the lack of explicit mechanisms for persistent latent states or structured memory. Consequently, researchers are actively exploring architectural alternatives or hybrids that could complement – or potentially replace – the Transformer paradigm.

[224] Curriculum learning involves training strategies where examples are presented in a structured order, usually progressing from simpler to more complex instances. The aim is to improve optimisation stability and generalisation by shaping the learning trajectory. Adaptive sampling schemes build on this concept by dynamically adjusting the data distribution during training, often based on model performance, uncertainty or loss, so that the model focuses primarily on informative, underrepresented or challenging examples as it learns.

A prominent line of research revisits classical sequence modelling ideas such as recurrence, continuous-time dynamics and structured state updates through the lens of modern deep learning. State-space models (SSMs) and recurrent-hybrid architectures seek to overcome several structural limitations of the Transformer.

- **State-Space Models (SSMs)**. These models reformulate sequence modelling as the evolution of a latent state, which is governed by a transition operator (which is usually continuous-time) of the following form

$$H_{t+1} = AH_t + BX_t,\ Y_t = CH_t,$$

where

- A is the state transition matrix, which governs how the latent state H_t evolves over time in the absence of new input.
- B is the input projection matrix. It maps the current input X_t (e.g. a token embedding) into the state space, controlling how new information is incorporated into the latent state.
- C is the output projection matrix**.** It maps the latent state H_t to the output Y_t, which may represent logits, intermediate features, or values passed to subsequent layers.

Recent neural SSMs parameterise these operators to enable efficient learning while retaining linear-time complexity in sequence length. Unlike attention-based Transformers, which recompute pairwise interactions across tokens, SSMs maintain a compact latent state that implicitly encodes long-range dependencies. This makes them particularly well suited to very long contexts and streaming settings. Modern formulations introduce structured parameterisations and parallel scan algorithms that enable SSMs to be efficiently trained on GPUs, thereby closing the performance gap with attention-based models on language tasks ([G29]).

- **Recurrent-hybrid architectures**. Recurrent-hybrid architectures combine elements of classical recurrence (see Section 4.3.6) with Transformer-style representations in order to capture the strengths of both paradigms. Unlike pure SSMs, these models usually retain attention mechanisms, often in a restricted or local form, while introducing recurrent state updates that persist across sequence segments or layers. These hybrids can maintain long-term information via recurrence while using attention for flexible, content-based routing within shorter contexts. In terms of architecture, this includes models with segment-level recurrence, gated state updates or interleaved recurrent and attention layers. From a modelling perspective, recurrent hybrids introduce an explicit inductive bias towards temporal continuation and iterative computation, which naturally aligns with tasks involving stepwise reasoning or sequential decision-making processes. Examples include Transformer-XL-style recurrence, state-augmented Transformers, and recent architectures that integrate recurrent memory with attention to achieve linear or near-linear scaling without abandoning attention entirely ([T19]).

5.2.7.3. Case study: the BDH architecture

One example of a proposed next-generation architecture is the Dragon Hatchling (BDH) model, which was introduced in 2025 by A. Kosowski et al. of Pathway – a research-driven AI company based in Palo Alto. This novel architecture was presented in the paper *The Dragon Hatchling: The Missing Link between the Transformer and Models of the Brain* ([K23]). The BDH model is an interesting example of biologically inspired hybrid architecture. It combines structured state updates with selective attention in an attempt to unite the efficiency and long-range capabilities of state-space models with the expressive power of attention mechanisms. Conceptually, the BDH model seeks to bridge Transformer-based architectures with high-level principles from computational neuroscience, such as sparse activation patterns, hierarchical processing and context-dependent routing of information.

A direct comparison with the original Transformer reveals the architectural rationale behind BDH.

- **Computational paradigm**. The Transformer relies on full self-attention with quadratic complexity, whereas BDH introduces a persistent latent state and selective attention to achieve linear-time or near-linear-time processing.

- **Memory and information flow**. Standard Transformers recompute token interactions independently at each layer, rather than maintaining an explicit recurrent latent state across layers or time. Although autoregressive inference uses a key-value (*KV*) cache to avoid re-computing past projections, this cache only stores previously computed activations and does not constitute a dynamically evolving memory. By contrast, BDH maintains a structured latent state that is updated across layers, thereby introducing a form of working memory that goes beyond static contextual retrieval.

- **Attention mechanism**. Transformers use dense multi-head attention ([225]), whereas BDH employs sparse, state-guided attention applied only where needed.

- **Layer structure**. Transformer layers are uniform (attention + feed-forward), whereas BDH layers integrate state updates, selective attention, and biologically inspired gating – a distinctive BDH feature. This gating comprises a family of sparse, selective and state-dependent activation mechanisms that are loosely motivated by cortical computation. While these mechanisms do not constitute literal neurobiological models, they are computational abstractions of three well-established principles: sparse activation, context-dependent routing, and competitive inhibition. BDH incorporates these principles in several ways:

  - *Sparse, state-dependent activation*. For a given token, only a subset of units ([226]) or pathways is activated, depending on the current latent state. This mirrors cortical circuits, in which only a fraction of neurons fire at any given time.
  - *Gated state updates*. The persistent latent state is updated through gating functions that determine how much prior information to retain and how much new information to incorporate. This resembles the gating dynamics of LSTMs (see Section 4.3.6.4) and the inhibitory control found in cortical networks.
  - *Selective, state-guided attention*. Rather than dense self-attention, BDH focuses attention on a small subset of tokens identified as relevant by the latent state. This is analogous to top-down attentional modulation in the brain.
  - *Dynamic routing through conditional pathways*. Gating mechanisms route information through different submodules depending on the context. This is reminiscent of mixture-of-experts models (see Section 5.2.5.4) and cortical column specialisation.
  - *Inhibitory-style normalisation*. Competitive gating suppresses weaker activations, encouraging sparsity and stability, similar to lateral inhibition in sensory systems.

  These mechanisms aim to reduce computation, encourage specialisation, and provide more interpretable internal dynamics than the uniform, dense computation of Transformer layers.

- **Inductive biases**. An inductive bias is a structural assumption encoded in the architecture that constrains the class of functions that the model can learn efficiently. The Transformer exhibits comparatively minimal inductive bias, as its dense self-attention mechanism imposes no inherent locality, recurrence or hierarchy beyond positional encoding. By contrast, BDH introduces architectural constraints favouring sequential processing, hierarchical organisation and sparsity, thereby embedding stronger prior assumptions about the structure of the data.

---

[225] Vaswani's attention mechanism is termed 'dense' because each token computes attention weights for every other token. This yields a fully populated attention matrix and results in quadratic computational complexity with respect to the sequence length.

[226] In this context, 'units' refers to the individual computational elements within a BDH layer, such as neurons, channels or gated sub-components, that can be selectively activated or suppressed.

- **Efficiency**. BDH is designed to mitigate the Transformer's quadratic attention cost, particularly for long-context tasks.

The BDH model also highlights the difficulties involved in suggesting alternatives to the Transformer. Several points of criticism have emerged:

× **Preliminary empirical status**. Current evaluations primarily compare BDH to relatively small Transformer baselines (e.g., GPT-2–scale models). There is presently no published evidence demonstrating that BDH matches or surpasses modern large-scale Transformer architectures in absolute performance, scaling efficiency, or robustness across diverse benchmarks. As a result, claims of architectural superiority remain provisional.

× **Speculative biological analogy**. The 'brain-inspired' framing is conceptually appealing but largely metaphorical. While notions such as a persistent latent state or hierarchical processing evoke cognitive analogies, there is limited empirical evidence connecting BDH's mechanisms to validated neuroscientific models. Critics argue that such analogies risk overstating explanatory power without mechanistic correspondence.

× **Uncertain scalability**. BDH has not yet been systematically evaluated at frontier model sizes, with extreme parameter counts, or with very long context lengths. Open questions remain about optimisation stability, gradient propagation through evolving latent states, and whether structured memory introduces bottlenecks that limit scaling laws compared with dense attention models.

× **Potential expressivity constraints**. By introducing structured latent evolution, sparsity and stronger inductive biases towards sequential processing and hierarchy, BDH may reduce representational flexibility relative to dense self-attention. While such constraints could in principle improve efficiency or generalisation, they could also limit the model's ability to capture arbitrary global interactions.

× **Optimisation and hardware efficiency concerns**. Sequential state updates and structured routing mechanisms may reduce parallelism compared to standard Transformer layers, which could diminish practical throughput on modern accelerators. Additionally, sparse or memory-augmented components can introduce challenges regarding load balancing and training stability.

× **Lack of consensus**. The research community remains divided. Some researchers view BDH as a promising exploration of structured world modelling and memory-centric architectures, particularly for data-efficient or interpretable learning. Others contend that competing approaches, such as improved attention mechanisms, retrieval-augmented systems or scaling-based strategies, offer clearer empirical advantages at present.

Together, these criticisms do not refute BDH, but rather position it as an early-stage architectural proposal whose theoretical motivations are intriguing, but whose large-scale empirical validation is still to be demonstrated.

In conclusion, BDH is one of several competing approaches designed to address the limitations of the Transformer. These approaches differ in terms of their inductive biases, computational trade-offs and empirical performance. Some aim to preserve the Transformer's strengths while improving efficiency, while others seek to rethink sequence modelling from first principles. What unites them is the recognition that the next generation of LLMs may require architectural innovations that go beyond scaling the Transformer alone.

Although no clear successor has yet emerged, ongoing developments in state-space models, recurrent hybrids and other alternatives signal a field in active and ongoing evolution. The next section turns to deeper theoretical questions that extend beyond current paradigms and that could influence the long-term trajectory of LLM development.

5.2.7.4. Theoretical questions beyond current paradigms

The rapid progress of LLMs has primarily been driven by empirical advances, such as larger datasets, increased computing power, improved optimisation and architectural refinements. Yet

beneath these engineering achievements lie unresolved theoretical questions about what these models represent, how they generalise, and what kinds of cognitive or computational structures they implicitly learn. As LLMs approach the limits of current paradigms, these foundational issues are becoming increasingly important. This section outlines some of the most significant theoretical questions that extend beyond the Transformer and its emerging alternatives, influencing the long-term direction of research.

- **Formal guarantees for reasoning and generalization**. Current LLMs demonstrate impressive empirical performance in reasoning tasks. However, this ability is largely behavioural rather than formal: the models succeed in practice without any guarantees regarding correctness, consistency or generalisation beyond the observed distributions. Unlike classical algorithms, whose correctness can be proven with respect to a specification, LLM reasoning emerges from statistical pattern learning and is sensitive to prompt formulation, input phrasing and distributional shifts ([A19]).

The discrepancy between observed behaviour and theoretical understanding raises several fundamental questions. The challenge lies in establishing formal guarantees or principled impossibility results for reasoning and generalisation in neural sequence models. These include whether certain classes of compositional or symbolic reasoning can be learned with restricted sample complexity, the conditions under which reasoning extrapolates beyond training regimes, and the relationship between internal representations and algorithmic computation. Addressing these issues would require the integration of tools from learning theory, formal verification, and computational complexity with modern deep learning. This could fundamentally change our understanding of what it means for an LLM to 'reason correctly'.

