Teaching Sand to Dream

Artificial Intelligence Explained



Dr. Jesse Wood

Teaching Sand to Dream

Artificial Intelligence Explained

Dr. Jesse Wood

Teaching Sand to Dream: Artificial Intelligence Explained

Copyright © 2026 Jesse Wood. All rights reserved.

No part of this publication may be reproduced or transmitted in any form without permission in writing from the author.

First draft edition. May 2026.

Who This Book Is For

This book is for anyone who has found themselves curious about artificial intelligence but unsure where to start — or who has started and found most explanations either too shallow or too technical. It assumes no mathematics, no programming experience, and no prior knowledge of the field beyond what you have absorbed from using the internet.

It does assume you are willing to think carefully, follow an argument across a few pages, and sit with questions that do not have clean answers. Artificial intelligence is a field that has been arguing with itself for nearly a century about what it is even trying to do. This book does not paper over those arguments. It treats them as the most interesting part of the story.

If you finish this book and want to go further, the Further Reading section at the back points toward the technical literature where the real depth lives. That next step is harder. This book is the preparation for it.

Contents

Introduction **1**

I The Dream of Thinking Machines **5**

1 What Is Intelligence, Anyway? **7**
1.1 Why a single definition keeps failing 8
1.2 The constellation 9
1.3 The shapes intelligence takes 11
1.4 The serious definitions — and their tensions 12

2 The Pioneers and Their Impossible Dream **17**
2.1 The prophet who didn't see it (Alan Turing) 18
2.2 The biological logic gates (McCulloch & Pitts) . . 20
2.3 The top-down optimists (McCarthy & Minsky) . 21
2.4 The bottom-up rebel (Frank Rosenblatt) 23
2.5 Keepers of the flame (Rumelhart & Hinton) 25

II Learning to Learn **29**

3 Rules vs. Patterns **31**
3.1 The illusion of logic (Top-Down AI) 33
3.2 The edge case explosion 34
3.3 The pattern recognition paradigm (Bottom-Up AI) 36
3.4 The second winter 39
3.5 The bridge to the machine that learns 40

4 The Neuron That Started Everything **43**
4.1 The committee in your head . . . 43
4.2 Learning is just adjusting the weights . . . 45
4.3 Why one neuron isn't enough . . . 47
4.4 What a layer actually does . . . 49
4.5 Why this is harder than it sounds . . . 50

III Machines That See **53**

5 Teaching Machines to See **55**
5.1 The pixel grid . . . 55
5.2 The same object, a thousand different pictures . . . 56
5.3 Why the rule-based approach failed . . . 58
5.4 The shape of the problem the field couldn't solve 59

6 Convolutional Networks and the Art of Noticing **63**
6.1 The trick: looking at patches, not pixels . . . 63
6.2 Stacking the patches: depth in convolutional networks . . . 65
6.3 ImageNet and the 2012 moment . . . 67
6.4 What convolutional networks see (and what they miss) . . . 69

IV Machines That Understand **71**

7 The Meaning of Words **73**
7.1 Words don't sit on a number line . . . 73
7.2 Meaning as position . . . 75
7.3 The geometry of meaning . . . 76
7.4 The "bank" problem . . . 76

8 Attention Is All You Need **79**
8.1 The problem attention solves . . . 79
8.2 Paying attention . . . 81
8.3 Attention is all you need . . . 81
8.4 Why this changed everything . . . 82

9 The Large Language Model **85**
9.1 Words, actually tokens . . . 85
9.2 Predicting the next token . . . 87

9.3 Scale changes everything . . . 88
9.4 Emergence . . . 88
9.5 ChatGPT, Claude, and the present . . . 90

V The Dream, Revisited **93**

10 What Machines Still Can't Do **95**
10.1 The jagged frontier . . . 95
10.2 Common sense and the world the words came from . . . 97
10.3 Coherence over time . . . 98
10.4 The systems we cannot read . . . 100
10.5 What the gaps tell us . . . 102

11 Sand, Silicon, and What Comes Next **105**
11.1 Sand, Silicon, Dreams . . . 107
11.2 Being versus Tool . . . 107
11.3 AI Consciousness . . . 108
11.4 Teaching Sand to Dream . . . 110

Glossary **113**

Bibliography **119**

Further Reading **129**

Introduction — Why Sand?

There are approximately 7.5 quintillion grains of sand on earth. In 1959, we started putting that sand to good use, when Robert Noyce demonstrated the first practical planar silicon integrated circuit. Since then computers have become ubiquitous in human civilisation, giving rise to technological marvels like the internet, and what seems more recent, the rise of artificial intelligence. Artificial intelligence started as the pipe dream of the thinking machine from visionaries like Alan Turing. One day we turned sand into machines, then slowly but surely, we taught those machines to dream. First of all, what is AI? It is an acronym for artificial intelligence. Although there are many definitions, the following will be sufficient for the purposes of this book. Arthur Samuel, one of AI's earliest pioneers, gave what became a foundational definition of machine learning: computers learning without being explicitly programmed [1]. We can think of a program as similar to a recipe for a cake, but instead as a set of instructions that tell the computer how to complete a task. Artificial intelligence gives computers the ability to complete tasks without being told exactly what to do. For example, we could teach a computer how to play chess [2] without explicitly programming a response to every possible move on the board.

Now given the prevalence of generative AI tools like ChatGPT it is common to think of artificial intelligence as *just* ChatGPT. But as you will find out in this book, there is a long lineage of research and technology spanning nearly a century that constitutes artificial intelligence. Also, as a matter of technicality, ChatGPT is a form of generative AI, which is a subfield of deep learning, which is a subfield of machine learning, which is a sub-

field of artificial intelligence — all terms that will be introduced in good time throughout this book.

Could a machine ever think? What started as science fiction and thought experiment has become concrete with today's advances in AI. Machines have beaten us at chess [2], at Go [3], and at Dota 2 and StarCraft [4, 5]. AI systems have achieved landmark results in protein-structure prediction [6], matched or exceeded expert performance on specific medical-imaging benchmarks such as diabetic retinopathy detection [7], and helped researchers recover text from ancient charred scrolls buried by Vesuvius [8]. And yet most people encounter AI only through a chatbox, unaware of the decades of struggle, ingenuity, and occasional madness that made it possible. This book is an attempt to change that — to pull back the curtain and show you not just what AI can do, but how it came to be able to do it at all.

We'll start where AI started — with the question of what intelligence even is, and the handful of mid-century dreamers who believed machines might one day possess it. From there we'll follow the long, sometimes frustrating road from simple rules to pattern-learning, from the first artificial neuron to the systems that today can write, produce reasoning-like outputs, and surprise even their creators. Along the way you'll meet the ideas that changed everything — the perceptron, the convolutional network, the transformer — explained not with equations, but with the intuitions behind them.

By the last page, you'll understand how we got here — and why the story of artificial intelligence is, at its heart, the story of teaching sand to dream.



Figure 0.1. From grains of sand to silicon wafers to language models: the unlikely material journey behind modern artificial intelligence. Ordinary silicon dioxide is refined into pure silicon, etched into circuits, and eventually becomes the physical substrate on which machines can learn, generate, and "dream."

Part I

The Dream of Thinking Machines

CHAPTER 1

What Is Intelligence, Anyway?

Before we can ask whether machines are intelligent, we need to interrogate the word itself. This chapter provides a playful tour through what intelligence means in humans, animals, and systems — planting seeds the rest of the book will return to.

Maybe we can come to an understanding of intelligence by looking at the things in life we have come to call intelligent or smart. We call chess grandmasters intelligent. We call octopuses opening jars intelligent. We call immune systems and thermostats intelligent, sometimes seriously, sometimes loosely. The word stretches to cover all of it and tears under the weight.

Consider games of strategy. For most of the twentieth century, chess and Go were treated as paradigm cases of intelligence — we looked up to grandmasters as geniuses, and the ability to play these games well was taken as evidence of something deep. Then, in the 1997 rematch, IBM's Deep Blue beat world chess champion Garry Kasparov [2]. We did not concede that computers were intelligent. Instead, we moved the goalposts. Critics argued that Deep Blue relied on brute-force search rather than genuine understanding — a charge we will meet again, in more sophisticated form, when we reach Searle's Chinese Room later in this chapter. This pattern has a name: the *AI Effect* [9]. As soon as AI becomes able to complete a task, the task is no longer considered a real example of intelligence. Optical character recognition, speech recognition, machine translation — once

foundational problems in artificial intelligence, now quietly reclassified as engineered solutions. The boundary of intelligence keeps retreating just ahead of whatever machines can currently do.

Two and a half millennia of trying — from Aristotle's *De Anima* to contemporary cognitive science — and there is still no agreed definition. The question "can machines be intelligent?" inherits all of that unresolved baggage. The chapter's honest stance: we will not settle this. But we can map the disagreement, and the map turns out to be most of what you need to read the rest of the book.

1.1 Why a single definition keeps failing

A pocket calculator, if you are old enough to remember those, can beat every human at arithmetic. A two-year-old beats every computer at picking up an unfamiliar object on a cluttered floor. A teenager learns to drive in twenty hours of practice; an autonomous driving agent needs hundreds of thousands [10]. If intelligence were one thing on a single scale, these comparisons would not be coherent — yet most people feel the toddler is "more intelligent" than the calculator, and we would hope, despite the evidence of their behaviour, that the teenager is more intelligent than both.

The frontier of machine intelligence is *jagged*: capabilities advance unevenly, with strange peaks and gaps that often surprise even the developers of the systems themselves [11]. Modern large language models can produce competent legal analysis on Monday and fail at a primary-school arithmetic puzzle on Tuesday. This is not what we would expect from a single underlying quantity. It is what we would expect from a bundle of distinct capabilities that happen to share a name.

This is the move Gardner made for humans. His theory of multiple intelligences [12] proposed at least seven kinds — linguistic, logical-mathematical, spatial, musical, bodily-kinesthetic, interpersonal, and intrapersonal — on the grounds that no single measure captures the variety. The theory is controversial as psychometrics, generative as a frame. The older and statistically more orthodox tradition is Spearman's [13], which used factor analysis on cognitive test scores to identify g, a general intelli-

gence factor that accounts for the bulk of variance across tasks. The strength of Spearman's case is that performance across very different cognitive tests really does correlate reliably in humans. The limit of it, for our purposes, is that even if g exists, it is a statistical regularity in one species' cognitive architecture — not a definition of what intelligence *is*. Machines may realise some points of the constellation and miss others entirely, in patterns the g-factor framing was never designed to describe.

1.2 The constellation

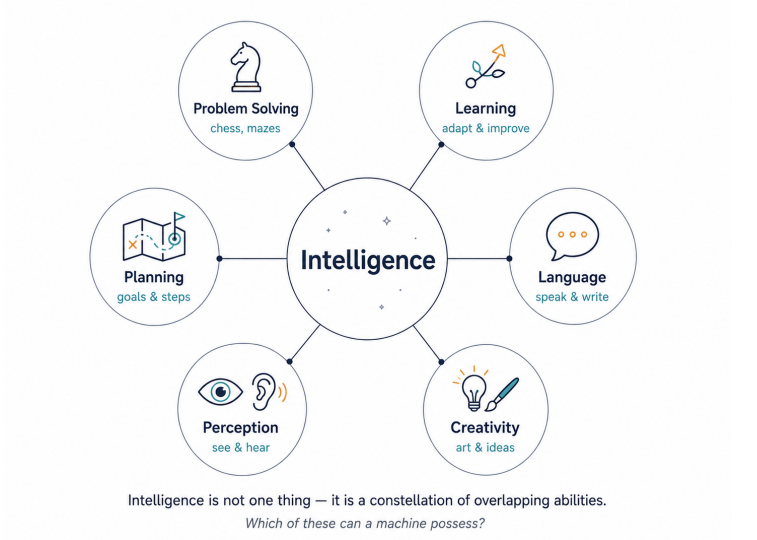


Figure 1.1. Intelligence is not a single ability but a constellation of overlapping capacities: problem solving, learning, language, creativity, perception, and planning. The question of machine intelligence is therefore not whether a system possesses intelligence in some total sense, but which parts of this constellation it can realise.

The diagram above is also, conveniently, a table of contents. Each point of the constellation has yielded to silicon in a different way, on a different timeline, with a different cast of characters, and the rest of this book is the story of how. Treat what follows as a map of where we are heading.

Problem solving is where AI's earliest victories were won. Computers learned to play games of strategy long before they learned anything else, because games of strategy reduce to search: enumerate the possible moves, evaluate the positions they lead to, pick the best one. Chapter 2 follows the pioneers who first imagined this — Turing, Shannon, McCarthy — and Chapter 3 traces the moment the field discovered that hand-written rules would only get them so far.