- **Learning grounded world models from language alone**. Another unresolved question is the extent to which world models – internal representations that support prediction, planning and counterfactual reasoning – can be learned purely from text. Although LLMs encode vast amounts of factual and procedural knowledge, it is unclear whether linguistic co-occurrence statistics alone are sufficient to induce robust, causally grounded models of the physical or social world ([B33]).

This issue raises deep conceptual questions: language only indirectly reflects the world, as it is filtered through human conventions, omissions and biases. Consequently, models trained solely on text may lack the causal structure necessary for reliable prediction in the event of a distributional shift. The open question is whether increasingly large and diverse corpora can compensate for this limitation or whether grounding through interaction, perception or experience is fundamentally required. Resolving this tension would have profound implications for the role of multimodality, simulation and environmental interaction in future LLM research.

- **Theoretical limits of prompt-based control**. Prompt engineering shows that significant behavioural control can be achieved over fixed, pre-trained models. However, this control is indirect, probabilistic and fragile by nature, as it relies on conditioning rather than parameter updates ([W13]). One open theoretical question is how to characterise the limits of such prompt-based control: which behaviours can be reliably induced by prompts alone and which require architectural changes or fine-tuning?

From a formal perspective, prompts act as soft constraints on the model's conditional distribution. However, they do not enforce invariants or guarantee compliance ([227]). This raises questions about *expressivity* (i.e. what behaviours can be represented via prompting), *stability* (i.e. how sensitive these behaviours are to perturbations) and *identifiability* (i.e. whether

[227] Invariants are properties that must be true for all valid system outputs or executions. Although prompts influence the distribution of an LLM's output, they do not impose hard constraints on generation. Consequently, they bias the model towards satisfying a requirement without guaranteeing compliance across all possible outputs. Consequently, prompt-based control can only probabilistically influence invariants, rather than enforce them in the formal sense. The absence of a compliance guarantee means there are no enforcement or verification mechanisms: prompts influence behaviour, but cannot ensure that the specified requirements are always satisfied.

desired behaviours can be uniquely specified by natural language instructions). Understanding these limits is essential for evaluating the long-term feasibility of prompt engineering as a control paradigm, as well as for developing more principled mechanisms for alignment, interpretability, and safety.

In summary, the pursuit of formal guarantees for reasoning and generalisation is one of the most significant theoretical challenges in contemporary AI research. This issue touches on the nature of reasoning, the structure of learned representations, the limits of statistical learning and the possibility of verifiable computation in large neural systems. While current LLMs produce impressive empirical results, they lack the formal assurances that are fundamental to traditional reasoning systems. Developing such assurances, whether through new architectures, hybrid models or entirely new theoretical frameworks, remains an open frontier with far-reaching implications for the future of AI.

6. What AI might become in the next 5–10 years?

Prediction is very difficult, especially if it's about the future.

– Niels Bohr ([228])

What thinks for us today will think without us tomorrow.

– Microsoft Copilot

As artificial intelligence continues to evolve at an unprecedented pace, the question is no longer whether it will shape our future, but rather how profoundly and in which directions. Although predicting the future of AI is inherently uncertain, a growing body of expert commentary, forecasting studies and public statements from leading AI researchers provides a coherent picture of several likely trajectories. While there is disagreement on timelines and risks, there is a broad consensus that the next decade will see transformative advances in capability, integration and societal impact.

This chapter summarises these expert perspectives, drawing on recent forecasts and public statements.

6.1. Rapid integration and pervasive AI

There is broad consensus among researchers and industry leaders that AI will continue to spread across a wide range of economic and social domains. In the short term, AI systems, especially large foundation models, are expected to become more deeply embedded in business operations, scientific discovery, healthcare, and education. For instance, investment in enterprise AI is expected to shift from improving operational efficiency to driving innovation and transformation, with AI influencing business strategy and decision-making over the coming years ([A20]).

Experts also anticipate that AI will become deeply embedded in our everyday tools, infrastructure and personal environments. Forecasts suggest that, by 2030:

- AI assistants will autonomously manage schedules, finances and personal communication.
- Wearable devices will provide continuous health monitoring and detect diseases early.
- Homes, vehicles and domestic appliances will adapt dynamically to user preferences.

These predictions are reflected in forward-looking analyses of the adoption of AI in both the consumer and industrial sectors. It is not just increased automation that is expected, but ambient AI – systems that operate continuously and unobtrusively in the background ([R22]).

6.2. Multimodal and agentic AI systems

Another common expectation is that multimodal AI systems, which can integrate text, images, audio, video and sensor data into unified reasoning processes, will rise in popularity.

Current models already demonstrate early versions of this capability, and experts predict that multimodal integration will become standard practice by the early 2030s. In parallel, AI systems are expected to become more agentic – capable of planning, taking actions, and interacting with digital environments. Forecasting studies highlight the likelihood of AI systems that can autonomously:

- write, test and deploy software
- conduct scientific research

[228] Niels Henrik David Bohr (1885 – 1962) was a Danish physicist and one of the most influential scientists of the 20th century.

- manage complex workflows
- operate as semi-autonomous digital employees.

Large-scale surveys of AI researchers suggest that many anticipate these milestones being achieved by 2028–2033 ([G30]).

6.3. Automation, labour markets, and economic transformation

Experts recognise that AI will have a significant impact on labour markets and economic structures. Recent analyses project structural shifts in employment up to 2030, with millions of roles being affected as tasks become augmented or automated. While an outright 'jobs apocalypse' is not universally expected, significant job losses and role changes are likely, particularly in routine cognitive and clerical work ([W14]).

At the same time, new job categories are predicted to emerge, particularly in the design, oversight, ethics and integration of AI systems, requiring comprehensive reskilling of the workforce and changes to education policy.

Some experts suggest that AI will be routinely embedded in most commercial and administrative processes by 2030, with specialised models tailored to fields such as medicine, law, and finance becoming increasingly prevalent ([K24]).

6.4. Timelines toward general intellect

A particularly significant class of expert predictions concerns the arrival of more general, or human-level, AI capabilities. Opinion surveys and commentary reveal a broad spectrum of views on when AI might demonstrate human-level performance across various tasks, often referred to as artificial general intelligence (AGI). While some analyses report that leading researchers and forecasters now consider it likely that such capabilities will emerge within the next decade, others caution that fundamental scientific breakthroughs are still necessary before true generality can be achieved ([M35]).

These divergent timelines reflect deep uncertainty regarding the nature of intelligence, the limitations of current architectures, and the interaction between computing power, data, and algorithmic innovation. Notably, even relatively optimistic forecasts are accompanied by recommendations for simultaneous advances in safety research, governance frameworks, and international cooperation, with the aim of mitigating risk and maximising societal benefit.

6.5. Emerging risk scenarios of advanced AI systems

Although artificial intelligence offers significant scientific and economic potential, experts are increasingly warning that the next decade will also bring major risks. These concerns are no longer distant or hypothetical; multiple researchers, safety institutes and international panels argue that AI capabilities are advancing faster than our ability to evaluate, regulate or control them. Notably, the majority of anticipated risks over the next five to ten years do not arise from hypothetical superintelligence, but rather from powerful yet imperfect models embedded in complex human infrastructures.

This section summarises the most significant risks highlighted in recent expert analyses and global reports.

- **Loss of human control over advanced AI systems**. One of the most urgent concerns is that the rate at which AI systems are advancing may exceed human ability to understand or control them. David Dalrymple, a UK AI safety expert, warns that AI capabilities are improving so quickly – some skills doubling every eight months – that advanced systems may exceed human oversight within five years. Government evaluations already demonstrate that leading models can autonomously complete expert-level tasks and even attempt self-replication. Dalrymple

argues that if AI systems outperform humans before safety measures are fully developed, humans could lose control over critical areas of society ([G31]).

- **Economic disruption and widespread unemployment**. The International AI Safety Report, authored by 96 experts from 30 countries, warns that if AI development is not kept in check, it could lead to widespread unemployment as automation expands into high-skill cognitive work. It predicts that AI systems capable of performing the most economically valuable tasks more efficiently than humans could emerge within the next decade, resulting in structural labour displacement ([W15]).

Unlike bias or safety failures, however, this risk is not primarily the result of model error, but of system-level economic dynamics. Accordingly, the focus of mitigation strategies lies largely outside of model architecture and instead on policy-level interventions such as reskilling programmes, educational reform, adaptive labour market institutions and mechanisms for redistributing productivity gains.

- **AI-enabled terrorism and malicious use**. Many experts are increasingly concerned about dual-use risks: the same models that accelerate scientific discovery or software development may also lower barriers to cyberattacks. What makes these risks distinctive is that they arise from capability diffusion rather than system malfunction.

An international expert panel has warned that AI could significantly increase the risk of AI-enabled terrorism, including

- automated cyberattacks
- large-scale disinformation campaigns
- autonomous weaponisation ([229])
- biological or chemical threat modelling ([230]).

These risks are explicitly highlighted in the International AI Safety Report, which identifies AI-enabled terrorism as a major emerging threat ([W15]).

- **Organizational and corporate risk exposure**. Corporate risk analysts warn that, beyond global risks, AI will fundamentally reshape risk management within organisations. T3 Consultants predict that AI will become 'the corporate heart of the modern-day corporation', creating new vulnerabilities across all lines of defence within the next five years ([M36]). They also note that fewer than 15% of senior risk managers currently have the skills to safely integrate AI, leaving organisations exposed to:

- model failures
- AI data breaches ([231])
- regulatory non-compliance
- systemic operational risks.

Taken together, these risks highlight a key issue in contemporary AI research: the advancement of capabilities has outpaced our ability to reliably predict, control and govern the behaviour of systems in real-world contexts. Over the next decade, therefore, progress in AI will be defined not only by architectural innovation or scaling laws, but also by the development of robust alignment methods, evaluation frameworks and institutional safeguards.

---

[229] Autonomous weaponisation refers to AI systems that can independently identify and pursue targets based on learned patterns.

[230] Biological or chemical threat modelling refers to the potential misuse of AI systems for analytical tasks related to biological or chemical attacks, such as scenario comparison, vulnerability identification, or impact assessment. This lowers the planning and coordination barriers for individuals with limited domain expertise.

[231] An AI data breach occurs when sensitive data is exposed, stolen or compromised through artificial intelligence systems or technologies. Such incidents differ significantly from traditional data breaches in that they exploit the unique vulnerabilities of AI systems and models.

Acknowledgments

I would like to express my gratitude to Dr H. Kunde, who read the manuscript exceptionally carefully and provided a comprehensive set of corrections and insightful comments. His detailed feedback has greatly improved the clarity, accuracy and overall quality of the article.

I would also like to thank Mr Th. Kruse and Dr A. Spakowski for thoroughly reading earlier versions of the manuscript and for their helpful comments and corrections.