Learning is what made the modern field possible. The breakthrough came in 1959, when Arthur Samuel wrote a checkers program that did not memorise opening books or follow grandmaster heuristics, but improved by playing itself thousands of times and noticing which positions tended to lead to wins [1]. By 1962 it could beat a human champion. Chapter 3 stays with this idea — the shift from rules to patterns — and Chapter 4 introduces the single component that made pattern-learning practical at scale: the artificial neuron.

Perception stumped the field for forty years. It is easy to write down what a cat looks like; it is hard to write down what a cat looks like in terms a computer can use, because a cat is a different grid of numbers in every photograph. Part III — Chapters 5 and 6 — is the story of how convolutional networks finally cracked the problem, with the 2012 ImageNet moment serving as AI's equivalent of the moon landing.

Language is the constellation point we are still inside. Words mean different things in different contexts; meaning depends on what surrounds it, and on the world the words are about. Part IV — Chapters 7, 8, and 9 — traces how the field went from treating words as discrete symbols to treating them as positions in geometric space, and from there to the transformer architecture that powers every large language model you have ever heard of.

Creativity and planning are the points where the chapter ends and the questions begin. Machines now produce novel chess moves and novel sentences and novel images; whether any of this counts as creativity in the sense humans mean depends on which side of the Turing–Searle debate you fall on. Planning — multi-step reasoning over time — is the area where current systems remain visibly patchy. Chapter 10 is honest about both, and Chapter 11 returns to the spiral the book opened with: where this is all going, and what questions we should be asking

on the way.

Six constellation points. Eleven chapters. The question *is the system intelligent?* is going to keep dissolving, throughout the book, into the more useful question: *which points has it realised, and how?*

1.3 The shapes intelligence takes

The diagram at the start of this chapter asked whether intelligence requires a brain, a body, a self. Biology's answers are surprising. Take the slime mould *Physarum polycephalum*, a single-celled organism with no nervous system at all. Drop it into a maze with food at two corners and it will, within hours, retract its body from every dead end and form a single yellow strand tracing the shortest path between the two food sources [14]. A decade later, researchers laid out oat flakes on a map of Tokyo at the positions of the city's major railway stations. The slime mould grew a network of nutrient channels between them whose topology closely matched the actual Tokyo rail system — a transport network refined over a century of human engineering, approximated overnight by an organism without a thought in its head [15]. A brain is sufficient for problem solving. It is plainly not necessary.

The octopus tries a different solution. Two-thirds of its neurons are not in its central brain but distributed along its eight arms [16]. Each arm can solve problems semi-independently — unscrewing a jar, navigating a crevice, deciding what to taste — while the central brain attends to something else entirely. If you wanted a working example of a cognitive architecture radically unlike the human one, you would not have to leave the ocean.

And what about the self? New Caledonian crows perform spontaneous metatool use, fashioning a short stick to extract a longer stick to extract food [17]. Honeybees communicate distance and direction through the waggle dance, with fewer than a million neurons between them [18]. Are these creatures self-aware? Honestly, we do not fully know, and the question may not have a clean answer. The intuition that real intelligence requires a someone-home is persistent. The intuition does not survive contact with a honeybee.

If intelligence on Earth takes this many shapes within carbon-

based biology — distributed across cells without brains, across bodies without central nervous systems, across colonies and arms and insect brains the size of a sesame seed — then expecting machine intelligence to look like ours is the strangest assumption we could make. The constellation framing predicts machine intelligence will hit some points and miss others, in patterns we may not even have words for yet.

1.4 The serious definitions — and their tensions

In 1950, Alan Turing dodged the question of definitions and replaced "can machines think?" with what he called the imitation game [19]. Imagine you are seated at a teletype terminal. You can type questions and read the answers that come back. On the other end of the wire are two correspondents you cannot see or hear — one is a person, the other is a machine — and your task is to work out which is which by conversation alone. You can ask anything. The weather, the meaning of a poem, what they had for breakfast, what they think of the question itself. If, after a long enough exchange, you cannot reliably tell the machine from the human, Turing argues, the question of whether the machine "really" thinks has become meaningless. The text-only constraint is doing real work here: by stripping out voice, appearance, and every other cue, Turing isolates thinking from sounding human. He defines intelligence by what a thing does, not by what it is — an instance of what philosophers call operationalism. Take the move seriously, because without it the field of AI cannot exist. Every result becomes vulnerable to the same complaint: Deep Blue beats Kasparov, but is it really thinking? AlphaGo plays move 37, but is it really thinking? Turing's bet is that behaviour, given enough of it, is enough.

Thirty years later, Searle bet the opposite. His 1980 thought experiment, the Chinese Room [20], is designed to make Turing's position feel obviously wrong. Imagine you are locked in a room. You speak English and have no knowledge of Chinese. Inside the room is a vast rulebook in English: when a piece of paper with these particular squiggles arrives, find the matching entry, copy out these other squiggles, and pass that paper back out. A Chinese speaker outside the room is sliding questions under the door and reading your responses. The rulebook is so

good that the answers coming out are indistinguishable from those of a fluent native speaker. They ask "What is your favourite season?" in Chinese and receive, in elegant Chinese, "Autumn, because I love the way the light changes." From outside, it is completely convincing. The room passes the Turing test in Chinese. Now Searle asks: do you understand Chinese? Obviously not. You don't know what any of the symbols mean. You could have just discussed poetry or planned a bank robbery — there is no understanding happening anywhere inside you. The Chinese Room, Searle argues, is structurally identical to a computer running a program: symbolic inputs, formal rules, symbolic outputs. Formal symbol manipulation, however sophisticated, however successful at producing intelligent-seeming behaviour, does not give rise to understanding or meaning. Syntax is not semantics.

Turing and Searle gave us thought experiments. Shane Legg and Marcus Hutter gave us a formula. By the mid-2000s the philosophy-of-mind debate had been running for fifty years without resolution, and the working field of AI had its own problem: no agreed definition of its own subject matter. Chess players, theorem provers, image classifiers, robotic controllers — each subfield had its own benchmarks and its own implicit notion of what "intelligence" meant. The Turing test was philosophically charged and empirically vague: how long an exchange, which judges, which topics? Legg, who was then Hutter's PhD student and would later go on to co-found DeepMind, wanted something different. Not a philosophical position. A measurement framework. Their 2007 paper proposed one in a single sentence: intelligence measures an agent's ability to achieve goals across a wide range of environments [21]. Each piece carries weight. An agent is anything that takes an action, receives a response from its surroundings, and acts again — a thermostat, a chess engine, a toddler, a self-driving car. A goal is whatever determines success: winning the game, holding the room at 22°C, getting the cookie off the counter. And the range of environments is the part that does the heaviest lifting. A calculator achieves its goal in exactly one environment: arithmetic. A toddler achieves goals in thousands — language, physics, social manipulation, unfamiliar living rooms before lunch. The Legg–Hutter framework averages performance across all of them, weighted so that simple, structured worlds count more than

arbitrary noise. The critique writes itself: the definition presupposes goals, which is precisely the murky part. Where do the goals come from? Humans don't just achieve goals; they decide which ones are worth having, and that capacity sits upstream of anything Legg and Hutter measure. But the move itself is what matters for our purposes. Where Turing said *test it*, and Searle said *no test will do*, Legg and Hutter said something stranger: stop arguing about what intelligence is, and write down a number.

Forty years after Searle, the Chinese Room came back — no longer a thought experiment about a hypothetical person following a rulebook, but an empirical claim about systems you were using last week. A language model sees only text during training: strings of tokens, never the world the tokens refer to, never the intent behind them. It learns which tokens follow which others with extraordinary statistical richness. But form alone, the argument goes, cannot reconstruct meaning, because meaning is *about* something, and aboutness is exactly what raw form leaves out. The model could be producing a love poem or instructions for a heist; nothing inside it has any way to know. In 2021, Emily Bender, Timnit Gebru, Angelina McMillan-Major, and Margaret Mitchell published "On the Dangers of Stochastic Parrots" [22], and gave the position its lasting name. A stochastic parrot is a system that haphazardly stitches together sequences of linguistic forms it has observed in training, according to probabilistic information about how they combine, without any reference to meaning. The catchphrase has since escaped the paper and become shorthand for the whole position. Searle's rulebook was hand-written by a programmer; the transformer's "rulebook" is learned from data. The substrate changed; the philosophical move did not. Symbolic surface in, symbolic surface out, no understanding required. What makes the paper hard to dismiss as armchair philosophy is that it is Searle's argument made by working NLP researchers, about systems that actually exist. We have built machines that pass the Turing test by giving up on the question the Turing test was asking.

Most working AI researchers ignore the Turing–Searle debate entirely. They build systems, run benchmarks, and let philosophers worry about whether what they have built "really" understands anything. This is not necessarily a vice — fields make progress by deferring foundational questions — but in 2026 the question has stopped being deferrable, because the two

most prominent voices in the live debate disagree completely. Geoffrey Hinton, Turing Award winner and often called the godfather of deep learning, argues that next-token prediction, done well, requires understanding. The example he uses in interviews: imagine a detective novel that ends with the sentence "And the killer is __." To predict that word correctly, a model has to have followed the plot, weighed the evidence, tracked the suspects, grasped the motives. If a system can reliably fill in that blank, Hinton argues, then either it understands the novel or "understanding" is doing no real work as a concept. Gary Marcus, cognitive scientist and longtime critic of pure deep learning, argues the opposite: prediction can look like understanding without being understanding. A model can produce the right name through statistical association without grasping the structure that makes the answer correct in any deeper sense, and current systems reliably fail at exactly the kind of consistent multi-step reasoning genuine understanding would produce — they hallucinate citations, contradict themselves across long contexts, and break down on problems that vary slightly from their training distribution. The disagreement is not really about language models. It is the Turing–Searle debate happening again, in 2026, with concrete systems in place of thought experiments. The philosophical fog has not lifted; it has only moved into the present tense.

* * *

We have not defined intelligence. We have, I hope, shown why it resists definition — why two and a half millennia of trying have produced a constellation rather than a point, why biology realises that constellation in shapes that surprise us, and why the philosophical positions on what it all means remain in genuine, productive tension. The rest of this book asks a narrower and more answerable question: which slices of this constellation have we managed to put into silicon, and how did we manage it? The diagram earlier in this chapter ended with a question — *which of these can a machine possess?* The chapters ahead are the long answer.

CHAPTER 2

The Pioneers and Their Impossible Dream

The story of artificial intelligence is usually told as a story of ideas: 1936, universal computation; 1943, the artificial neuron; 1956, the Dartmouth conference; 1986, backpropagation. The dates form a tidy progression. The papers stack into a citation chain. The field looks, in retrospect, like a relay race in which each runner handed a baton cleanly to the next.

That is not how it happened.

The people who built the intellectual foundations of AI were doing it in the dark, against the limits of mid-century computing hardware, often against the indifference of their own academic peers, and sometimes against each other. Several of them did not live long enough to see whether their ideas were right. One died at forty-one. Another died at forty-three. Two died in the same year, having barely spoken to each other for two decades despite once being intellectual partners closer than most marriages. The longest-lived of them watched the field they helped found go through two complete cycles of triumph and collapse, and outlived nearly everyone they had argued with.

This chapter is about those lives. The reason it is organised around people rather than papers is that the timing of their lifespans is the chapter's argument: the theoretical work of artificial intelligence was largely complete by the late 1980s, decades before the hardware existed to test whether it would work. The

dream of thinking machines was paid for, in significant part, by people who never got to see the answer.

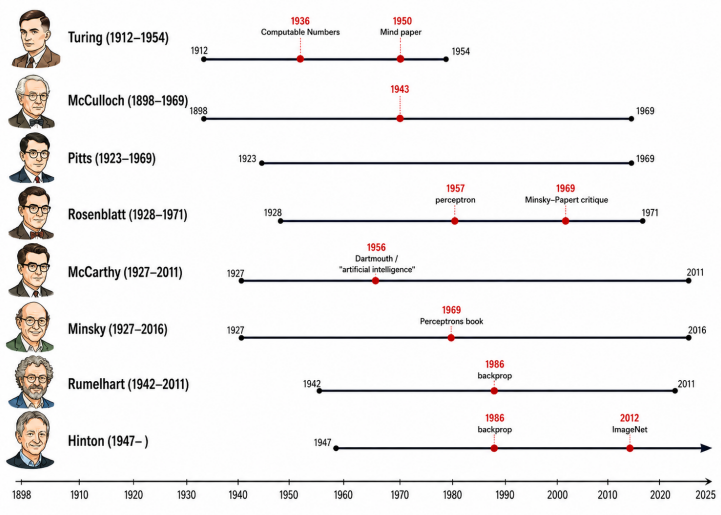


Figure 2.1. The human timeline of early AI. Some pioneers died before their ideas could mature; others lived long enough to see the field pass through optimism, winter, and revival.