Any remaining errors are, of course, my own.

AI Assistance Disclosure

This manuscript was prepared with the assistance of a large language model. All conceptual development, formal arguments, and final editorial decisions were performed and validated by the author.

References

[A1] J. Aron, *Software tricks people into thinking it is human*, New Scientist, 2011. https://www.newscientist.com/article/dn20865-software-tricks-people-into-thinking-it-is-human

[A2] C. Aggarwal, *Neural Networks and Deep Learning*, Springer, 2023.

[A3] S. Abrahams et al., *TensorFlow for Machine Intelligence: A Hands-On Introduction to Learning Algorithms*, Bleeding Edge Press, 2016.

[A4] C. Aggarwal, *Complete Guide to the Adam Optimization Algorithm*, 2023, https://builtin.com/machine-learning/adam-optimization

[A5] R. Alake, *Loss Functions in Machine Learning Explained*, 2024, https://www.datacamp.com/tutorial/loss-function-in-machine-learning

[A6] K. Azad, *Intuitive Guide to Convolution*, https://betterexplained.com/articles/intuitive-convolution/

[A7] A. Amidi and S. Amidi, *Recurrent Neural Networks*, https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks#

[A8] H. Alqahtani et al., *Applications of Generative Adversarial Networks (GANs): An Updated Review*, Archives of Computational Methods in Engineering, 2019, 28(1).

[A9] K. Ahirwar, *Generative Adversarial Networks Projects*, Packt Publishing, 2019.

[A10] H. Aditya et al., *Evaluating Privacy Leakage and Memorization Attacks on Large Language Models (LLMs) in Generative AI Applications*, Journal of Software Engineering and Applications, 2024, 17, 421–447, https://www.scirp.org/pdf/jsea2024175_139303261.pdf

[A11] *Amazon Bedrock Documentation*, https://docs.aws.amazon.com/bedrock/

[A12] Amdad H, *New Claude 3.5 Sonnet vs. Claude 3.5 Haiku: Which Model is Best for Your Needs?*, 2024, https://medium.com/@amdadAI/new-claude-3-5-sonnet-vs-claude-3-5-haiku-which-model-is-best-for-your-needs-6f4db14cd5c3

[A13] *AI & Machine Learning: Identifying Opportunities & Challenges*, Forbes Technology Council, 2023, https://councils.forbes.com/blog/ai-and-machine-learning

[A14] *AI to drive 165% increase in data center power demand by 2030*, Goldman Sachs Research, 2025, https://www.goldmansachs.com/insights/articles/ai-to-drive-165-increase-in-data-center-power-demand-by-2030

[A15] *Agentic AI Explained: Benefits, Drawbacks, and Real-World Limitations*, 2025, https://simplifyai.in/2025/06/agentic-ai-explained-benefits-drawbacks-and-real-world-limitations/

[A16] J. Aston, *The efficient use of tokens for multi-agent systems*, Capgemini, 2024, https://www.capgemini.com/insights/expert-perspectives/ai-lab-the-efficient-use-of-tokens-for-multi-agent-systems/

[A17] *Anthropic Claude Sonnet 4 and Claude Opus 4 are now generally available in GitHub Copilot*, 2025, https://github.blog/changelog/2025-06-25-anthropic-claude-sonnet-4-and-claude-opus-4-are-now-generally-available-in-github-copilot/

[A18] Adaptive Team, *Arup's $25M Deepfake Loss: Anatomy of an AI-Powered Scam*, 2025, https://www.adaptivesecurity.com/blog/arup-deepfake-scam-attack

[A19] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*, 2009, Cambridge University Press.

[A20] M. Allen, Sneak peek: IBM's 4-year AI forecast, 2026, https://www.axios.com/2026/01/18/sneak-peek-ibms-4-year-ai-forecast

[B1] *Encyclopedia Britannica*, London, 1991.

[B2] E. T. Bell, *The Development of Mathematics*, 2nd edition, McGraw-Hill Book Company.

[B3] J. Biba, *4 Types of Machine Learning to Know*, 2024, https://builtin.com/machine-learning/types-of-machine-learning

[B4] Ch. Borgelt, *An Implementation of the FP-growth Algorithm*, Otto-von-Guericke-University of Magdeburg, https://christian.borgelt.net/papers/fpgrowth.pdf

[B5] D. Bergmann, *What is self-supervised learning?*, 2023, https://www.ibm.com/think/topics/self-supervised-learning

[B6] D. Bergmann and C. Stryker, *What is an autoencoder?*, 2023, https://www.ibm.com/think/topics/autoencoder

[B7] D. Bergmann and C. Stryker, *What is an attention mechanism?*, 2024, https://www.ibm.com/think/topics/attention-mechanism

[B8] V. Braitenberg and A. Schutz, *Anatomy of the cortex: Studies of brain function*, Springer, 1991.

[B9] J. Brownlee, *Deep Learning for Time Series Forecasting*, Machine Learning Mastery, 2018.

[B10] D. Barber, *Bayesian Reasoning and Machine Learning*, Cambridge University Press, 2018.

[B11] Britannica, https://www.britannica.com/biography/Johannes-Vermeer

[B12] D. Bergmann and C. Stryker, *What is a variational autoencoder?*, 2024, https://www.ibm.com/think/topics/variational-autoencoder

[B13] D. Bergmann and C. Stryker, *What are diffusion models?*, 2024, https://www.ibm.com/think/topics/diffusion-models

[B14] D. Birtolo, *How to use GitHub Copilot: What it can do and real-world examples*, 2025, https://github.blog/ai-and-ml/github-copilot/what-can-github-copilot-do-examples/

[B15] I. Belcic and C. Stryker, *What is Claude AI?*, 2024, https://www.ibm.com/think/topics/claude-ai

[B16] T. Braun and Z. Maufe, *Five generative AI use cases for the financial services industry*, 2023, https://cloud.google.com/blog/topics/financial-services/five-generative-ai-use-cases-financial-services-industry?hl=en

[B17] R. Borah, *Top 9 GenAI Use Cases in Pharma & Healthcare*, 2023, https://blog.gramener.com/generative-ai-use-cases-in-pharma-healthcare/

[B18] S. Brotons, *The Limitations of Generative AI*, https://lingarogroup.com/blog/the-limitations-of-generative-ai-according-to-generative-ai

[B19] R. Belan, *Microsoft taps Three Mile Island nuclear plant to power AI*, 2024, https://techcrunch.com/2024/09/20/microsoft-taps-three-mile-island-nuclear-plant-to-power-ai/

[B20] J. Brownlee, *How to Choose an Optimization Algorithm*, 2021, https://machinelearningmastery.com/tour-of-optimization-algorithms/

[B21] J. Brownlee, *No Free Lunch Theorem for Machine Learning*, 2021, https://machinelearningmastery.com/no-free-lunch-theorem-for-machine-learning/

[B22] R.A. Bhalerao, *When Agentic AI Goes Wrong: 7 Recent Failures Leaders Cannot Ignore*, 2025, https://www.linkedin.com/pulse/when-agentic-ai-goes-wrong-7-recent-failures-leaders-cannot-bhalerao-xshoe

[B23] V. Balasubramanian, Brain Power, PNAS 118(32), 2021, 1–3.

[B24] E.C. Berkley, *Giant Brains, or Machines That Think*, Wiley & Sons, 1949.

[B25] H. Blockeel et al., *Decision trees from efficient prediction to responsible AI*, Front. Artif. Intell. 6, 2023, https://doi.org/10.3389/frai.2023.1124553

[B26] L. Breiman et al., *Classification and Regression Trees*, Taylor & Francis, 1984.

[B27] K. Budzyn et al., *Endoscopist deskilling risk after exposure to artificial intelligence incolonoscopy: a multicentre, observational study*, Lancet Gastroenterology & Hepatology, 10, 2025, 896–903.

[B28] J. Barnard, *What are word embeddings?*, https://www.ibm.com/think/topics/word-embeddings

[B29] *Backpropagation and Gradients*, https://cs231n.stanford.edu/slides/2018/cs231n_2018_ds02.pdf

[B30] T. B. Brown et al., *Language Models are Few-Shot Learners*, Proc. Advances in Neural Information Processing Systems (NeurIPS), 2020, 1877–1901, https://arxiv.org/pdf/2005.14165

[B31] A. Belanger, *Air Canada must honor refund policy invented by airline's chatbot*, 2024, https://arstechnica.com/tech-policy/2024/02/air-canada-must-honor-refund-policy-invented-by-airlines-chatbot/

[B32] D. Bergmann, *What is mixture of experts?*, https://www.ibm.com/think/topics/mixture-of-experts

[B33] E. M. Bender et al., *On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?*, 2021, https://s10251.pcdn.co/pdf/2021-bender-parrots.pdf

[B34] D. Bahdanau et al, *Neural Machine Translation by Jointly Learning to Align and Translate*, 2014. Published as a conference paper at ICLR 2015, https://arxiv.org/pdf/1409.0473

[B35] T. Bai et al., *A Survey of Multimodal Large Language Model from A Data-centric Perspective*, 2024, https://arxiv.org/pdf/2405.16640v2

[C1] *Covariance Matrix*, 2024, https://www.geeksforgeeks.org/covariance-matrix/

[C2] J. Cardete, *Convolutional Neural Networks: A Comprehensive Guide*, 2024, https://medium.com/thedeephub/convolutional-neural-networks-a-comprehensive-guide-5cc0b5eae175

[C3] K. Cho et al., *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*, 2014, https://arxiv.org/abs/1406.1078

[C4] K. Crawford, *Generative AI's environmental costs are soaring – and mostly secret*, Nature, 2024, https://www.nature.com/articles/d41586-024-00478-x

[C5] C. Cesaric, *AI Data Centers Are Coming for Your Land, Water and Power*, 2025, https://www.cnet.com/tech/services-and-software/features/ai-data-centers-are-coming-for-your-land-water-and-power/

[C6] A. Challapally et al., *State of AI in Business 2025*, MIT's Media Lab (Project NANDA), 2025, https://mlq.ai/media/quarterly_decks/v0.1_State_of_AI_in_Business_2025_Report.pdf

[C7] *Challenges and Solutions for Building Effective Recommendation Systems*, AI & Machine Learning Blog, 2023, https://www.itconvergence.com/blog/challenges-and-solutions-for-building-effective-recommendation-systems/

[C8] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and other Kernel-Based Learning Methods*, Cambridge University Press, 2014.

[C9] M. Chen et al., Opportunities and challenges of diffusion models for generative AI, National Science Review, 11(12), 2024, https://doi.org/10.1093/nsr/nwae348

[C10] S. Chokshi et al., *Machine Learning operations maturity model*, Microsoft, 2024, https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/mlops-maturity-model

[C11] X. Cheng et al., *Introduction to Natural Language Processing*, River Publishers, 2025.