2.1 The prophet who didn't see it (Alan Turing)

Alan Turing did not build the first artificial intelligence, but he built the intellectual runway for everyone who followed. In 1936, at the age of twenty-four, Turing published a paper that fundamentally altered our relationship with machines: "On Computable Numbers, with an Application to the Entscheidungsproblem" [23]. The title is forbidding; the idea inside it is one of the most consequential in the history of mathematics.

Before Turing, a machine did one thing. A loom wove cloth. A calculator crunched numbers. A telegraph sent messages. Each machine was built for its task and could not do anything else. Turing imagined a single, theoretical machine — an abstract device we now call a Turing machine — that could simulate any logical process, provided it was given the right instructions. The hardware no longer mattered. What mattered was the program.

By separating the physical machine from the abstract task it performed, Turing made it possible, decades before the first programmable computer was built, to ask whether any task at all could be reduced to a sequence of mechanical steps. Including, eventually, the task of thinking.

The Second World War turned this abstraction into hardware. At Bletchley Park, Turing led the team that broke the German Enigma cipher, designing electromechanical machines called Bombes that systematically searched through the daily key settings of the Wehrmacht's communications. The Bombes were not general-purpose computers — they were dedicated code-breaking engines — but they demonstrated something that the academic philosophers of mind had no way to see: that a machine could do work which everyone had previously assumed required a human mind. By 1944, Turing's machines were reading German naval communications faster than the Germans could send them. The intellectual gap between "computing a number" and "understanding a sentence" suddenly looked smaller than anyone had supposed.

In 1950, Turing took the separation to its logical extreme. "Computing Machinery and Intelligence" [19], published in the philosophy journal *Mind*, asked whether machines could think. We met this paper in the previous chapter; what is worth saying here is what it cost him to write. Turing was an English mathematician working in a discipline that had no name and a community that did not exist. The paper proposed not only the imitation game but also nine objections to machine intelligence and his replies to each. The replies are long and detailed and occasionally annoyed. Turing was arguing into silence. There was no field yet for him to argue with.

He gave the field its north star: bypass the philosophical quagmire of "what is consciousness" and replace it with a testable behavioural game. But look at his timeline. The line ends with a hard stop in 1954. In 1952, Turing had been convicted of "gross indecency" for a homosexual relationship and forced to undergo chemical castration as an alternative to prison. He died in 1954, at the age of forty-one, of cyanide poisoning. The coroner ruled suicide; the historical record remains contested. Whichever it was, the result is the same: Turing lived to see the first stored-program computers begin to operate — he wrote the first programmer's manual for the Manchester Mark I — but he died

before artificial intelligence became an organised research field, eight years before the first machine to beat a human at a board game, and forty-three years before Deep Blue beat Kasparov.

He was a prophet who never got to see the promised land.

2.2 The biological logic gates (McCulloch & Pitts)

If Turing provided the theoretical machine, Warren McCulloch and Walter Pitts looked for the software in a strange place: human biology. In 1943, this unlikely duo published "A Logical Calculus of the Ideas Immanent in Nervous Activity" [24], a paper that proposed something audacious. The brain, they argued, could be understood as a network of logic gates. Each neuron was a binary switch: either it fired, or it did not. Wire enough of these switches together in the right configuration, and you could compute anything that was computable.

Their partnership is one of the strangest stories in twentieth-century science. McCulloch was an established neurophysiologist at the University of Illinois, a philosopher by inclination and a poet in his spare time, who had spent two decades trying to understand how nervous systems gave rise to thought. Pitts was a fifteen-year-old runaway from Detroit who had taught himself logic by reading Bertrand Russell's *Principia Mathematica* in a public library, then written to Russell pointing out errors in the proofs. Russell, astonished, eventually arranged for him to study informally at the University of Chicago. Pitts slept in libraries, had no formal degree, and lived on whatever food his friends could spare. He was perhaps the most gifted mathematical logician of his generation, and he never finished high school.

The two met in 1942, when McCulloch took Pitts into his home and into his research. Within a year they had written the 1943 paper. Its central claim was startling and precise: any function that can be computed by a Turing machine can also be computed by a network of artificial neurons, given enough of them and the right connections. The brain, they argued, was not mysterious in principle. It was a logical machine of a particularly intricate kind, and once you understood the principle, you could in theory build one.

This was the moment the connectionist programme in arti-

ficial intelligence was born. Every neural network in this book — every convolutional layer, every transformer attention head, every parameter in a large language model — descends from the McCulloch–Pitts neuron. Their 1943 paper is the genealogical root of the entire bottom-up tradition in AI.

It did not end well between them. In 1952, Pitts learned of a personal scandal involving Norbert Wiener, the MIT cybernetician who had been his mentor and intellectual father after McCulloch. The details are murky and disputed, but the result is clear: Wiener cut off contact with Pitts permanently, and the cybernetics community fractured. Pitts, who had structured his entire intellectual life around these mentorships, fell into severe depression and alcoholism. He burnt his unpublished work, refused to publish further papers, and effectively withdrew from science by his mid-thirties. He had been the most promising mathematical mind of his cohort. He published almost nothing for the last fifteen years of his life.

Despite being born twenty-five years apart, McCulloch and Pitts both died in 1969. McCulloch was seventy; Pitts was forty-six, in poor health from years of drinking, living alone in a Cambridge boarding house. They had not collaborated in seventeen years. They died within four months of each other — and they died in the same year that Minsky and Papert published the book that would freeze the field of artificial neurons for the next two decades. The intellectual marriage that had birthed connectionism ended in simultaneous tragedy, just as the paradigm they had founded was being publicly entombed.

2.3 The top-down optimists (McCarthy & Minsky)

If McCulloch and Pitts wrote the foundational paper of bottom-up artificial intelligence, John McCarthy and Marvin Minsky founded the field that ignored it.

In the summer of 1955, McCarthy — then a young assistant professor at Dartmouth College — wrote a funding proposal to the Rockefeller Foundation [25]. He had co-authored it with three of the most respected names in early computing: Marvin Minsky, then at Harvard; Nathaniel Rochester, the chief architect of IBM's first commercial computer; and Claude Shannon, the founder of information theory. The proposal asked for funding

to convene ten researchers at Dartmouth for two months in the summer of 1956. Their goal, stated with the breathtaking confidence that would come to define the decade, was to determine "how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves." McCarthy estimated this would take two months and ten people.

It did not take two months and ten people.

What the Dartmouth Summer Research Project on Artificial Intelligence actually produced, in the summer of 1956, was the name of a field. McCarthy coined the term "artificial intelligence" for the proposal partly to distinguish his work from Norbert Wiener's cybernetics, which McCarthy disliked for what he saw as its sloppy philosophical inclinations. The naming was political. By giving the field a new name, McCarthy could claim its founding for himself and his collaborators, and could shape it in opposition to the neuron-and-feedback tradition that had grown up around Wiener and the McCulloch–Pitts circle.

McCarthy and Minsky championed what came to be called the symbolic approach to artificial intelligence. Their conviction was that human intelligence was, at its core, the manipulation of high-level logical symbols. When you reason about whether to bring an umbrella, you are not running a neural network — you are running something more like a logical proof. *If it is raining, and I do not want to get wet, and an umbrella prevents wetness, then I should bring an umbrella.* The argument was that if you could just write down enough of these rules, with enough nuance and enough exceptions, a machine running them would eventually exhibit intelligence. You would build a mind from the top down, by encoding logic directly.

For a while, this seemed to work. McCarthy invented the programming language LISP in 1958, specifically designed for symbolic computation, and it became the lingua franca of AI research for the next thirty years. Minsky founded the MIT Artificial Intelligence Laboratory in 1959 and turned it into the most productive AI research centre in the world. His students built programs that did genuinely impressive things: SHRDLU, completed by Terry Winograd around 1970, could carry on a fluent conversation about a simulated world of coloured blocks, answering questions like "Does the red pyramid sit on top of the green cube?" and obeying instructions like "Put the small cube

into the box." To anyone watching, it looked as if the symbolic approach was on the verge of cracking general intelligence. Minsky told *Life* magazine in 1970 that human-level AI was "three to eight years" away.

It was not three to eight years away. The simulated block-world that SHRDLU navigated was a closed universe of twenty objects and a dozen verbs. Scaling that approach to the real world — a world of millions of objects, infinite verbs, contextual exceptions, sarcasm, lighting changes, and edge cases that no programmer could anticipate — proved to be a problem of fundamentally different character. By the mid-1970s, the symbolic approach was visibly stalling. In 1973, the British government commissioned the mathematician James Lighthill to report on the state of AI research; his report was scathing [26]. The Lighthill Report contributed to severe cuts in UK AI funding. The American Defense Advanced Research Projects Agency, which had been the main funder of AI in the United States, made similar cuts shortly afterwards. The field entered what became known as the first AI winter — a period of disillusionment, lost funding, and academic embarrassment that lasted, depending on how you count, between five and fifteen years.

Look at McCarthy's and Minsky's lifespans on the timeline. McCarthy died in 2011 at the age of eighty-four. Minsky died in 2016 at the age of eighty-eight. They outlived nearly everyone else from the pioneer era. They commanded funding and shaped the field for over half a century, and because of that longevity they had the time to see their rule-based reign flourish, stumble into winter, recover, and finally, in their last years, give way to a radically different approach that they had spent their careers dismissing. Minsky, in particular, lived long enough to watch the neural networks he had helped to discredit in 1969 become the dominant paradigm of artificial intelligence in the decade before his death. He never quite conceded the point.

2.4 The bottom-up rebel (Frank Rosenblatt)

While McCarthy and Minsky were writing rules, Frank Rosenblatt was teaching a machine to learn its own.

In 1957, Rosenblatt — a research psychologist at Cornell — proposed something that ran directly against the symbolic or-

thodoxy. His insight was simple, and it descended directly from McCulloch and Pitts: instead of programmers painstakingly writing down the rules of the universe, what if the machine learned its own rules from raw data? Rosenblatt called his invention the Perceptron [27]. It was a machine that guessed an answer, looked to see whether it had made a mistake, and mathematically adjusted its internal weights to do a little better the next time. Show it a thousand pictures of cats and a thousand pictures of dogs, label them, and the Perceptron would gradually learn the boundary between the two categories without anyone ever having to define a cat. It was bottom-up learning. It was the birth of the pattern-recognition engine that, sixty years later, would beat humans at Go.

Rosenblatt did not stop at the mathematics. With funding from the Office of Naval Research, he built a physical machine: the Mark I Perceptron, a room-sized device with 400 photocells wired to a bank of motorised potentiometers that adjusted the weights mechanically as the machine learned. The Mark I could distinguish printed letters and simple shapes. It was, in 1958, the world's first machine that visibly learned from experience.

The press loved it. On 8 July 1958, *The New York Times* published an article describing the Perceptron as "the embryo of an electronic computer that the Navy expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence." Rosenblatt, twenty-nine years old and giving press interviews in a borrowed lab coat, did not entirely discourage these predictions. The Perceptron, the Navy claimed, would soon be reading handwritten letters, recognising spoken words, and translating between languages. None of this was true in 1958. Almost all of it would eventually become true, decades later, using descendants of Rosenblatt's idea — but not in his lifetime.

The hype invited a backlash. In 1969, Marvin Minsky and his MIT colleague Seymour Papert published *Perceptrons* [28], a book-length mathematical analysis of what Rosenblatt's machines could and could not do. The book's central proof was devastating: a single-layer Perceptron, the kind Rosenblatt had actually built, cannot learn the exclusive-or function. XOR, the simplest logical operation that returns true when exactly one of two inputs is true, is mathematically impossible for the architecture Rosenblatt had spent a decade defending. The book did

not say that neural networks were fundamentally impossible — it said, carefully, that multi-layer networks might overcome the limitation. But the practical effect of the book was to give the symbolic AI community the ammunition they needed to dismiss connectionism entirely. Funding for neural network research fell sharply in the years that followed. Rosenblatt's career, which had been built on the promise of his machines, collapsed.

Two years later, in July 1971, Rosenblatt died in a sailing accident on Chesapeake Bay. He was forty-three years old. It was his birthday.