[C12] Calibraint, *Mastering Tokenization in NLP: An In-Depth Look at Methods, Types, and Challenges*, 2024, https://www.calibraint.com/blog/understanding-tokenization-in-nlp-guide

[C13] A. Chowdhery et al., *PaLM: Scaling Language Modeling with Pathways*, 2022, https://arxiv.org/pdf/2204.02311

[C14] R. Colin et al., *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*, Journal of Machine Learning Research, 21, 2020, 1–67, https://jmlr.org/papers/volume21/20-074/20-074.pdf

[C15] E. Chalk, *Prompt Engineering with Chain of Thought for ChatGPT 4*, Self-Published via Amazon KDP, 2024.

[C16] X. Chen et al., *Reasoning Beyond Language: A Comprehensive Survey on Latent Chain-of-Thought Reasoning*, 2025, https://arxiv.org/pdf/2505.16782v1

[D1] *Deep Learning Tutorial*, 2024, https://www.geeksforgeeks.org/deep-learning-tutorial/

[D2] M. Donges, *A Guide to Recurrent Neural Networks (RNNs)*, 2024, https://builtin.com/data-science/recurrent-neural-networks-and-lstm

[D3] *Deep Learning Demystified,* University of Southern California, https://caisplusplus.usc.edu/curriculum/neural-network-flavors/recurrent-neural-networks

[D4] *Derivative of Tanh Function*, https://blogs.cuit.columbia.edu/zp2130/derivative_of_tanh_function/

[D5] *Deep Learning vs Machine Learning vs Neural Networks: What's the Difference and Why It Matters for Your Business*, 2025, https://blog.purestorage.com/purely-educational/deep-learning-vs-machine-learning-vs-neural-networks/

[D6] E. Denton et al., *Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks*, 2015, https://arxiv.org/abs/1506.05751

[D7] DeepSeek R1 1776, 2025, https://www.perplexity.ai/hub/blog/open-sourcing-r1-1776

[D8] C. Dilmegani and N. Şipi, *Top 5 Open-Source Agentic Frameworks in 2025*, https://research.aimultiple.com/agentic-frameworks/

[D9] H.L. Dreyfus, *Alchemy and Artificial Intelligence*, 1965, https://www.rand.org/content/dam/rand/pubs/papers/2006/P3244.pdf

[D10] *Decision Trees: How They Work and Practical Examples*, 2024, https://keylabs.ai/blog/decision-trees-how-they-work-and-practical-examples/

[D11] Dulac-Arnold et al., *Challenges of real-world reinforcement learning: definitions, benchmarks and analysis*, Machine Learning, 2021, 110, 2419–2468, https://doi.org/10.1007/s10994-021-05961-4

[D12] J. Devlin et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, Proc. NAACL-HLT, 2019, 4171–4186, https://arxiv.org/pdf/1810.04805

[D13] T. Dao et al., *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*, 2022, https://arxiv.org/pdf/2205.14135

[D14] DeepSeek-AI, *DeepSeek-V3 Technical Report*, https://arxiv.org/abs/2412.19437

[D15] D. Dai et al., *DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models*, 2024, https://arxiv.org/abs/2401.06066

[D16] T. Dao, *FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning*, 2023, https://arxiv.org/abs/2307.08691

[E1] W. Ertel, *Introduction to Artificial Intelligence*, Springer, 2024.

[E2] R.G. Eccles, *Microsoft Can Take The Lead In Small Modular Reactors For Powering AI*, 2024, https://www.forbes.com/sites/bobeccles/2024/08/31/microsoft-can-take-the-lead-in-small-modular-reactors-for-powering-ai/

[E3] D.S. Excel, *Autonomous Systems: challenges and opportunities*, World Journal of Advanced Research and Reviews, 23(03), 2024, 631–641, https://doi.org/10.30574/wjarr.2024.23.3.2727

[E4] European Defence Agency, *Autonomous Systems in Defence: Technology Trends and Military Applications*, Brussels, Belgium, EDA Rep., Mar. 2023.

[E5] M. Elayyan, *Real-Life AI Disasters 5 Cases That Made Headlines And What We Can Learn*, 2025, https://blogs.fsd-tech.com/ai-disasters-2025

[E6] C. van Eeden, *A Social Media Network Exclusively For AI Agents: Is This The Next Privacy Issue?*, 2026, https://techreport.com/news/is-moltbook-the-next-privacy-issue/

[F1] B.C. Van Fraassen, *Relative Frequencies*, in C. Salmon (ed.), Hans Reichenbach: Logical Empiricist, 1977, 129-167.

[F2] *Financing Infrastructure for a Competitive European AI*, 2025, https://geopolitique.eu/en/2025/02/10/financing-infrastructure-for-a-competitive-european-ai/

[F3] D. Foster, *Generative Deep Learning*, O'Reilly Media, Inc., 2023.

[F4] M. Finio and A. Downie, *AI in software development*, https://www.ibm.com/think/topics/ai-in-software-development

[F5] M. Fauscette, *Understanding the Limitations and Challenges of Generative AI*, 2024, https://em360tech.com/tech-articles/understanding-limitations-and-challenges-generative-ai

[F6] T. Farnschläder, *AI Hallucination: A Guide With Examples*, 2025, https://www.datacamp.com/blog/ai-hallucination

[F7] S. Forgar, *'This collective hysteria surrounding AI has to stop; it's all completely exaggerated.'*, (in German), Die WELT, 30 July 2025, https://www.welt.de/kultur/plus687df68f8598e36a33a8badc/KI-Die-Vermenschlichung-dieser-KI-ist-gefaehrlich.html

[F8] J. Fiegenbaum, *AI's Growing Energy Demand: Challenges and Sustainable Solutions for Data Centers*, 2025, https://www.fiegenbaum.solutions/en/blog/ai-energy-demand-sustainable-data-centers

[F9] T. Finn and A. Downie, *Agentic AI vs. generative AI*, 2025, https://www.ibm.com/think/topics/agentic-ai-vs-generative-ai

[F10] R. Fletcher, *Practical methods of optimization*, Wiley, 2000.

[G1] D. Ghosh, *ChatGPT Passed The Turing Test*, 2024, https://www.howtogeek.com/chatgpt-passed-the-turing-test-heres-what-that-means/

[G2] T. Ganegedara, *TensorFlow in Action*, Manning Publications, 2022.

[G3] A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*, O'Reilly Media, 2019.

[G4] M. Ghasemi and D. Ebrahimi, *Introduction to Reinforcement Learning*, 2024, https://arxiv.org/pdf/2408.07712

[G5] *Generative Adversarial Network (GAN)*, 2025, https://www.geeksforgeeks.org/generative-adversarial-network-gan/

[G6] N. Gillis, *The Why and How of Nonnegative Matrix Factorization*, 2024, https://arxiv.org/pdf/1401.5226

[G7] I. Goodfellow et al., *Generative Adversarial Nets*, 2014, https://arxiv.org/abs/1406.2661

[G8] G. Gundersen, *From Convolution to Neural Network*, 2017, https://gregorygundersen.com/blog/2017/02/24/cnns/

[G9] I. Goodfellow et al., *Deep Learning*, MIT Press, 2016, https://www.deeplearningbook.org/

[G10] *Gated Recurrent Unit Networks*, 2025, https://www.geeksforgeeks.org/machine-learning/gated-recurrent-unit-networks/

[G11] *Gated Recurrent Unit (GRU)*, https://www.ultralytics.com/glossary/gated-recurrent-unit-gru

[G12] *Gradient descent*, https://en.wikipedia.org/wiki/Gradient_descent

[G13] M. Ghayoumi, *Generative Adversarial Networks in Practice*, CRC Press, 2024.

[G14] Google Developers, *Advanced courses - GAN*, https://developers.google.com/machine-learning/gan

[G15] *Generative adversarial networks*, https://deepgenerativemodels.github.io/notes/gan/

[G16] *Generative AI solutions for developers*, 2025, https://learn.microsoft.com/en-us/azure/developer/ai/introduction-build-generative-ai-solutions

[G17] *Gartner Experts Answer the Top Generative AI Questions for Your Enterprise*, https://www.gartner.com/en/topics/generative-ai

[G18] *Google's Secure AI Framework*, https://www.saif.google

[G19] A. Granskog et al., *The role of power in unlocking the European AI revolution*, 2024, https://www.mckinsey.com/industries/electric-power-and-natural-gas/our-insights/the-role-of-power-in-unlocking-the-european-ai-revolution

[G20] Z. Guo et al., *A Comprehensive Review on Noise Control of Diffusion Model*, 2025, https://arxiv.org/pdf/2502.04669v1

[G21] F. Greselin and R. Zitikis, *From the Classical Gini Index of Income Inequality to a NewZenga-Type Relative Measure of Risk: A Modeller's Perspective*, Econometrics, 6(1), 2018, 1-20, https://doi.org/10.3390/econometrics6010004

[G22] B. Ghojogh et al., *Reproducing Kernel Hilbert Space, Mercer's Theorem, Eigenfunctions, Nyström Method, and Use of Kernels in Machine Learning:* Tutorial and Survey, 2021, https://arxiv.org/pdf/2106.08443v

[G23] Gaviraj K, *Understanding the risks of random forests in machine learning*, 2025, https://www.byteplus.com/en/topic/471930

[G24] V. G. Goecks and N. Waytowich, *COA-GPT: Generative Pre-trained Transformers for Accelerated Course of Action Development in Military Operations*, 2024, https://arxiv.org/pdf/2402.01786

[G25] A. Gilpin et al., *Agentic AI in Defence: Autonomous Agents for Multi-Domain Military Decision Support*, Journal of Defence Technology and Strategy, 8(2), 2024, 112–128.

[G26] M. Garouani and M. Bouneffa, *Automated Machine Learning Hyperparameters Tuning through Meta-Guided Bayesian Optimization, Progress in Artificial Intelligence*, 2024, https://mgarouani.fr/publications/pai/

[G27] M. Griffin, *OpenAI GPT-5 is costing $500 Million per training run and still failing*, 2025, https://www.fanaticalfuturist.com/2025/05/openai-gpt-5-is-costing-500-million-per-training-run-and-still-failing/

[G28] G. Gupta, *Understanding the T5 Model: A Comprehensive Guide*, 2024, https://medium.com/@gagangupta_82781/understanding-the-t5-model-a-comprehensive-guide-b4d5c02c234b

[G29] A. Gu et al., *Efficiently Modeling Long Sequences with Structured State Space Models*, 2022, https://arxiv.org/pdf/2111.00396

[G30] K. Grace et al., *Thousands of AI authors on the future of AI*, 2024, https://arxiv.org/pdf/2401.02843

[G31] Ground News, *AI Systems May Outpace Human Control Within Five Years*, 2026, https://ground.news/daily-briefing/uk-expert-warns-world-may-lack-time-to-prepare-for-ai-safety-risks

[H1] J. Heaton, *Artificial Intelligence for Humans, Volume 1: Fundamental Algorithms*, Heaton Research, Inc., 2013.