The timeline overlap is stark, and the chapter does not need to belabour it. Rosenblatt's line ends two years after the publication of the book that dismantled his life's work. The boating accident was almost certainly an accident — contemporary accounts describe a sudden squall and an inexperienced crew. But the symbolic synchrony of the timing has haunted the field ever since: the man who built the first learning machine died in the immediate shadow of the field's verdict that his approach was a dead end.

The verdict was wrong. It would take seventeen years to find out.

2.5 Keepers of the flame (Rumelhart & Hinton)

Rosenblatt's Perceptron was brilliant, and Minsky was right about its limitations. A single layer of artificial neurons could not handle XOR, and could not handle a thousand other patterns that mattered in the real world. The torch was dropped in 1969. For most of the next two decades, neural network research was something you did at the margins of computer science — in psychology departments, in fringe conferences, in research groups that did not need to apply for grants in the keywords "artificial intelligence" or "connectionism."

The torch was picked up by people who did not care that the rest of the field had moved on.

In 1986, the cognitive scientist David Rumelhart, together with Geoffrey Hinton and Ronald Williams, published a short paper in *Nature* titled "Learning Representations by Back-Propagating Errors" [29]. The paper described a training algorithm that Minsky and Papert had implicitly suggested might be possible: a

way to train neural networks with more than one layer.

The intuition behind backpropagation is genuinely elegant, and worth sitting with for a moment because the rest of this book depends on it. Imagine a network of artificial neurons stacked in layers, where the inputs to one layer are the outputs of the previous one. You feed the network an image of a cat. The network produces a guess — say, seventy per cent dog, thirty per cent cat. The guess is wrong. Backpropagation is the rule for assigning blame for that wrongness. It looks at the final layer and asks: how much did each of these neurons contribute to the error? Then it looks at the layer below and asks the same question, weighted by how much each lower-layer neuron contributed to the upper-layer neurons. The blame propagates backwards through the network, one layer at a time, all the way to the input. Then every weight in the network is nudged slightly in the direction that would have produced a better guess, and the process repeats with the next image. Do this a few million times and the network learns to recognise cats.

The mathematics had been worked out independently several times before — by Paul Werbos in his 1974 PhD thesis [30], by Seppo Linnainmaa in 1970 in a Finnish master's thesis on numerical methods [31], and arguably by several others. What Rumelhart, Hinton, and Williams did was publish it in *Nature*, demonstrate that it worked on practical problems, and make the case that neural networks were back. The 1986 paper is conventionally cited as the moment connectionism returned from exile.

It is Hinton's line that carries the narrative forward from here. Born in London in 1947, the great-great-grandson of the logician George Boole, Hinton trained as an experimental psychologist before moving into computer science. He spent the 1970s and 1980s as one of the loneliest researchers in the field, working on neural networks at a time when getting a paper accepted at a major AI conference required not mentioning that the system involved neural networks at all. He moved to Toronto in 1987, opposed to the Reagan administration's policy of funnelling the majority of US AI research funding through military agencies, and finding in Canada a place willing to support his work on different terms. For most of the 1990s, his research group at the University of Toronto was the global centre of a paradigm that

almost nobody else believed in.

The persistence paid off in 2012. Hinton, with his students Alex Krizhevsky and Ilya Sutskever, entered a deep neural network into the ImageNet visual recognition competition — a benchmark that the symbolic AI community had spent fifteen years failing to crack with rule-based approaches. The network, which the authors called AlexNet, won by a margin so large that the previous year's winner looked like it had not been trying [32]. Within eighteen months, every serious computer vision researcher in the world was working on neural networks. The paradigm shift was complete.

Hinton received the Turing Award in 2018 for his work on backpropagation. He resigned from Google in 2023, in part to speak more freely about the risks of the technology he had spent his life building. He received the Nobel Prize in Physics in 2024. His line on the timeline is the only one still moving. He is the bridge from the pioneers to the present, the last living connection between McCulloch and Pitts's 1943 paper and the language model you were using last week.

* * *

Look at the timeline one more time.

Turing's line ends in 1954. He never saw a programmable digital computer running a program he had not written by hand. Pitts and McCulloch both end in 1969, in the same year their paradigm was publicly entombed. Rosenblatt ends in 1971, two years after his life's work was dismissed. McCarthy ends in 2011, having lived long enough to see the field he named go through one full cycle of triumph and collapse, but not the second. Minsky ends in 2016, having lived to see the neural networks he discredited become the dominant paradigm of AI, and never quite admitting it. Only Hinton's line continues.

The theoretical framework for thinking machines was drafted by people who lived and died in the mid-twentieth century. The tragedy of early AI is that the ideas were mathematically sound long before the hardware was capable of sustaining them. Turing's 1936 paper described universal computation forty years before personal computers existed. McCulloch and Pitts's 1943

neural network model described the basic architecture of deep learning seventy years before AlexNet. Rosenblatt's 1958 Perceptron described the learning rule that, fifty years later, would beat human champions at Go. The pioneers were not wrong about what was possible. They were wrong about *when*.

It took over half a century for the sand — refined into silicon chips and scaled to billions of operations a second — to catch up to the dreams drafted by Turing, McCulloch, and Rosenblatt. Most of the people who drafted those dreams did not live to see the catching up.

But the dreams were drafted. That is the answer to why this chapter is here. Everything that follows in this book — the convolutional networks, the transformers, the large language models that have made artificial intelligence a household phrase — is built directly on the foundations laid down by people who died before they could see what they had started. The relay race did happen. The runners just had to hand the baton across decades, sometimes across deaths.

The next chapter takes up the baton at the bottom of the rebound: the moment in the 1980s when the field, chastened by the failures of pure symbolic AI, began to seriously ask whether the symbol people and the neuron people had been arguing about the wrong thing all along. The great war of early AI was not fought with weapons, but with philosophies. It was a war between machines that follow rules, and machines that find patterns. And, like most wars, neither side entirely won.

Part II

Learning to Learn

CHAPTER 3

Rules vs. Patterns

By 1956, the field of artificial intelligence had a name, a manifesto, and a roomful of brilliant people working on it. What it did not have was an answer to the question that was already, quietly, dividing its founders: what *kind* of thing is a mind?

There were two answers on offer, and they ran in opposite directions. The symbolic camp, led by McCarthy and Minsky, believed a mind was a system that manipulated symbols according to logical rules — and that the way to build a mind was to write the rules down, carefully and completely, in a programming language designed for the task. The connectionist camp, descended from McCulloch, Pitts, and Rosenblatt, believed a mind was a network of simple units that adjusted their connections in response to experience — and that the way to build a mind was to give the network examples and let it learn the rules itself. The two camps did not just disagree about implementation. They disagreed about what intelligence *was*.

For thirty years, the symbolic camp won. Then it stopped winning. This chapter is the story of why.

Arthur Samuel saw the problem early. Samuel was an electrical engineer at IBM in the 1950s, and in 1959 he published a paper that, on the surface, was about playing checkers [1]. What Samuel had actually done was something stranger: he had written a program that played checkers *without being told how*. Rather than encoding the rules of expert play — the openings, the strategies, the heuristics that human champions had

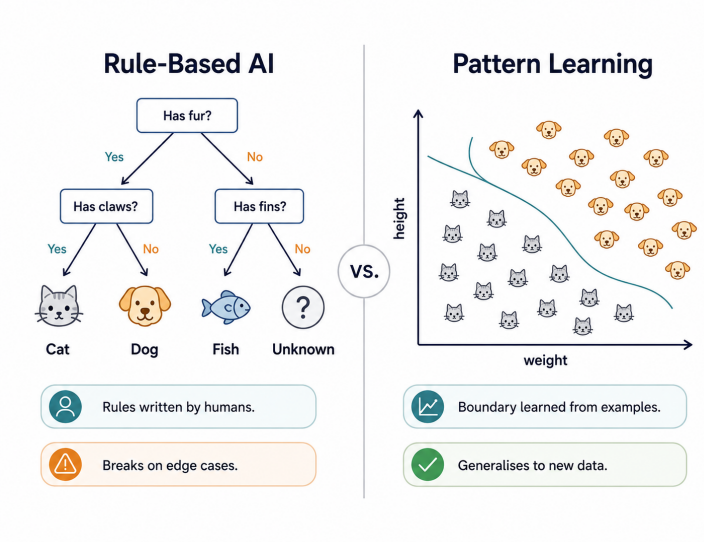


Figure 3.1. Two philosophies of artificial intelligence. Rule-based systems follow hand-written decision trees and can fail on edge cases, while pattern-learning systems infer boundaries from examples and can generalise to new data.

spent centuries refining — Samuel had given his program a way to evaluate board positions, then let it play thousands of games against itself. After each game, the program adjusted its evaluation function based on what had worked and what had not. By 1962, Samuel's program had won a publicised game against Robert Nealey, a strong Connecticut player, though later accounts often exaggerated the significance of the result. Truly world-championship-level computer checkers would not arrive until Chinook, developed at the University of Alberta, which became the first program to win a human world-championship title in checkers in the 1990s. Samuel had not written a checkers expert. He had started a lineage that eventually became one.

In 1959, this looked like a curiosity. By the 1980s, it would look like prophecy.

3.1 The illusion of logic (Top-Down AI)

For the first few decades of AI research, the field focused on rule-based systems. The intuition was deeply human-centric: when humans explain their reasoning, we use logical rules. *If the patient has a fever and a rash, and recently visited a malarial region, then test for malaria. If the contract was signed under duress, then it is voidable. If the chess opponent has castled kingside, then attack on the kingside.* Intelligence, in this view, was just formalised reasoning. Build a machine that could chain these rules together fast enough, and you would have built a mind.

The clearest expression of this philosophy is the decision tree [33]. In figure 3.1, on the left, we see a small decision tree for classifying animals. We ask a series of yes-or-no questions and arrive at an answer. Does it have fur? Yes. Does it have claws? Yes. Then we have a cat. The tree is interpretable: every classification can be traced back to a chain of human-readable rules. It is auditable, debuggable, and feels intuitively like how a careful thinker would reason. For most of the 1960s and 1970s, this was what AI research looked like.

And for a while, it worked extraordinarily well. Rule-based systems — often called *expert systems* — delivered the field's first genuine commercial successes. MYCIN, developed at Stanford in the early 1970s, was a rule-based system for diagnosing bacterial infections and recommending antibiotics [34]. It encoded the knowledge of expert physicians in roughly six hundred rules of the form *if these symptoms, then this diagnosis with this confidence*. In controlled studies, MYCIN performed at the level of the senior medical experts whose knowledge it had been built from — and significantly outperformed junior doctors. DENDRAL, also from Stanford, analysed mass spectrometry data to identify unknown chemical compounds [35]. PROSPECTOR, built in the late 1970s, was a rule-based system for mineral exploration that famously predicted the location of a previously undiscovered molybdenum deposit in Washington State.

Edward Feigenbaum, who led much of this work at Stanford, coined the phrase "knowledge engineering" to describe what was happening: the careful extraction of expert knowledge from human practitioners and its encoding into formal rules [36]. The phrase was not modest. It implied that human expertise was a form of structured knowledge that could be *engineered* —

captured, formalised, scaled, and eventually surpassed. By the late 1970s, the dominant view in the AI community was that knowledge engineering was the road to general intelligence. Build enough expert systems, integrate them, and the long-promised thinking machine would emerge.

McCarthy gave the field its preferred tool: the programming language LISP, which he had designed in 1958 specifically for symbolic computation. LISP made it natural to represent rules as data, to manipulate them programmatically, and to build systems of arbitrary complexity. By the early 1980s, an entire industry had grown up around LISP-based expert systems. Specialised computers called *LISP machines*, designed to run symbolic AI software efficiently, were sold by companies like Symbolics and Lisp Machines Incorporated. Major corporations — DEC, DuPont, American Express — deployed expert systems for tasks ranging from configuring computer orders to detecting credit card fraud. The Japanese government launched the Fifth Generation Computer Systems project in 1982, a major ten-year national initiative to build a generation of computers based on symbolic AI and logic-programming principles. The project's stated goal was to leapfrog Western computing entirely. The British and American governments responded with their own competitive programmes.

The symbolic approach had become an industry. And like all industries built on a single idea, it was vulnerable to the discovery that the idea did not scale.

3.2 The edge case explosion

The real world is not a chessboard. It is not even a hospital ward, where MYCIN's six hundred rules could capture most of what a doctor needed to know about bacterial infections. The real world is messier, in ways that rule-based AI was structurally unable to handle.