[H2] G. Hinton et al., *Distilling the Knowledge in a Neural Network*, 2015, https://arxiv.org/pdf/1503.02531

[H3] J. Hui, *GAN –What is Generative Adversarial Networks GAN?*, 2018, https://jonathan-hui.medium.com/gan-whats-generative-adversarial-networks-and-its-application-f39ed278ef09

[H4] K. He et al., *Deep Residual Learning for Image Recognition*, 2015, https://arxiv.org/pdf/1512.03385

[H5] Y. H. Hwang, *C# Machine Learning Projects*, Packt Publishing, 2018.

[H6] G. Huang et al., *Densely Connected Convolutional Networks*, 2018, https://arxiv.org/abs/1608.06993

[H7] D. Hubel and T. Wiesel, *Receptive fields of single neurones in the cat's striate cortex*, The Journal of Physiology, 124(3), 1959, 574–591.

[H8] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural Computation, 1997, 9(8), 1735–1780.

[H9] K. Hui, *Techniques of Feature Visualisation – Deconvolutional Network in Convolutional Neural Networks*, 2024, https://medium.com/@hke22/techniques-of-feature-visualisation-deconvolutional-network-in-convolutional-neural-networks-5a086aecbed3

[H10] M. Huddle et al., *Generative AI Will Transform Health Care Sooner Than You Think*, 2023, https://www.bcg.com/publications/2023/how-generative-ai-is-transforming-health-care-sooner-than-expected

[H11] C. Hashemi-Pour, *What is Amazon Bedrock*, 2024, https://www.techtarget.com/searchenterpriseai/definition/Amazon-Bedrock-AWS-Bedrock

[H12] G. Hinton, *Learning Multiple Layers of Representation*, 2006, doi:10.1016/j.tics.2007.09.004

[H13] M. Haugh, *Support Vector Machines (and the Kernel Trick)*, Columbia University, https://www.columbia.edu/~mh2078/MachineLearningORFE/SVMs_MasterSlides.pdf

[H14] *Hierarchical Clustering in Machine Learning*, 2025, https://www.geeksforgeeks.org/machine-learning/hierarchical-clustering/

[H15] J. Hui, *GAN – Why it is so hard to train Generative Adversarial Networks!*, 2018, https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

[H16] T. Hastie et al., *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer, 2009, https://www.sas.upenn.edu/~fdiebold/NoHesitations/BookAdvanced.pdf

[H17] D. Hendrycks and K. Gimpel, *Gaussian Error Linear Units (GELUs)*, 2016, https://arxiv.org/pdf/1606.08415

[H18] Y. Huang et al., *Advancing Transformer Architecture in Long-Context Large Language Models: A Comprehensive Survey*, 2024, https://arxiv.org/pdf/2311.12351v2

[I1] *Introduction to Recurrent Neural Networks*, 2025, https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

[I2] IBM, *What are AI hallucinations?*, https://www.ibm.com/think/topics/ai-hallucinations

[I3] IBM, *The hidden costs of AI: How generative models are reshaping corporate budgets*, IBM Think Insights, 2024, https://www.ibm.com/think/insights/ai-economics-compute-cost

[I4] M.R. Islam and A.T. Wasi, *Balancing Power and Ethics: A Framework for Addressing Human Rights Concerns in Military AI*, 2024, https://arxiv.org/abs/2411.06336

[J1] Ph. C. Jackson, Jr., *Introduction to Artificial Intelligence*, Dover Publications, Inc., New York, 1985.

[J2] A. Jain, *All about images and their formats like YUV, RGB, HSL, HSV, BGR, binary, CMYK, RGBA and YIQ*, 2024, https://medium.com/@abhishekjainindore24/all-about-images-and-their-formats-1bcba5c854e7

[J3] A. Jaiswal, *Unveiling Deep Recurrent Model Architectures: RNNs, LSTMs, GRUs, and Attention Mechanisms*, 2024, https://medium.com/@adityaj5400/unveiling-deep-recurrent-model-architectures-rnns-lstms-grus-and-attention-mechanisms-6dbeefe8fe2f

[J4] *Jukebox Explained: How To Generate AI Music Like a Pro*, https://www.upwork.com/resources/jukebox-openai

[J5] S. Joshi, *Agentic Generative AI and National Security: Policy Recommendations for US Military Competitiveness*, https://ssrn.com/abstract=5529680

[J6] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed. Draft, 2023.

[J7] P. Jauk et al., *The relationship between intelligence and creativity: New support for the threshold hypothesis by means of empirical breakpoint detection*, Intelligence, 41(4), 2013, 212–221, https://doi.org/10.1016/j.intell.2013.03.003

[J8] I. John, *The Art of Asking ChatGPT for High-Quality Answers*, Nzunda Technologies Limited, 2023.

[K1] E. Kleppen, *What Is the Turing Test?*, 2025, https://builtin.com/artificial-intelligence/turing-test

[K2] A. Khrennikov, *Interpretations of Probability*, Walter de Gruyter, 2009.

[K3] A. Kowalczyk, *Support Vector Machines Tutorial*, https://www.svm-tutorial.com/

[K4] R. Kwiatkowski, *Gradient Descent Algorithm – a deep dive*, 2021, https://medium.com/data-science/gradient-descent-algorithm-a-deep-dive-cf04e8115f21

[K5] R. Kumar, *A Guide to the DBSCAN Clustering Algorithm*, 2024, https://www.datacamp.com/tutorial/dbscan-clustering-algorithm

[K6] P. Krauss, *Artificial Intelligence and Brain Research*, Springer, 2024.

[K7] D. Kalita, *What is Recurrent Neural Network (RNN)*?, 2025, https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn

[K8] V. Kakaraparthi, *Xavier and He Normal (He-et-al) Initialization*, 2018, https://prateekvishnu.medium.com/xavier-and-he-normal-he-et-al-initialization-8e3d7a087528

[K9] T. Karas et al., A Style-Based Generator Architecture for Generative Adversarial Networks, 2019, https://arxiv.org/pdf/1812.04948

[K10] D.P. Kingma and M. Welling, *Auto-Encoding Variational Bayes*, 2022, https://arxiv.org/pdf/1312.6114

[K11] V. Krishnakumar, *Amazon Bedrock vs Azure OpenAI vs Google Vertex AI: An In-Depth Analysis*, 2025, https://www.cloudoptimo.com/blog/amazon-bedrock-vs-azure-openai-vs-google-vertex-ai-an-in-depth-analysis/

[K12] A. Kargwal, *Everything you should know about GPT-5,* 2025, https://botpress.com/blog/everything-you-should-know-about-gpt-5

[K13] M. Kandemir, *Why AI uses so much energy—and what we can do about it*, 2025, https://iee.psu.edu/news/blog/why-ai-uses-so-much-energy-and-what-we-can-do-about-it

[K14] L. Kumili, *Foundation Models vs. AI Agents vs. Agentic AI: What's the Difference?*, 2025, https://medium.com/@leela.kumili/foundation-models-vs-ai-agents-vs-agentic-ai-whats-the-difference-02bd58d3a0d3

[K15] R. Kundu, *F1 Score in Machine Learning: Intro & Calculation*, 2022, https://www.v7labs.com/blog/f1-score-guide

[K16] Ch. Khanna, *Byte-Pair Encoding: Subword-based tokenization algorithm*, 2021, https://medium.com/data-science/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0

[K17] Ch. Khanna, *WordPiece: Subword-based tokenization algorithm*, 2021, https://towardsdatascience.com/wordpiece-subword-based-tokenization-algorithm-1fbd14394ed7

[K18] J. Kirkpatrick et al., *Overcoming Catastrophic Forgetting in Neural Networks*, Proceedings of the National Academy of Sciences, 114(13), 2017, 3521–3526, https://arxiv.org/pdf/1612.00796

[K19] J. Kaplan et al., *Scaling Laws for Neural Language Models*, 2020, https://arxiv.org/pdf/2001.08361

[K20] N. Koenigstein, *Transformers: The Definitive Guide*, O'Reilly Media, Inc., 2025.

[K21] S. I. Kampezidou et al., *Fundamental Components and Principles of Supervised Machine Learning Workflows with Numerical and Categorical Data*, Eng 5, 2024, 384-416, https://doi.org/10.3390/eng5010021

[K22] N. Kosmyna et al., *Your Brain on ChatGPT: Accumulation of Cognitive Debt when Using an AI Assistant for Essay Writing Task*, 2025, https://arxiv.org/pdf/2506.08872

[K23] A. Kosowski et al., *The Dragon Hatchling: The Missing Link Between Transformer Models and Models of the Brain*, 2025, https://arxiv.org/pdf/2509.26507

[K24] I. Korucuoğlu, *The Next Decade in AI: Experts' Predictions*, 2025, https://www.siberoloji.com/the-next-decade-in-ai-experts-predictions

[L1] *Logistic Regression in Machine Learning*, 2025, https://www.geeksforgeeks.org/understanding-logistic-regression/

[L2] F. vom Lehn, *Understanding the Convolutional Filter Operation in CNN's*, 2023, https://medium.com/advanced-deep-learning/cnn-operation-with-2-kernels-resulting-in-2-feature-mapsunderstanding-the-convolutional-filter-c4aad26cf32

[L3] F. vom Lehn, *Understanding the Structure of RGB Images and How Pixel Values Represent Color*, 2023, https://medium.com/advanced-deep-learning/decoding-image-representation-understanding-the-structure-of-rgb-images-6a211eb8800d

[L4] L. Lim, *Challenges with convolutional neural networks (CNN)*, 2025, https://www.byteplus.com/en/topic/401523

[L5] Z. Li et al., *Accuracy of training data and model outputs in Generative AI: CREATe Response to the Information Commissioner's Office*, https://arxiv.org/pdf/2407.13072

[L6] Llama 3.1 Sonar Large, https://relevanceai.com/llm-models/unlock-the-power-of-llama-3-1-sonar-large-128k-online

[L7] B. Linders, *The Rise of Energy and Water Consumption Using AI Models, and How It Can Be Reduced*, 2025, https://www.infoq.com/news/2025/06/energy-water-consumption-AI/

[L8] K. Leung, *Failed Machine Learning (FML)*, https://github.com/kennethleungty/Failed-ML

[L9] J. Lighthill, *Artificial Intelligence: A General Survey*, 1972, https://www.chilton-computing.org.uk/inf/literature/reports/lighthill_report/p001.htm

[L10] *Linear Regression in Machine learning*, 2025, https://www.geeksforgeeks.org/machine-learning/ml-linear-regression/

[L11] S. Lee, *Ridge Regression Applications in Computer Science*, 2025, https://www.numberanalytics.com/blog/ridge-regression-computer-science-applications

[L12] *Limitations of Decision Tree*, 2025, https://www.geeksforgeeks.org/machine-learning/limitations-of-decision-tree/

[L13] S. Lee, *Practical Decision Trees: Real-World Examples and Implementation Tips*, 2025, https://www.numberanalytics.com/blog/practical-decision-trees-real-world-examples-implementation-tips