Go back to our decision tree. It worked for cats and dogs. What happens when the system encounters a Sphynx cat? The Sphynx is a hairless breed — a cat with no fur. The rigid logic of the tree forces a verdict: not a cat. The programmer's options are two. They can add a special rule for hairless cats. But hairless cats are not the only edge case. There are also Manx cats, which

have no tails. Persian cats, which have flat faces. Polydactyl cats, which have extra toes. Bengal cats, whose fur looks more like a leopard's than a domestic cat's. Each exception requires a new rule. Each new rule interacts with the existing rules in ways the programmer must carefully manage. And then someone shows the system a wet cat, which to a computer's eyes looks nothing like a dry cat, and the entire structure begins to creak.

This is the *edge case explosion*, and it is the rock against which symbolic AI broke. The world contains a vast, possibly unbounded number of exceptions to any rule a human can write down. Every rule needs sub-rules. Every sub-rule needs exceptions. Every exception interacts with every other exception. The complexity does not scale linearly with the size of the rule base; it scales combinatorially, because every new rule must be checked against every existing rule for consistency and conflict. By the time you have ten thousand rules in a system, the system is unmaintainable. By the time you have a hundred thousand, no human can fully understand what it does.

The most ambitious attempt to push through the edge case explosion was a project called Cyc, started in 1984 by the AI researcher Douglas Lenat [37]. Lenat's argument was that the real obstacle to artificial intelligence was *common sense* — the vast body of unstated knowledge that humans use to navigate ordinary situations. A toddler knows that water makes things wet, that you cannot push a rope, that if you let go of an object it falls, that people get angry when you take their things, that ice cream is cold, that birds fly and fish swim and dogs bark and cats do not. None of this is written down anywhere. It is the substrate of reasoning, not its product. Lenat proposed to write it down — millions of facts, encoded in a formal logic, by a team of philosophers and engineers, over what he estimated would be a few years of work.

Cyc has been running for over forty years. It has accumulated roughly twenty-five million rules. It has cost tens of millions of dollars. And it has never quite worked. By the late 1990s, Lenat was still claiming that the system was only a few years from human-level common sense. The same claim was being made in 2010. The project still exists, in a much-reduced form, and its current proponents are careful not to make predictions about timelines anymore. Cyc is the most expensive demonstration in computing history that you cannot write down common sense

by hand.

There were other clues that the symbolic approach was hitting structural limits. In the late 1970s, the AI researcher Roger Schank had proposed that human understanding of stories could be captured in formal structures he called *scripts* — stereotypical sequences of events for common situations [38]. The restaurant script. The birthday party script. The doctor's visit script. The idea was that if a machine had a library of scripts, it could understand stories that fit those scripts, because the script told the machine what to expect. Scripts worked beautifully for stories that fit them. They failed catastrophically for stories that did not. A restaurant story in which the customer was robbed at gunpoint did not fit the script. A birthday party at which the cake was on fire did not fit the script. The world contained too many scripts to enumerate, and the interesting moments in any story were precisely the ones where the script broke down. Schank's programmes could parse a story about ordering soup and getting a bill. They could not parse a story about anything that mattered.

By the mid-1980s, the field was acquiring a name for the structural problem: *the knowledge acquisition bottleneck*. Even when expert systems worked, they worked because a team of knowledge engineers had spent years interviewing experts and codifying what they knew. The bottleneck was not computing power. It was the human labour of extracting knowledge from experts and turning it into rules. And the experts themselves often could not articulate what they knew. A doctor diagnosing a rash, an engineer designing a bridge, a chess grandmaster choosing a move — none of them could fully explain what they were doing. They had *intuition*, by which they meant: knowledge that worked but resisted formalisation.

Intelligence, it turned out, could not be handed down from human to machine like a legal document. Most of what humans knew about the world, they did not know they knew.

3.3 The pattern recognition paradigm (Bottom-Up AI)

If the top-down approach failed because no one could write down enough rules, the bottom-up approach offered an inver-

sion. Instead of giving the machine the rules and asking it to find the answers, give the machine the answers and ask it to find the rules.

The right side of figure 3.1 shows the simplest possible version of this. We have a scatter plot of animals, with weight on one axis and height on the other. The cats cluster in one region; the dogs cluster in another. A pattern-learning algorithm does not need to be told what a cat is. It is given labelled examples — this one is a cat, this one is a dog, this one is a cat — and it draws a boundary that separates them. When a new animal arrives, the algorithm checks which side of the boundary it falls on. If the new animal is unusually tall or unusually light, the boundary accommodates it. If it does not fit cleanly, the algorithm reports its uncertainty. The boundary was *learned*, not legislated. And because it was learned from examples, it generalises to examples the programmer never anticipated.

This sounds elegant in the abstract. In practice, what made pattern learning take over the field was not philosophy. It was that pattern-learning systems started solving problems the symbolic approach could not.

The first crack came in postal sorting. By the mid-1980s, the United States Postal Service was processing tens of billions of letters per year, each of which had to have its zip code read by a human operator if the address was handwritten. Rule-based handwriting recognition — which tried to characterise each digit by its shape, its strokes, the angles between its lines — had been attempted for two decades and had never worked well enough to deploy commercially. Yann LeCun, a young French researcher who had spent the early 1980s working on neural networks at Hinton's lab in Toronto, took a different approach. He trained a multi-layer neural network on tens of thousands of handwritten digits, using the backpropagation algorithm we met in chapter 2. By 1989, LeCun's system was reading handwritten zip codes on real envelopes with accuracy good enough for commercial deployment [39]. By the mid-1990s, neural network systems descended from LeCun's work were reading roughly ten per cent of all handwritten cheques in the United States. The same approach that had been dismissed as a fringe research programme was quietly doing one of the largest pattern-recognition jobs in the world economy.

Speech recognition followed a similar trajectory. For most of

the 1970s and 1980s, the dominant approach had been symbolic: try to encode the rules of phonetics, identify the phonemes in the audio signal, look up the corresponding words in a dictionary. This worked poorly. Speech is not a clean sequence of phonemes; it is a continuous stream of overlapping sounds, shaped by the speaker's accent, mood, gender, age, the acoustics of the room, and a dozen other variables a rulebook could not anticipate. In the late 1980s, a small group of researchers — led by Frederick Jelinek at IBM — abandoned the rule-based approach entirely and replaced it with statistical models trained on large corpora of transcribed speech [40]. Jelinek summarised the philosophy with a line that became infamous in the linguistics community: "every time I fire a linguist, the performance of the speech recognizer goes up." He was, in retrospect, exactly right.

Credit card fraud detection took the same route. So did optical character recognition for printed text. So did medical image analysis. Through the late 1980s and 1990s, a quiet pattern emerged: in any domain where rule-based AI had stalled, statistical and pattern-learning approaches eventually overtook it. The transition was not philosophical. It was driven by deployment. Companies that needed working systems chose the approach that worked, and the approach that worked was almost always the one that learned from data rather than the one that ran on hand-written rules.

The symbolic AI community resisted this shift for a long time. They had built careers around the conviction that intelligence was logic, and conceding ground to the statistical approach felt like conceding that intelligence was *just* statistics — a kind of category error, dignifying mere pattern-matching with the name of thought. The argument was philosophically serious and it remains alive today, in the form of the Hinton–Marcus debate we encountered at the end of chapter 1. But while the philosophical argument continued, the deployments kept happening. By the late 1990s, the question of whether pattern learning could match human performance in narrow domains had been answered. It could. The remaining question was how far the approach could be pushed.

3.4 The second winter

The expert systems industry of the early 1980s had grown fast. It collapsed faster.

In 1987, the market for LISP machines — the specialised hardware that ran symbolic AI software — evaporated almost overnight. Two things had happened. First, ordinary workstations from Sun Microsystems and other manufacturers had become powerful enough to run LISP software directly, eliminating the need for dedicated hardware. Second, and more importantly, the expert systems that had justified the entire industry began to fail in the field. They were brittle. They could not handle situations outside the narrow domain of their training. Maintaining them, as new exceptions accumulated, cost more than the systems had cost to build in the first place. Corporate buyers, who had been promised flexible artificial intelligence, found themselves stuck with inflexible rule-bases that needed expensive human attention every time the world changed.

The Symbolics company, which had been the leading manufacturer of LISP machines and at one point operated the very first commercial `.com` domain name, went into freefall. Its stock dropped from a high of $35 to under $1 within two years. Lisp Machines Incorporated went bankrupt. Texas Instruments and Xerox, which had been making competing LISP hardware, exited the market. The Japanese Fifth Generation project, which had been the flagship of state-funded symbolic AI, quietly missed its goals and was wound down at the end of its decade-long run, having produced no commercial products of significance.

The funding followed the hardware. DARPA, which had been the largest funder of AI research in the United States, cut its AI budget significantly in the late 1980s and redirected the money toward more applied work. Academic AI departments shrank. Graduate students, who in 1985 had been queueing up to work on expert systems, were by 1990 advised to avoid the term “artificial intelligence" on their job applications. The field had a name for this. The first AI winter, in the 1970s, had killed neural networks. The second AI winter, in the late 1980s and early 1990s, killed the symbolic paradigm that had killed them.

This was the field’s second great failure of prediction, and it deserves to be acknowledged honestly. The pioneers had promised thinking machines in a decade. Forty years on, they

had produced expert systems that could not recognise a Sphynx cat. The gap between the promise and the delivery was not a matter of insufficient funding or insufficient computing power. It was a matter of the wrong paradigm. Knowledge, it turned out, was not something you could engineer. Intelligence, it turned out, was not the same thing as logic.

What survived the winter was, ironically, the approach the field had spent two decades dismissing. While the LISP machines were being scrapped, Geoffrey Hinton's research group in Toronto was quietly training neural networks. Yann LeCun's zip-code system was reading mail at the postal service. Jelinek's statistical speech recognisers were being deployed in telephone systems. The connectionist tradition, which the symbolic camp had triumphantly declared dead in 1969, had spent the winter doing useful work. It would emerge, in the 2000s and then explosively in the 2010s, as the dominant paradigm of the entire field.

The verdict of the great war between symbols and patterns was clear by the late 1990s, even if it took another decade for the field to admit it out loud. The symbol people had been right that intelligence involved structure, abstraction, and the manipulation of meaningful representations. They had been wrong that you could build those representations by hand. The pattern people had been right that intelligence emerged from learning, from many examples, from gradients flowing through layers of simple units. The next two parts of this book are about what happened when the pattern people got the computing power to prove it.

3.5 The bridge to the machine that learns

Arthur Samuel had seen all of this coming, in his quiet way, in 1959. His checkers program had not encoded the rules of expert checkers play; it had learned them from experience. Samuel had not framed his work as a manifesto against symbolic AI — he was an engineer, not a polemicist — but the implication was already there in his paper. The interesting kinds of intelligence might be the kinds you could not write down. The interesting kinds of expertise might be the kinds the experts themselves could not articulate. If you wanted a machine that could match

a human in a complicated task, you might do better to give it a way to improve from experience than to give it a set of rules.

But Samuel's checkerboard, like SHRDLU's blocks and MYCIN's bacterial infections, was a tidy mathematical universe. The real test of pattern learning was whether it could handle the messy, chaotic, infinite-edge-case physical world — the world of faces in different lighting, of speech in different accents, of handwriting in different hands. To take that test, the field needed a particular kind of machine: a small, fast, self-correcting decision-maker that could be wired together into networks of arbitrary complexity, and that could learn from its mistakes by adjusting its own connections.

And just as McCulloch and Pitts had done in 1943, the researchers who built such a machine found the blueprint hiding inside our own heads.

* * *

The symbolic approach did not fail because its practitioners were wrong about intelligence. It failed because they were wrong about how much of intelligence could be written down. Rules can capture what experts know they know; they cannot capture what experts know without knowing that they know it. The pattern-learning approach did not solve that problem by being cleverer. It solved it by giving up on rules entirely, and learning from examples instead. That shift — from writing down knowledge to extracting it from data — is the quiet revolution this chapter has been describing. Everything that follows in this book is a consequence of it.

CHAPTER 4

The Neuron That Started Everything

The neural network has been a label in this book for three chapters. Chapter 1 mentioned them. Chapter 2 told the story of the people who imagined them and the people who almost killed them. Chapter 3 showed why their approach, in the end, scaled where rule-based AI did not. But until now the reader has been taking on faith that something specific is happening inside these machines. This chapter is where the faith ends. By the time you reach the end, you should be able to read a sentence like “GPT-3 has 175 billion parameters trained on 300 billion tokens" and know what every word means. The whole of modern AI is built on the principle this chapter explains, and the principle is genuinely simple — which is part of why it took so long for anyone to believe it would work.