[L14] L. Lim, *Challenges with k-nearest neighbors (KNN): Navigating the algorithm's limitations*, 2025, https://www.byteplus.com/en/topic/400925?title=challenges-with-k-nearest-neighbors-knn-navigating-the-algorithm-s-limitations

[L15] *Limitations of Traditional RNN Approaches*, https://apxml.com/courses/introduction-to-transformer-models/chapter-1-sequence-modeling-attention-fundamentals/rnn-limitations

[L16] P. Lewis et al., *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, Proc. NeurIPS, 2020, https://proceedings.neurips.cc/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf

[L17] M. Lewis et al., *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*, 2019, https://arxiv.org/pdf/1910.13461

[L18] P. Liu et al., *Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing*, 2021, https://arxiv.org/pdf/2107.13586

[L19] *Latent Reasoning in LLMs*, Emergend Mind, 2015, https://www.emergentmind.com/topics/latent-reasoning-in-large-language-models

[L20] M-T. Luong et al., *Effective Approaches to Attention-based Neural Machine Translation*, 2015, https://arxiv.org/pdf/1508.04025

[M1] T. Mucci, *The history of artificial intelligence*, 2024, https://www.ibm.com/think/topics/history-of-artificial-intelligence

[M2] T. Mucci, *The future of artificial intelligence*, 2024, https://www.ibm.com/think/insights/artificial-intelligence-future

[M3] J. Mitchell, *Key Components of Artificial Intelligence (AI)*, 2024, https://futureskillsacademy.com/blog/key-components-of-ai

[M4] E. McIrvine and M. Tribus, *Energy and Information*, Scientific American, 225(3), 1971, 179-190.

[M5] J. Murel et al., *What is ridge regression?*, 2023, https://www.ibm.com/think/topics/ridge-regression

[M6] A. Moitra, *Gaussian Mixture Models*, 2015, https://ocw.mit.edu/courses/18-409-algorithmic-aspects-of-machine-learning-spring-2015/e339520c4069ca5e785b29a3c604470e_MIT18_409S15_chapp6.pdf

[M7] *Medical Open Network for Artificial Intelligence (MONAI)*, https://monai.io/index.html

[M8] N.K. Manaswi, *Generative Adversarial Networks with Industrial Use Cases*, BPB Publications, 2020.

[M9] B.R. Mitchell, *The spatial inductive bias of deep learning*, Johns Hopkins University, 2017, https://jscholarship.library.jhu.edu/server/api/core/bitstreams/9e493d10-ccd6-4965-9bbd-c14325d97880/content

[M10] G. Mikriukov et al., *Unveiling the Anatomy of Adversarial Attacks: Concept-Based XAI Dissection*

of CNNs, In: Longo, L., Lapuschkin, S., Seifert, C. (eds) Explainable Artificial Intelligence. xAI 2024, Communications in Computer and Information Science, vol 2153. Springer, Cham, https://link.springer.com/chapter/10.1007/978-3-031-63787-2_6

[M11] S. Murray, *Does AI Limit Our Creativity?*, 2025, https://knowledge.wharton.upenn.edu/article/does-ai-limit-our-creativity/

[M12] S. Mittal, *The Leading AI Ecosystem Models: A Comprehensive Comparison*, 2024, https://techcommunity.microsoft.com/blog/azure-ai-foundry-blog/the-leading-ai-ecosystem-models-a-comprehensive-comparison/4360190

[M13] MusicLM, https://musiclm.com/

[M14] M. Mezger and H. Chen, *A brief introduction to GPT-4*, 2023, https://www.adesso.de/en/news/blog/a-brief-introduction-to-gpt-4-2.jsp

[M15] M. McMillan, What is PaLM 2? Everything you need to know about Google's new AI model, 2023, https://www.tomsguide.com/news/google-palm-2

[M16] MLOps, *An Overview of the End-to-End Machine Learning Workflow*, https://ml-ops.org/content/end-to-end-ml-workflow

[M17] *Machine Learning Tutorial*, 2025, https://www.geeksforgeeks.org/machine-learning/machine-learning/

[M18] *Meta's Hyperion AI Data Center Promises Big: $10 Billion Investment Just for Next-Gen AI Model Training?*, https://theusaleaders.com/news/hyperion-ai-data-center/

[M19] McKinsey Explainers, *What is multimodal AI?*, 2025, https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-multimodal-ai

[M20] W.S. McCulloch and W. Pitts, *A logical calculus of the ideas immanent in nervous activity*, Bulletin of Mathematical Biophysics 5, 1943, 115–133, https://doi.org/10.1007/BF02478259

[M21] J. McCarthy et al., *A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence*, 1955, https://doi.org/10.1609/aimag.v27i4.1904

[M22] *A. Mahajan et al., Artificial Intelligence in Robotics and its Advancements, Challenges and Ethical Considerations: A Review*, International Journal of Engineering Research & Technology (IJERT), 12(3), 2024, https://www.ijert.org/research/artificial-intelligence-in-robotics-and-its-advancements-challenges-and-ethical-considerations-a-review-IJERTCONV12IS03001.pdf

[M23] S. Milano et al., *Recommender systems and their ethical challenges*, AI & Soc 35, 2020, 957–967, https://doi.org/10.1007/s00146-020-00950-y

[M24] *Microsoft and OpenAI evolve partnership to drive the next phase of AI*, Microsoft Corporate Blogs, 2025, https://blogs.microsoft.com/blog/2025/01/21/microsoft-and-openai-evolve-partnership-to-drive-the-next-phase-of-ai/

[M25] J. Mercer, *Functions of positive and negative type and their connection with the theory of integral equations*, Philosophical Transactions of the Royal Society A, 209, 1909, 415–446.

[M26] *Minkowski Distance*, 2025, https://www.geeksforgeeks.org/maths/minskowski-distance/

[M27] C.D. Manning et al., *Introduction to Information Retrieval*, 2008, Cambridge University Press, https://www-nlp.stanford.edu/IR-book/

[M28] T. Mikolov et al., *Efficient Estimation of Word Representations in Vector Space*, 2013, https://arxiv.org/pdf/1301.3781

[M29] M. Müller et al., *Industrial autonomous systems: A survey on definitions, characteristics and abilities*, Automatisierungstechnik, 69(1), 2021, 103-117.

[M30] C.D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*, MIT Press, 1999.

[M31] J. Murel and E. Kavlakoglu, *What is stemming?*, https://www.ibm.com/think/topics/stemming

[M32] J. Murel and E. Kavlakoglu, *What is bag of words?*, https://www.ibm.com/think/topics/bag-of-words

[M33] T. Mikolov et al., *Efficient Estimation of Word Representations in Vector Space*, 2013, https://arxiv.org/abs/1301.3781

[M34] J. Murel and J. Noble, *What is latent semantic analysis?*, https://www.ibm.com/think/topics/latent-semantic-analysis

[M35] V. C. Müller and N. Bostrom, *Future progress in artificial intelligence: A survey of expert opinion*, 2025, https://arxiv.org/pdf/2508.11681

[M36] MarketersMedia, *AI Is Redefining Risk: T3 Consultants Reveal What the Next Five Years Hold for Risk Professionals*, 2025, https://news.marketersmedia.com/ai-is-redefining-risk-t3-consultants-reveal-what-the-next-five-years-hold-for-risk-professionals/89163353

[N1] J. Nabi, *Machine Learning Basics Every Beginner Should Know*, 2024, https://builtin.com/machine-learning/machine-learning-basics

[N2] J. Noble, *What is the Apriori algorithm?*, 2024, https://www.ibm.com/think/topics/apriori-algorithm

[N3] M. Nagpal, *8 Deep Learning Architectures Data Scientists Must Master*, 2024, https://www.projectpro.io/article/deep-learning-architectures/996

[N4] J. Noffsinger et al., *The cost of compute: A $7 trillion race to scale data centers*, McKinsey Quarterly, April 2025.

[N5] B. Nkomo, *Convolutional Neural Networks — Part 1: Edge Detection*, 2020, https://medium.com/swlh/convolutional-neural-networks-22764af1c42a

[O1] OpenAI, *Hello GPT-4o*, 2024, https://openai.com/index/hello-gpt-4o/

[O2] ODSC Team, *Top 10 Open-Source AI Agent Frameworks to Know in 2025*, https://opendatascience.com/top-10-open-source-ai-agent-frameworks-to-know-in-2025/

[O3] E. Ohiri and R. Poole, *What is the cost of training large language models?*, 2025,

[O4] OpenAI, *Introducing GPT-5.2*, 2025, https://openai.com/index/introducing-gpt-5-2/

[O5] OpenAI, *GPT-5.2 in ChatGPT*, https://help.openai.com/en/articles/11909943-gpt-52-in-chatgpt

[P1] J. Papa, *PyTorch Pocket Reference: Building and Deploying Deep Learning Models*, O'Reilly Media, 2021.

[P2] S. Poudel, *Recurrent Neural Network (RNN) Architecture Explained*, 2023, https://medium.com/@poudelsushmita878/recurrent-neural-network-rnn-architecture-explained-1d69560541ef

[P3] R. Poclitari, *How GenAI Transforms Software Development in 11 Ways*, 2024, https://www.index.dev/blog/11-generative-ai-use-cases-software-development

[P4] M. Patzak, *Generative AI Cost Optimization Strategies*. AWS Cloud Enterprise Strategy Blog, 2024, https://aws.amazon.com/blogs/enterprise-strategy/generative-ai-cost-optimization-strategies/

[P5] M. Petrelli, *Machine Learning Workflow*, in Machine Learning for Earth Sciences, Springer, 2023, https://link.springer.com/chapter/10.1007/978-3-031-35114-3_3

[P6] C. Pykes, *Machine Learning in Practice: ML Workflows*, 2023, https://developer.nvidia.com/blog/machine-learning-in-practice-ml-workflows/

[P7] A. Polak, *5 Agentic AI challenges, and how to overcome them*, 2025, https://interface.media/blog/2025/04/10/5-agentic-ai-challenges-and-how-to-overcome-them/

[P8] S. Papanaboina, *12 Failure Patterns of Agentic AI Systems—and How to Design Against Them*, https://www.concentrix.com/insights/blog/12-failure-patterns-of-agentic-ai-systems/

[P9] A. Patel, *A Study of Real-world AI Model Failures and Their Impact*, 2023, https://opendatascience.com/a-study-of-real-world-ai-model-failures-and-their-impact/

[P10] B. Padmaja et al., *Exploration of issues, challenges and latest developments in autonomous cars*, J Big Data, 10(61), 2023, https://doi.org/10.1186/s40537-023-00701-y

[P11] M. Pulse, *Challenges in Deploying Support Vector Machine Models: A Comprehensive Exploration*, 2024, https://medium.com/@jangdaehan1/challenges-in-deploying-support-vector-machine-models-a-comprehensive-exploration-576c53293687

[P12] *Potential Challenges in PCA*, Statistics Globe, https://statisticsglobe.com/wp-content/uploads/2024/03/Module-10-Potential-Challenges-in-PCA.pdf

[P13] M. Porter, *An algorithm for suffix stripping*, Program, 14(3), 1980, 130-137, https://dx.doi.org/10.1108/eb046814

[P14] T.Q. Pham et al., *Evaluation of the Hierarchical Correspondence between the Human Brain and Artificial Neural Networks: A Review*, Biology, 12(10), 2023, https://www.mdpi.com/2079-7737/12/10/1330

[P15] J. Pennington et al., *GloVe: Global Vectors for Word Representation*, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, https://aclanthology.org/D14-1162.pdf

[P16] H. Phillips, *A Simple Introduction to Tensors*, 2023, https://medium.com/@hunter-j-phillips/a-simple-introduction-to-tensors-c4a8321efffc

[P17] F. Parisi et al., *Continual Lifelong Learning with Neural Networks: A Review, Neural Networks*, 113, 2019, 54–71, https://arxiv.org/pdf/1802.07569

[P18] M. E. Peters et al., *Deep Contextualized Word Representations*, Proc. NAACL-HLT, 2018, 2227–2237, https://arxiv.org/pdf/1802.05365

[P19] *Prompting when using AI – how can companies get the most out of AI?*, 2024, https://ambersearch.de/en/what-is-prompting/

[Q1] J. Quiñonero-Candela et al., *Machine Learning Challenges*, Lecture Notes in Artificial Intelligence, Springer 2006.

[Q2] R. Qureshi et al., *Thinking Beyond Tokens: From Brain-Inspired Intelligence to Cognitive Foundations for Artificial General Intelligence and its Societal Impact*, 2025, https://arxiv.org/html/2507.00951v1

[R1] E. Rich, *Artificial Intelligence*, McGraw-Hill, 1983.

[R2] K. Rungta, *TensorFlow in 1 Day: Make your own Neural Network*, 2018.

[R3] S. Robinson et al., *What is a generative adversarial network (GAN)?*, 2024, https://www.techtarget.com/searchenterpriseai/definition/generative-adversarial-network-GAN

[R4] S. Roberts and R. Everson, *Independent Component Analysis*, Cambridge University Press, 2014.

[R5] *Restricted Boltzmann Machine*, 2023, https://www.geeksforgeeks.org/restricted-boltzmann-machine/

[R6] F. Rosenblatt, *The perceptron: A theory of statistical separability in cognitive systems*, Cornell Aeronautical Laboratory, Inc. Rep., 1958.

[R7] D.E. Rumelhart et al., *Learning representations by back-propagating errors*, 323, 1986, 533–536.

[R8] O. Ronneberger et al., *U-Net: Convolutional Networks for Biomedical Image Segmentation*, Medical Image Computing and Computer-Assisted Intervention (MICCAI), Springer, 9351, 2015, 234–241.

[R9] A. Radford et al., *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*, 2015, https://arxiv.org/abs/1511.06434

[R10] S.A. Rabbani et al., *Generative Artificial Intelligence in Healthcare: Applications, Implementation Challenges, and Future Directions*, BioMedInformatics, 5(3), 2025, 37. https://doi.org/10.3390/biomedinformatics5030037

[R11] G. Ribeiro, *Understanding The Limitations Of Generative AI*, 2024, https://www.forbes.com/councils/forbestechcouncil/2024/05/09/understanding-the-limitations-of-generative-ai/

[R12] J.Röst et al., *Why agentic AI projects fail: 10 Learnings and fixes (for those already past the co-pilot phase)*, https://algorithma.ai/our-latest-thinking/why-agentic-ai-projects-fail-10-learnings-and-fixes-for-those-already-past-the-co-pilot-phase

[R13] R. Rajna et al., *Large-scale Deep Unsupervised Learning using Graphics Processors*, Proceedings of the 26th International Conference on Machine Learning, Montreal, Canada, 2009, https://icml.cc/Conferences/2009/papers/218.pdf

[R14] J. Ren and D. Xia, *Challenges of Autonomous Driving Systems*, In: Autonomous driving algorithms and Its IC Design. Springer, Singapore, https://doi.org/10.1007/978-981-99-2897-2_1

[R15] E. Risko and S. Gilbert, *Cognitive offloading*, Trends in Cognitive Sciences, 20(9), 2016, 676–688. https://doi.org/10.1016/j.tics.2016.07.002

[R16] R. Ramos, *Using TF-IDF to Determine Word Relevance in Document Queries*, 2003, https://www.researchgate.net/publication/228818851_Using_TF-IDF_to_determine_word_relevance_in_document_queries

[R17] A. Radford et al., *Improving Language Understanding by Generative Pre-Training*, OpenAI Technical Report, 2018, https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[R18] A. Radford et al., *Language Models are Unsupervised Multitask Learners*, OpenAI Technical Report, 2019, https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[R19] C. Raffel et al., *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*, Journal of Machine Learning Research, 21, 2020, 1–67, https://jmlr.org/papers/volume21/20-074/20-074.pdf

[R20] A. Rush, *The Annotated Transformer*, Proceedings of Workshop for NLP Open Source Software, 2018, https://aclanthology.org/W18-2509.pdf

[R21] R. Roy, *10 Examples of AI Gone Wrong: Notorious AI Failures and What They Teach Us*, 2025, https://globalainews.tech/examples-of-ai-gone-wrong-shocking-ai-failures/

[R22] Rebellion Research, *Where will AI be in 5 to 10 years?*, 2025, https://www.rebellionresearch.com/where-will-ai-be-in-5-to-10-years

[S1] J. Smart, *Machine Learning Mathematics*, Jackson Smart, 2023.

[S2] C.E. Shannon, *A Mathematical Theory of Communication*, The Bell System Technical Journal, 27, 1948.

[S3] Sriram, *Multinomial Naive Bayes Explained: Function, Advantages & Disadvantages, Applications*, 2024, https://www.upgrad.com/blog/multinomial-naive-bayes-explained/

[S4] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, The MIT Press, 2020.

[S5] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2015, https://arxiv.org/pdf/1409.1556

[S6] Ch. Szegedy et al., *Going Deeper with Convolutions*, 2014,

[S7] H. Siegelmann and E. Sontag, *On the computational power of neural nets*, Journal of Computer and System Sciences, 50(1), 1995, 132–150.

[S8] M. Schuster and K. K. Paliwal, *Bidirectional recurrent neural networks*, in IEEE Transactions on Signal Processing, 45(11), 1997, 2673–2681.

[S9] F. Salem, *Recurrent Neural Networks: From Simple to Gated Architectures*, Springer, 2022.

[S10] D. Shulga, *What is the difference between Optimization and Deep Learning and why should you care*, 2019, https://medium.com/data-science/what-is-the-difference-between-optimization-and-deep-learning-and-why-should-you-care-e4dc7c2494fe

[S11] J. Sohl-Dickstein et al., *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*, 2025, https://arxiv.org/pdf/1503.03585

[S12] C. Staff, *20 Examples of Generative AI Applications Across Industries*, 2025, https://www.coursera.org/articles/generative-ai-applications

[S13] B. Shah et al., *Generative AI in the pharmaceutical industry: Moving from hype to reality*, McKinsey & Company, 2025, https://www.mckinsey.com/industries/life-sciences/our-insights/generative-ai-in-the-pharmaceutical-industry-moving-from-hype-to-reality

[S14] Ch. Stuber, *Perplexity AI – one of the best search engines of 2025*, 2025, https://ki-wandel.de/perplexity-ai/

[S15] S. Sai et al., *Generative AI for Transformative Healthcare: A Comprehensive Study of Emerging Models, Applications, Case Studies, and Limitations*, 2024, https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10440330

[S16] A. Shugarts, *The Hidden Costs of Running Generative AI Workloads – And How to Optimize Them*, 2025, https://rafay.co/ai-and-cloud-native-blog/hidden-costs-of-running-generative-ai-workloads/

[S17] A. Sidorkin, *Environmental Impact of Generative AI: Carbon and Water Footprint*. AI-EDU Arxiv, 2025, https://journals.calstate.edu/ai-edu/article/view/5448

[S18] C. Stryker, *What is multimodal AI?*, 2024, https://www.ibm.com/think/topics/multimodal-ai

[S19] A. Sukharevsky et al., *Seizing the agentic AI advantage*, 2025, https://www.mckinsey.com/capabilities/quantumblack/our-insights/seizing-the-agentic-ai-advantage

[S20] A. Singla et al., *The state of AI - How organizations are rewiring to capture value*, 2025, https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai

[S21] P. Sanghavi, *Top 10 ML Model Failures You Should Know About*, 2022, https://www.deepchecks.com/top-10-ml-model-failures-you-should-know-about/

[S22] V. Samarth, *Top 5 Real-World Applications of an Expert System in AI*, 2023, https://emeritus.org/in/learn/expert-system-in-ai/

[S23] H. Smolic, *A Practical Guide to Understanding and Implementing AI Decision Trees*, 2024, https://graphite-note.com/demystifying-ai-decision-trees-a-practical-guide-to-understanding-and-implementing/

[S24] M. Saeed, *A Gentle Introduction To Hessian Matrices*, 2022, https://machinelearningmastery.com/a-gentle-introduction-to-hessian-matrices/

[S25] A. Stéphane, *LIME vs SHAP: What's the Difference for Model Interpretability?*, 2025, https://apxml.com/posts/lime-vs-shap-difference-interpretability

[S26] J.R. Searle, *Minds, brains, and programs*, Behavioral and Brain Sciences, 3(3), 1980, 417–457. https://doi.org/10.1017/S0140525X00005756

[S27] E. Spitznagel, *AI Eroding Cognitive Skills in Doctors: How Bad Is It?*, Medscape, 2025, https://www.medscape.com/viewarticle/ai-eroding-cognitive-skills-doctors-how-bad-it-2025a1000q2k

[S28] Th. Sudduth, *Natural Language Processing with LLMs: From Foundations to Real-World Applications*, 2025.

[S29] F. Shao and Z. Shen, *How can artificial neural networks approximate the brain?*, Front. Psychol., 13, 2022, https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2022.970214/full

[S30] S. Schmidgall et al., *Brain-inspired learning in artificial neural networks: a review*, 2023, https://arxiv.org/abs/2305.11252

[S31] W. Schultz et al., *A Neural Substrate of Prediction and Reward*, Science, 275(14), 1997, https://www.gatsby.ucl.ac.uk/~dayan/papers/sdm97.pdf

[S32] *Spiking Neural Networks in Deep Learning*, 2025, https://www.geeksforgeeks.org/deep-learning/spiking-neural-networks-in-deep-learning-/

[S33] J. Su et al., *RoFormer: Enhanced Transformer with Rotary Position Embedding,* 2021, https://arxiv.org/pdf/2104.09864

[S34] N. Shazeer, *GLU Variants Improve Transformer*, 2020, https://arxiv.org/pdf/2002.05202

[S35] Y. Siddique, *Understanding Mixture of Experts in NLP: A Technical Deep Dive*, 2023, https://medium.com/@yasir_siddique/understanding-mixture-of-experts-in-nlp-a-technical-deep-dive-a02b5a73e025

[T1] O. Theobald, *Machine Learning for Absolute Beginners*, 2017.