4.1 The committee in your head

The artificial neuron was inspired by the biological neuron, but it is not a model of one. It is a vast simplification — a cartoon of the real thing, and a neural network is not, in any meaningful sense, an attempt to simulate the brain. What we borrow from biology is the basic principle: a unit that receives signals from many inputs and fires when those signals accumulate past a threshold. McCulloch and Pitts, whom we met in chapter 2, took

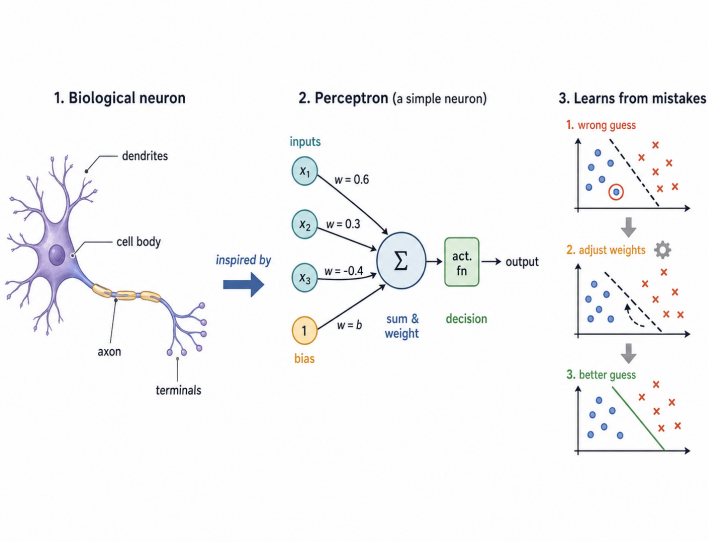


Figure 4.1. The perceptron was inspired by the biological neuron: multiple inputs are weighted, combined, and passed through a simple decision function. Unlike a fixed rule, the perceptron can improve by adjusting its weights after mistakes.

that principle and abstracted it into mathematics. Everything that follows is built on the abstraction, not the biology.

A good metaphor for an artificial neuron is a committee meeting with a single chairperson. The committee members are the inputs. Each member casts a vote — a number, positive or negative. The chairperson does not weigh all votes equally; some members are trusted more, some less. The chairperson sums the weighted votes, and if the total crosses a threshold, the motion passes and the neuron fires.

Here is a concrete example. Imagine a neuron whose job is to decide whether a photograph is of a cat or a dog. Its inputs are simple features the system can measure: the size of the animal, the length of its fur, the shape of its ears, the presence of whiskers. The weights are how much each feature counts toward the decision: ear shape might count for a lot, fur length for less, size for almost nothing because cats and dogs overlap heavily in size. The neuron multiplies each input by its weight, adds the results, and if the sum is high enough, it fires. Cat.

It is worth naming the parts now, because the vocabulary will recur in every chapter that follows. The inputs to a neuron are called *features*. The numbers that determine how much each input counts are called *weights*, or sometimes *parameters*. The threshold the sum has to cross is called the *bias*. The sum-and-threshold step is called the *activation function*. The output is the neuron's *prediction*. Five terms. That is the entire vocabulary of a single artificial neuron, and it is the vocabulary modern AI is built on.

A single neuron, by itself, is not interesting. It is a glorified if-statement. The interesting question is what happens when you wire many of them together and let them figure out their own weights — and that question is what the rest of the chapter is for.

4.2 Learning is just adjusting the weights

When you first build a neural network, the weights are random. The neuron is a committee whose members have been chosen at random and given random levels of trust. Its predictions are nonsense — no better than a coin flip. The cat-dog neuron starts out giving you wrong answers about half the time, because the weights are not yet tuned to which inputs actually matter.

This is what training fixes. You show the neuron an example: a photograph of an animal, with its ear shape, fur length, size, and whiskers measured and fed in as inputs. The neuron makes its prediction. You compare the prediction to what the animal actually was — a cat or a dog. If the neuron predicted “cat" and it was a dog, the weights on the inputs that pushed it toward cat" get nudged down. If the neuron predicted “dog" and it was a cat, the weights get nudged up. Then you show it the next example. Then the next. Thousands of examples, millions of examples, until the predictions are reliably right.

This is the central claim of the chapter, and it is worth stating plainly: learning, in a neural network, is nudging the weights up and down. That is all it is. There is no understanding happening. There is no insight. There is no aha-moment inside the neuron. There is a numerical procedure that adjusts weights based on whether predictions were right or wrong, and that procedure, repeated enough times, produces a system that gets

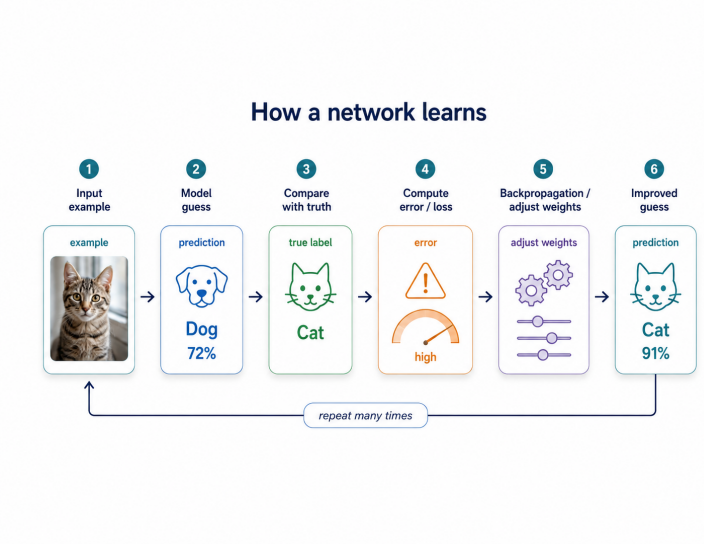


Figure 4.2. A neural network learns by repeatedly making predictions, measuring its error, and adjusting its weights. Early guesses may be wrong, but each correction nudges the model toward better predictions. Training is this loop repeated over many examples, often millions or billions of times.

predictions right. Every neural network in this book — every image recogniser, every chess engine, every language model — works on this principle.

In a network with many neurons, the question of which weights to nudge and by how much is the question backpropagation answers. Chapter 2 already introduced the algorithm; the detail to remember here is that backpropagation makes the training loop work for any size of network, not just for a single neuron. The mathematics scales.

When AI researchers and engineers talk about “training a model," this is what they mean: starting with random weights, showing the model examples, adjusting the weights based on the errors, and repeating until the weights are tuned. The word will appear constantly in the rest of the book. You now know what it points to.

A modern neural network does not have one neuron with four weights. It has billions of neurons with trillions of weights.

The training loop is the same. It just runs at a scale that requires industrial computing infrastructure.

4.3 Why one neuron isn't enough

A single neuron is a powerful tool, but it has a hard mathematical limit. The reader heard about this limit in chapter 2 — Minsky and Papert proved in 1969 that a single-layer perceptron cannot compute the XOR function [28] — but the reason *why* was glossed. This section is the why.

Geometrically, a single neuron draws a straight line. Imagine the scatter plot from chapter 3: cats on one side of a graph, dogs on the other, with a line dividing them. The neuron *is* that line. It computes whether each new data point falls on the cat side or the dog side. As long as the two categories can be separated by a single straight line, a single neuron can learn to do it. The cat-dog decision from the previous section, in the version where the four features cleanly separate the animals, is exactly this kind of problem. A neuron can solve it. But most real-world problems cannot be separated by a straight line. The XOR problem is the simplest example. Place four points on a piece of paper: top-left and bottom-right both labelled *true*, top-right and bottom-left both labelled *false*. Try to draw one straight line that puts the trues on one side and the falses on the other. You cannot. No straight line can separate a checkerboard pattern, and a single neuron's representational power is fundamentally bounded by what one straight line can carve out. The world contains many patterns no straight line can capture.

The same problem shows up in the cat-dog example as soon as you make it realistic. Suppose you add a third category — foxes — whose features overlap with cats on some dimensions and dogs on others. A single line through the feature space cannot cleanly separate three categories arranged this way. The neuron starts misclassifying. Add a fourth and fifth category, and the geometry becomes hopeless. The single neuron is not, in any deep sense, wrong about how to learn. It has simply run out of representational power.

The fix is to stack neurons in layers. When you put one neuron's output into another neuron as input, something interesting happens: the second neuron is now drawing a straight line not

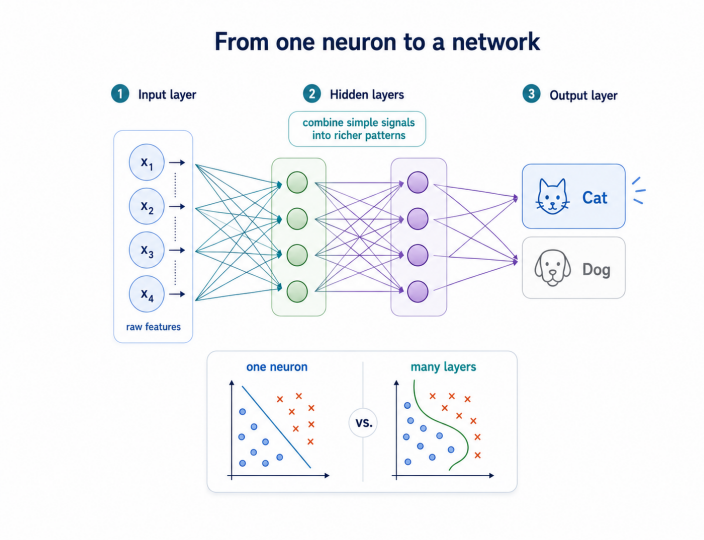


Figure 4.3. A multilayer perceptron turns many simple neurons into a hierarchy of decisions. The input layer receives raw features, hidden layers combine those features into increasingly useful patterns, and the output layer produces the final prediction. Unlike a single neuron, which can only draw a straight decision boundary, multiple layers can represent curved and complex boundaries.

in the original input space, but in the *transformed* space the first neuron created. Each layer transforms the space the next layer sees. By the time you have stacked four or five layers, the network can carve out boundaries of arbitrarily complex shape — not a straight line but a curve, not a curve but a region, not a region but a constellation of regions. This is the central architectural fact of deep learning.

And that, finally, is what makes deep learning *deep*. The word refers to the number of layers stacked between input and output. Shallow networks have one or two layers and are limited in what they can represent. Deep networks have dozens or hundreds and can represent almost anything. You have heard the phrase "deep learning" a thousand times without anyone telling you what makes it deep. Now you know.

This is also why Minsky and Papert's 1969 critique was technically correct but practically wrong. They proved that single-layer

networks couldn't handle XOR. They explicitly noted that multi-layer networks might. The field heard the first half and ignored the second, and the result was the seventeen-year winter we covered in chapter 2. The technical fix was always there. What was missing was a way to train multi-layer networks efficiently — the backpropagation algorithm that Rumelhart, Hinton, and Williams finally published in 1986 [29].

4.4 What a layer actually does

If stacking neurons is the trick, what are the deeper neurons *seeing*? The first layer sees the raw input. What do the layers above it see?

Take a neural network trained to recognise images of cats and dogs. The first layer of the network sees raw pixels — the same grid of numbers we met in chapter 5's preview. The neurons in this first layer learn to detect very simple things: a vertical edge in one part of the image, a horizontal edge in another, a patch of high contrast somewhere else. They are not yet detecting anything we would call a feature of a cat or a dog. They are detecting features of *images in general*.

The second layer takes the outputs of the first layer as its inputs. Now the neurons can combine the edges and contrast patches the first layer found, and they learn to detect slightly more complex patterns: a corner where two edges meet, a curve, a region of textured fur. The third layer combines those into recognisable parts: the pointed shape of a cat's ear, the floppy shape of a dog's ear, the round geometry of an eye, the texture of a whiskered muzzle. The fourth layer combines parts into whole animals. By the time the signal reaches the output layer, the network has built a representation of the entire image at multiple levels of abstraction, and the final decision — cat or dog — is a straightforward consequence of what was found along the way.

Here is the part that genuinely surprised the field when it first became apparent. Nobody tells the network what these intermediate representations should be. The network *discovers* them, layer by layer, because the architecture forces each layer to build on the patterns the layer below it found. The hierarchy emerges from training. A system that nobody designed to look

for edges first and faces last learns to do exactly that, because it turns out to be the most efficient way to perform the task.

This is not metaphor. Researchers can inspect what each layer of a trained network has learned to detect, by examining which inputs cause each neuron to fire most strongly. Some of the most famous images in modern AI are these layer visualisations: abstract patterns of edges and curves in the early layers, recognisable objects in the deeper ones. The hierarchy is empirically observable.

The same principle holds across every domain neural networks have been applied to. In speech recognition, the early layers of the network learn phonetic features — the acoustic shape of a vowel, the burst of a consonant. The middle layers learn words. The late layers learn sentence structure. In a language model, the early layers learn the structure of tokens, the middle layers learn syntactic relationships, and the late layers learn something we still do not fully know how to name but which behaves a lot like meaning. The hierarchy-of-representations principle is the spine of modern AI, and it shows up everywhere.

This is the idea the rest of the book is going to keep delivering on. Chapters 5 and 6, on vision, will show how convolutional networks make this hierarchy concrete and efficient — how, specifically, a network learns to recognise a cat. Chapter 7 will show how the first layer of a language network represents words geometrically. Chapter 8 will show how attention lets each layer focus on the patterns that matter. Chapter 9 will show what happens when this principle is scaled to the limits of what hardware can run.

One honest caveat. The hierarchy is real, but the *interpretability* of any single neuron's role gets murkier in the deeper layers. Edges are easy to identify in the first layer. By the time you are at layer 50 of a 100-layer network, the features the network has learned often resist neat human description. The neuron is firing for something, but no human-readable label fits. This is the seed of what chapter 10 will call the interpretability problem.

4.5 Why this is harder than it sounds

You have just been handed a clean four-step picture. Neurons are weighted voters. Learning is adjusting weights. Stacking

layers gives you representational power. Each layer learns a more abstract representation than the one below. Every part of modern AI is built on these four ideas. Which raises the obvious question: if it is that simple, why did it take seventy years from McCulloch and Pitts to ChatGPT?

The principle was settled in 1986. Making it actually work required overcoming three engineering walls that have nothing to do with the principle itself.

The first wall is the sheer number of weights. A small useful network might have ten thousand of them. A modern image recogniser has tens of millions. A large language model has hundreds of billions to trillions. You cannot adjust these by hand. You adjust them with backpropagation running on specialised hardware, and even then a single training run can take weeks. The arithmetic involved in training a frontier model is comparable to the lifetime computational output of the entire pre-2010 software industry. The principle of weight adjustment is simple. The scale at which it has to be done is not.

The second wall is data. A neural network learns from examples, and the number of examples it needs grows with the complexity of what it is trying to learn. To learn to recognise cats and dogs reliably, the network needs millions of labelled images, in different lighting, from different angles, of different breeds, against different backgrounds. To learn language, it needs hundreds of billions of words of text. Where does that data come from? How is it labelled? Who owns it? These are not minor footnotes. They are the questions that have driven the entire AI industry for the past fifteen years, and they have produced most of the field's current controversies — about copyright, about consent, about whose work was used to train whose model.

The third wall is hardware. AlexNet, the network that broke ImageNet in 2012 [32], took five to six days to train on two GPUs. GPT-4 is rumoured to have cost hundreds of millions of dollars to train and required a small power station's worth of electricity. The mathematics of deep learning was settled forty years ago. The field had to wait for graphics-card manufacturers to accidentally build hardware perfectly suited to running it — GPUs, originally designed to render video game graphics, turn out to do the matrix multiplications neural networks need at orders of magnitude greater speed than ordinary processors. The field then had to spend a decade buying that hardware by

the data-centre.

The principle of neural networks is genuinely simple. The reason it took until 2012 for the principle to break through is that the scale required to make it work was beyond what anyone could afford until very recently. The story of modern AI is not the story of a brilliant new idea. It is the story of a sixty-year-old idea finally meeting the hardware that could run it.

The next part of the book takes up the story at the moment the hardware finally caught up — and it starts where the catching-up was most visible, in the problem that had stumped AI for forty years: teaching machines to see.

* * *

A single artificial neuron is almost nothing — a weighted sum, a threshold, a number. Stack enough of them into layers, train them on enough examples, and something unexpected happens: the network begins to carve the world at its joints. Not because anyone told it where the joints were, but because the structure of the data made those divisions the most useful ones to learn. This is the central surprise of connectionism, and it is still surprising. The rules were never written down. They were never even designed. They emerged, weight by weight, from the accumulated pressure of getting the answer wrong and adjusting until it was less wrong. That process — simple, mechanical, repeated billions of times — is what the rest of this book is built on.

Part III

Machines That See

CHAPTER 5

Teaching Machines to See

How do you explain a cat to something that has never seen one? Not by pointing, because the system has no eyes. Not by description, because the system has no language. You explain a cat by feeding the system millions of photographs of cats and hoping it figures the rest out. Whether this would work was an open question for forty years. The field of computer vision was founded on the conviction that machines could be taught to recognise the world the way humans do, and for four decades it produced steady, slow progress on narrow problems and almost no progress on the general one. The story of why is the story of this chapter — and the story of how the problem was finally cracked is the story of the next.

5.1 The pixel grid

Machines see in pixels: a grid of numbers. Each number represents the light intensity at one point on the grid, typically ranging from 0 to 255. For grayscale images — black and white — there is one intensity, or *channel*, for each position. For colour images there are three channels, one each for red, green, and blue. The intensity at each position is just a number, and the image itself is just the grid those numbers form.

The scale of these grids has grown. A modern phone camera produces images with around twelve million pixels, which

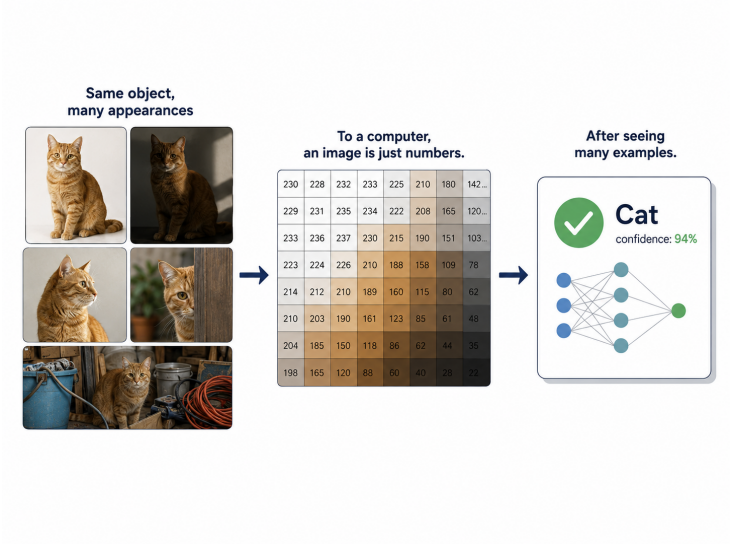


Figure 5.1. To a computer, an image is only a grid of numbers. The challenge of vision is that the same object can appear under many different conditions; learning from many examples allows a model to recognise the underlying object despite changes in lighting, angle, occlusion, and background.

means thirty-six million numbers per photograph. The whole image is just that array of numbers. There is no “cat" in it. There are only the numbers, and the meaning is not in the numbers themselves. It is in the pattern the numbers form. The pattern is invisible to anything that does not already know how to look.

And here is the problem the field could not solve for forty years: the same cat, photographed twice, produces two different patterns. Different lighting, different angle, different background — and the numbers change. Sometimes most of them change. A system trained to recognise one pattern of numbers will fail on a different pattern of numbers, even when the two patterns are pictures of the same animal.

5.2 The same object, a thousand different pictures

The figure at the start of this chapter shows the same cat in four different photographs. To a human, the four pictures are

obviously the same animal. To a computer reading the pixel grids, the four photographs have almost nothing in common.

Start with lighting. The cat in bright midday sunlight is a grid of high numbers — the pixels are mostly bright. The same cat in shadow is a grid of low numbers — the pixels are mostly dark. The pattern in the sunlit photograph is not a slightly-changed version of the pattern in the shadowed photograph. The patterns are, numerically, completely different. A system that has learned what the sunlit cat's pixels look like will see the shadowed cat as a different object entirely.

Then there is angle. The cat photographed from the front, the cat from above, the cat from behind, the cat curled up on a couch, the cat mid-stretch — each of these produces a different shape in the pixel grid. The cat from the front has eyes near the top of the cat-shape. The cat from above has no eyes visible at all. The cat curled up is roughly circular; the cat mid-stretch is roughly rectangular. A system that has only learned "cat means this configuration of pixels" will fail on every angle except the one it was trained on. Cats do not stay still.

Then there is occlusion. The cat half-hidden behind a chair. The cat photographed through a wire fence. The cat whose own tail is across its face. To a human, these are obviously still cats — we mentally fill in the missing parts. To a pixel-counting system, the cat is incomplete. Some of the cat is, from the system's point of view, just not there.

Then there is background. The cat on a couch, the cat in a tree, the cat on a sidewalk, the cat in front of a brick wall. The cat is the same in every photograph. Everything around the cat changes completely. A system that learns "cat means this configuration of pixels" has a deeper problem than it knows: most of the pixels in any photograph of a cat are not the cat. The system has to learn to ignore the background — and the background is most of what is there.

Human vision handles all of this so effortlessly that we do not notice we are doing it. We can recognise a friend's face in sunlight, in shadow, from three-quarters profile, with half of it hidden behind a coffee cup, against any background, at any distance within reason, and we never have to stop and think. This operation has a name. It is called *invariance* — recognising the same object across all the variations the world can produce. Building a system that performs invariance is one of the hardest

problems in artificial intelligence. For forty years, the field had no idea how to do it.

5.3 Why the rule-based approach failed

For most of the period from 1960 to the early 1990s, the dominant approach to computer vision was rule-based and symbolic — the same approach that chapter 3 said had broken on the rock of edge cases. The idea was to write down rules for what a cat looks like. *A cat has four legs. A cat has pointed ears. A cat has whiskers. A cat has roughly these proportions.* A program would then extract features from the image — edges where the pixel values changed sharply, corners where two edges met, regions of similar colour — and check the features against the rules. If enough rules matched, the system would declare the image a cat.

There is a famous early example of how badly this could go. In the summer of 1966, Marvin Minsky — whom we met in chapter 2 — assigned an undergraduate student at MIT a summer project [41]. The project was to connect a video camera to a computer and write the software needed to describe what the camera was looking at. Minsky genuinely seemed to think this was a one-summer problem. The student did not finish the project that summer. Computer vision, in slightly different forms, continued for the next forty years and did not produce a working general-purpose object recogniser. The summer project became a running joke in the field — a marker of how badly the early pioneers had misjudged the difficulty of what they were attempting.

The reason the approach failed was that every rule had counterexamples, and the counterexamples were everywhere. *A cat has four legs* — except in the photograph where two of its legs are tucked under its body, or the photograph taken from the side where only two legs are visible, or the photograph of a cat lying down. *A cat has pointed ears* — except for the Scottish Fold, a breed whose ears fold flat against its head, or the cat photographed from behind with its ears not visible at all. *A cat has these proportions* — except for kittens, who have proportionally larger heads and shorter limbs, and except for cats curled up, whose proportions are altered by their own posture. Every

rule worked for most cats and failed for some, and "some" was always large enough to matter.

Worse, every feature the system extracted carried its own ambiguity. An edge in the pixel grid could be part of a cat or part of the chair the cat was sitting on. A patch of orange could be a tabby's fur or a curtain. A pointed shape could be an ear or a tree branch behind the cat. Before the system could decide whether the image contained a cat, it had to solve a deeper problem: which part of the image *was* the cat in the first place. This question turned out to be at least as hard as the original. The pixels did not come with labels. A computer staring at a grid of numbers had no way to know which numbers belonged to the cat and which belonged to the world the cat happened to be in.

This is the edge-case explosion from chapter 3, and vision was its canonical case. Every rule needed exceptions. Every exception needed sub-exceptions. Every sub-exception interacted with every other exception in ways no programmer could fully anticipate. By the mid-1980s, working computer vision researchers had quietly given up on rule-based approaches for general object recognition and were experimenting with statistical methods. The statistical methods were the right idea. They just did not work on the hardware of the time. The field would spend another twenty years waiting for the computers to catch up.

5.4 The shape of the problem the field couldn't solve

By the late 1990s, the field of computer vision had a clear sense of what it could and could not do. It could do narrow things. Optical character recognition for printed text had been more or less solved — you could buy commercial software that read scanned books reliably. Industrial vision systems could identify a small fixed set of parts moving past on a factory conveyor belt. Military reconnaissance systems could spot tanks in satellite imagery, given enough preprocessing and a cooperative angle of view. The list of working applications was real, but every entry on it was a domain where the variation had been engineered away — where the lighting was controlled, the angle was known, the object set was small.

What the field could not do was the general problem. Hand a system a random photograph from anywhere on the internet and ask it what was in it. The answer would come back wrong, often spectacularly wrong, and there was no path from the systems that worked on conveyor belts to the systems that could handle the open world.