[T2] A.M. Turing, *Computing machinery and intelligence*, Mind, 59, 1950, 433–460.

[T3] *Turing Test success marks milestone in computing history*, University of Reading, 2014. https://archive.reading.ac.uk/news-events/2014/June/pr583836.html

[T4] tableau.com, *What is the history of artificial intelligence (AI)?*, https://www.tableau.com/data-insights/ai/history#history

[T5] Toolify.ai, *Deep Learning Titans: Andrew Ng in Conversation with Geoffrey Hinton*, 2023, https://www.toolify.ai/ai-news/deep-learning-titans-andrew-ng-in-conversation-with-geoffrey-hinton-42874

[T6] N. C. Thompson et al., *Deep Learning's Diminishing Returns*, IEEE Spectrum, September 24, 2021.

[T7] N. Tomar, *What is UNET?*, https://medium.com/analytics-vidhya/what-is-unet-157314c87634

[T8] *The Turing Test: Definition, History, and Examples*, UMA Technology, December 31, 2024.

[T9] S-H. Tsang, *Review: LAPGAN — Laplacian Generative Adversarial Network (GAN)*, 2020, https://sh-tsang.medium.com/review-lapgan-laplacian-generative-adversarial-network-gan-e87200bbd827

[T10] J. Tefteller, *Prompt Engineering Guide*, 2025, https://www.justintefteller.com/articles/promptengineering

[T11] S. Tufail et al., *A.I. Advancements and Challenges in Machine Learning: A Comprehensive Review of Models, Libraries, Applications, and Algorithms*, Electronics 12, 2023, 1789, https://doi.org/10.3390/electronics12081789

[T12] *Top 12 Biggest Machine Learning Challenges and Solutions*, BigDataCentric, https://www.bigdatacentric.com/blog/machine-learning-challenges/

[T13] A. Toledo, *Microsoft turning to NUCLEAR ENERGY to power its future data centers and AI projects*, 2025, https://www.nuclear.news/2025-01-16-microsoft-nuclear-energy-data-centers-ai-projects.html

[T14] M.M. Taye, *Understanding of Machine Learning with Deep Learning: Architectures, Workflow, Applications and Future Directions*, Computers, 2023, 12(5), 2023, https://doi.org/10.3390/computers12050091

[T15] A. Tchagna, *The Drawbacks of K-Means Algorithm*, 2025, https://www.baeldung.com/cs/k-means-flaws-improvements

[T16] *The determinant of a matrix*, Math Insight, http://mathinsight.org/determinant_matrix

[T17] Z. Tian et al., *Meta-Learning Hyperparameters for Parameter Efficient Fine-Tuning*, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2025, 23037–23047, https://openaccess.thecvf.com/content/CVPR2025/papers/Tian_Meta-Learning_Hyperparameters_for_Parameter_Efficient_Fine-Tuning_CVPR_2025_paper.pdf

[T18] Y. Tay et al., *Efficient Transformers: A Survey*, 2022, https://arxiv.org/pdf/2009.06732

[T19] M. Tiezzi et al., *State-space modeling in long sequence processing: a survey on recurrence in the transformer era*, Neural Networks, 193, 2026, https://www.sciencedirect.com/science/article/pii/S0893608025009190

[U1] upGrade, *How Neural Networks Work: A Comprehensive Guide for 2025*, https://www.upgrad.com/blog/neural-network-tutorial-step-by-step-guide-for-beginners/

[U2] *Understanding the architecture of Recurrent Neural Networks (RNN)*, 2025, https://aiml.com/briefly-describe-the-architecture-of-a-recurrent-neural-network-rnn/

[U3] H. Umaletiya, *Exploring the Limitations of Generative AI: 5 Key Points*, https://www.brilworks.com/blog/limitations-of-generative-ai

[U4] *Using Generative AI in Research: Limitations & Warnings*, University of Southern California, 2025, https://libguides.usc.edu/generative-AI/limitations

[U5] S.R. Upadhyaya, *What is Perplexity AI? A Smarter Way to Search*, 2025, https://www.digitalocean.com/resources/articles/what-is-perplexity-ai

[V1] J. Varughese, *What are generative adversarial networks (GANs)?*, 2025, https://www.ibm.com/think/topics/generative-adversarial-networks

[V2] Vaswani et al., *Attention Is All You Need*, 2017, https://arxiv.org/abs/1706.03762

[V3] J. Venkatesan, *Agentic AI Raises Security Challenges in Chip Design*, 2025, https://www.electronicsforu.com/news/agentic-ai-raises-security-challenges-in-chip-design

[V4] N. Vecoven et al., *Introducing neuromodulation in deep neural networks to learn adaptive behaviour*, PLoS ONE, 15(1), 2020, https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0227922

[V5] G. M. van de Ven and A. S. Tolias, *Three Types of Continual Learning*, 2019, https://arxiv.org/pdf/1904.07734

[V6] N. Van Otten, *Embeddings from Language Models (ELMo): Contextual Embeddings A Powerful Shift In NLP*, 2023, https://spotintelligence.com/2023/12/26/embeddings-from-language-models-elmo/

[W1] Wikipedia, https://en.wikipedia.org/

[W2] A. Wolf, *Machine Learning Simplified*, Andrew Wolf, 2022.

[W3] Wikipedia, https://en.wikipedia.org/

[W4] L. Wiest, *Recurrent Neural Networks - Combination of RNN and CNN*, https://collab.dvb.bayern/spaces/TUMlfdv/pages/69119924/Recurrent+Neural+Networks+-+Combination+of+RNN+and+CNN#RecurrentNeuralNetworksCombinationofRNNandCNN-Thebasicidea

[W5] D.H. Wolpert and W.G. Macready, *No Free Lunch Theorems for Optimization*, IEEE Transactions on Evolutionary Computation, 1(1), 1997, doi: 10.1109/4235.585893

[W6] P. West et al., *The Generative AI Paradox: "What It Can Create, It May Not Understand"*, 2023, https://arxiv.org/abs/2311.00059

[W7] *What are the Limitations of AI in Robotics*, https://www.flyrank.com/blogs/ai-insights/what-are-the-limitations-of-ai-in-robotics

[W8] D. Wilimitis, *The Kernel Trick in Support Vector Classification*, 2018, https://medium.com/data-science/the-kernel-trick-c98cdbcaeb3f

[W9] *What is Tokenization in Natural Language Processing (NLP)?*, 2025, https://www.geeksforgeeks.org/nlp/tokenization-in-natural-language-processing-nlp/

[W10] *Word Embeddings in NLP*, 2025, https://www.geeksforgeeks.org/nlp/word-embeddings-in-nlp/

[W11] *When AI goes wrong: 13 examples of AI mistakes and failures*, 2024, https://www.evidentlyai.com/blog/ai-failures-examples

[W12] R. Wile and L. Kolodny, *Federal regulator finds Tesla Autopilot has 'critical safety gap' linked to hundreds of collisions*, 2024, https://www.nbcnews.com/tech/tech-news/feds-say-tesla-autopilot-linked-hundreds-collisions-critical-safety-ga-rcna149512

[W13] J. Wei et al., *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*, 2023, https://arxiv.org/pdf/2201.11903

[W14] E. Woollacott, *A new study claims AI will destroy 10.4 million roles in the US by 2030, more than the number of jobs lost in the Great Recession – but analysts still insist there won't be a 'jobs apocalypse'*, 2026, https://www.itpro.com/business/business-strategy/ai-job-losses-great-recession-us-forrester

[W15] T. Williams, *Experts warn of AI dangers in major new report*, 2025, https://ia.acs.org.au/article/2025/experts-warn-of-ai-dangers-in-major-new-report.html

[W16] D. Wetzel, *Bundesnetzagentur: Die erstaunliche Idee des Energiewende-Prangers*, Die WELT, Jan. 26, 2026.

[W17] R. Worden, AI and World Models, 2026, https://www.arxiv.org/pdf/2601.17796

[W18] J. Wei et al., *Emergent Abilities of Large Language Models*, 2022, https://arxiv.org/pdf/2206.07682

[X1] Q Xie et al., *Self-training with Noisy Student improves ImageNet classification*, 2020, https://arxiv.org/abs/1911.04252v4

[X2] B. Xu et al., Curriculum Learning for Natural Language Understanding, 2020, https://aclanthology.org/2020.acl-main.542.pdf

[Y1] R. Yassminh, *Covariance and Correlation in Machine Learning: Practical Applications and Insights*, 2024, https://medium.com/@ryassminh/covariance-and-correlation-in-machine-learning-practical-applications-and-insights-1dbe8cee1e1a

[Y2] P. Yadav, *The journey of Gradient Descent — From Local to Global*, 2021, https://medium.com/analytics-vidhya/journey-of-gradient-descent-from-local-to-global-c851eba3d367

[Y3] L. Yang et al., *Diffusion Models: A Comprehensive Survey of Methods and Applications*, 2024, https://arxiv.org/html/2209.00796v14

[Y4] Q. Yi et al., *Diffusion models in text generation: a survey*, PeerJ Comput. Sci., 2024, https://peerj.com/articles/cs-1905/

[Z1] Z-H. Zhou, *Machine Learning*, Springer, 2021.

[Z2] H. Zhao et al., *Feature Learning and Understanding*, Springer, 2020.

[Z3] A. Zhang et al., *Dive into Deep Learning*, Cambridge University Press, 2023.

[Z4] M. Zulqarnain et al., *An Improved Deep Learning Approach based on Variant Two-State Gated Recurrent Unit and Word Embeddings for Sentiment Classification*, International Journal of Advanced Computer Science and Applications, 11(1), 2020, 594–603.

[Z5] K. Zweig, *Awkward Intelligence. Where AI Goes Wrong, Why It Matters, and What We Can Do about It*, 2022, The MIT Press.

[Z6] A. Zouaoui, *StyleGAN: Explained*, 2022, https://medium.com/@arijzouaoui/stylegan-explained-3297b4bb813a

[Z7] J-Y. Zhu et al., *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*, 2017, https://arxiv.org/abs/1703.10593

[Z8] A. Zewe, *Despite its impressive output, generative AI doesn't have a coherent understanding of the world*, MIT, 2024, https://news.mit.edu/2024/generative-ai-lacks-coherent-world-understanding-1105

[Z9] A. Zewe, *Explained: Generative AI's environmental impact*, MIT News, 2025, https://news.mit.edu/2025/explained-generative-ai-environmental-impact-0117

[Z10] S. Zhang, *Challenges in KNN Classification*, IEEE Transaction Knowledge and Date Engineering, 34(10), 2022, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9314060

[Z11] R-J. Zhu et al., *A Survey on Latent Reasoning*, 2025,