The reasons were clear, even at the time. Whatever the right approach was, it had to do four things at once. It had to handle the variation we walked through in section two — the same object across lighting, angle, occlusion, and background. It had to do this without requiring a programmer to write down what each object looked like, because the rule-based approach had failed for exactly the reasons the field could no longer ignore. It had to scale to thousands of object categories rather than the dozen or two that existing systems could distinguish. And it had to do all of this on hardware that actually existed.

The first three of these requirements were known by the mid-1990s. The fourth was the wall the field kept hitting. Approaches existed on paper that satisfied requirements one through three. A neural network with enough layers, trained on enough data, could in principle learn to perform invariance — chapter 4 has just walked us through the principle. Yann LeCun had even built a working version for handwritten digit recognition in the late 1980s [39], and it was, by the late 1990s, reading roughly ten per cent of all handwritten cheques in the United States. The mathematics was settled. The problem was that scaling LeCun's approach to general object recognition — thousands of categories, millions of training images, networks with many more layers — would have required computing resources that did not exist anywhere in the world.

One number captures how large the gap was. By 2010, the best general-purpose object recognition systems in the world were evaluated on a benchmark called ImageNet [42], which we will meet properly in the next chapter. The systems were given a photograph and allowed five guesses about what was in it. The best system in 2010 was correct, on at least one of its five guesses, about 72% of the time [43]. A human, doing the same task, was correct about 95% of the time. A 23-point gap between machine and human on a five-guess task is the polite way of saying the machine could not see.

In 2012, the gap closed. By half in a single year, and to near-

human performance within five years. The technique that did it was not new. The architecture had been worked out by Yann LeCun in the late 1980s and quietly refined by a small group of researchers, including Geoffrey Hinton, through the long winter of the 1990s and 2000s. What changed in 2012 was that the hardware finally caught up — exactly as the previous chapter promised. The story of that catching-up, and of the architecture that turned it into the breakthrough that broke the field open, is the story of the next chapter. It is the story of the convolutional neural network.

* * *

The difficulty was never the pixels. Computers could read pixel grids from the beginning. The difficulty was that the same object — a chair, a face, a dog — arrives at the sensor as a completely different grid of numbers depending on the angle, the lighting, the distance, and what happens to be behind it. No rule could cover all of that variation, because the variation is, in principle, infinite. What the field needed was not a better description of what objects look like. It needed a way to learn what objects look like from examples numerous enough and varied enough to cover the cases no rule-writer could anticipate. That is a different kind of problem entirely. It is also, as the next chapter shows, a solved one.

CHAPTER 6

Convolutional Networks and the Art of Noticing

By September 2012, the question of whether machines could see — open for forty years, as chapter 5 just established — was answered. The answer was yes. The system that answered it was a kind of neural network called a convolutional neural network. This chapter is the story of why that architecture worked when nothing else had. It is also the story of how the chapter 4 principle — hierarchies of representations, each layer building on the one below — gets made concrete for one specific problem. And it is the story of one famous benchmark, ImageNet, on which the entire history of modern computer vision can be told.

6.1 The trick: looking at patches, not pixels

The neural network we introduced in chapter 4 would treat each input as independent. But pixels in an image that are adjacent to each other are related; neighbouring pixels could form an edge, a shape, or a contour — what the field calls *local connectivity*. The kind of network we have so far has no way to learn these spatial patterns. It is also wasteful. Take a 224×224 colour image: approximately 150,000 pixels, each with three colour channels, so 450,000 inputs. If the first layer of the network has even modest size — say 10,000 neurons — that is 4.5 billion weights to learn between the input and the first layer alone. Computationally



Figure 6.1. A convolutional neural network learns to see by building patterns in layers. Early filters detect simple edges, later layers combine those edges into textures, shapes, and parts, and the final representation supports an object label such as "cat."

impractical. And worse, the network has no way to capture the spatial structure that makes an image an image.

This is where the convolution comes in: small filters sliding across the image. Each filter looks for a specific local feature — a vertical edge, a colour patch, a curve. The convolutional layer takes the image as input, slides its filters across it, and outputs a new grid showing where each filter's pattern occurred. Because the filters slide everywhere across the image, the convolution can find its pattern regardless of where it appears. This property is called *translation invariance*. Take an image of a cat: that cat can be in the top-left or the bottom-right, and the same filters will detect it. The convolutional network does not have to learn separately what a cat-in-the-top-left looks like and what a cat-in-the-bottom-right looks like. They are the same pattern, found in different places.

The architecture of the convolutional neural network was inspired by the work of two neurophysiologists, David Hubel and Torsten Wiesel, in the 1960s, exploring the visual cortex

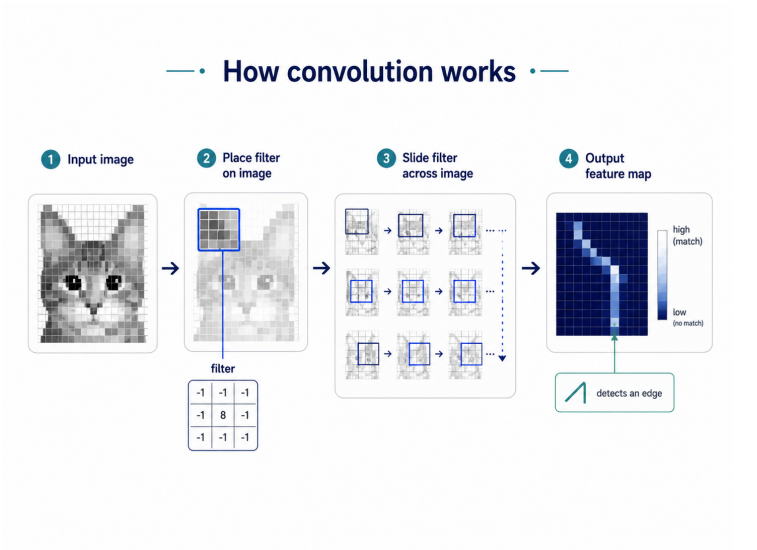


Figure 6.2. A convolutional filter scans across an image looking for a small local pattern, such as an edge. At each position, the filter produces a response showing how strongly that pattern is present. The result is a feature map: a new image-like representation that highlights where the network has noticed useful structure.

of cats [44]. They found that individual neurons in the early visual cortex were activated by specific local features — edges at particular orientations — and that the cortex was organised in layers of increasing abstraction. In the late 1980s, Yann LeCun and his colleagues turned this principle into a neural network architecture [39]. The convolutional neural network is biology-inspired in a deeper way than the neuron of chapter 4. But, as before, the principle was abstracted, not simulated.

6.2 Stacking the patches: depth in convolutional networks

The typical CNN has 64 filters in the first convolutional layer, each looking for a different local pattern. After they have been trained, these filters end up with striking similarities across different networks and different problems. They learn to detect

edges at various orientations, simple colour blobs, and basic textures. What is most surprising — and a recurring theme in artificial intelligence — is that no one programmed them to do this. Nobody told the network to look for edges. The network discovered, through training, that edges were the right thing to look for first. As with the cat cortex, every higher-level abstraction in vision can be built out of edges.

The second convolutional layer applies a new set of filters to the output of the first. The first layer's input was raw pixels; the second layer's input is the edge detectors' output. The filters in this layer learn to combine those edges into more complex patterns. A corner is two edges meeting at an angle. A T-junction is one edge ending on another. A curve is a sequence of edges at slowly changing angles. The second layer learns shapes from edges.

By the time we get further into the network — layers four and five of our cat-detection CNN — the filters start to detect parts that have human-recognisable names. A neuron that fires strongly when there is an eye in its receptive field. Another that fires for a tail. Another for a textured region of fur. Layer ten might have neurons that fire for whole faces, or whole animals. The hierarchy that chapter 4 promised is, in convolutional networks, empirically visible.

Researchers have inspected trained convolutional networks by finding the input image that most strongly activates each neuron. Do this across the network and a gallery emerges: blurry edges in the early layers, geometric shapes in the middle layers, recognisable parts in the deeper layers, and whole-object specimens in the final layers. The hierarchy is not a metaphor. You can look at it.

Between convolutional layers, CNNs use an operation called pooling, which downsamples the spatial map. Like a filter, it slides a small square over its input and returns a single number — usually the maximum value in that square. This shrinks the image as it flows through the network. The input might be 224×224 pixels, but by the time the signal reaches the deepest layers, after several rounds of pooling, the spatial map may only be 7×7. The point of pooling is to introduce another form of invariance: the network stops looking for the exact position of the eye and instead notices that there is an eye *somewhere in this region*. This is how the architecture handles the small variations

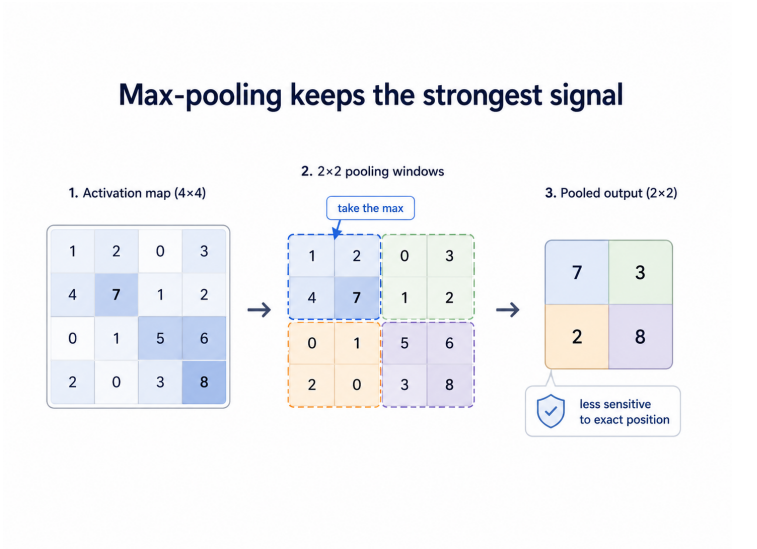


Figure 6.3. Max-pooling shrinks a feature map by keeping only the strongest activation in each small region. This preserves the most important signal while discarding some exact positional detail, making the network less sensitive to small shifts in where a feature appears.

in pose and angle that chapter 5 worried about.

Translation invariance from convolution. Pose invariance from pooling. The architecture has been engineered, layer by layer, to handle the variations that broke rule-based vision.

6.3 ImageNet and the 2012 moment

In 2007, Fei-Fei Li, a Stanford computer scientist, started the ImageNet project. Her idea was contrarian: the field of AI had been focussing on algorithms, but the real hurdle in computer vision was not algorithms, it was data. There were no large, labelled datasets that captured the diversity of the real world. Li and her team spent two years and roughly $100,000 of Amazon Mechanical Turk labour assembling a dataset of 14 million labelled photographs across 22,000 object categories. The dataset was released in 2009 [42]. The wider research community, accustomed to working with academic benchmarks containing

thousands of images, did not initially know what to do with it.

From 2010 onward, ImageNet hosted a competition called the Large Scale Visual Recognition Challenge (ILSVRC), using a subset of 1.2 million images across 1,000 categories. The task was simple: given an image, make five guesses about what was in it. The competition was scored on top-5 error — the percentage of images for which none of the five guesses were correct. In 2010, the best system scored 28% top-5 error. In 2011, the best system scored 26%. These were rule-based systems with hand-engineered features, the kind chapter 5 said had stalled. Progress was real but slow, and it was slowing further [43].

Geoffrey Hinton's group at the University of Toronto decided to enter the competition for the first time in 2012. Two of Hinton's graduate students, Alex Krizhevsky and Ilya Sutskever, had spent the previous year building a convolutional neural network they called AlexNet. The network had 8 layers, 60 million parameters, and was trained over the course of a week on two consumer-grade GPUs. They entered it into the competition and scored 15.3% top-5 error [32]. Second place that year scored 26.2%. The gap between first and second was larger than the entire previous decade's worth of progress.

The aftermath was decisive. In 2013, almost every entry to the competition used CNNs; by 2014, every entry did. By 2015, Microsoft's ResNet [45] won with 3.6% top-5 error, beating estimated human performance on the same task [43]. The field had spent forty years failing to solve general object recognition. It went from unsolved to near-human in three.

Within eighteen months of AlexNet, every major technology company in the world had hired or acquired a deep learning team. Google, Facebook, Microsoft, Apple, Baidu, Tencent — all reorganised their research priorities around convolutional networks. Funding into AI poured in at a scale unseen since the 1980s. The entire AI summer of the 2010s, leading to the language models the book will reach in the next part, descends from a single competition entry on a single day in September 2012.

6.4 What convolutional networks see (and what they miss)

These networks could handle the four invariance problems we introduced in chapter 5. They generalised across angle, lighting, occlusion, and background in ways no previous approach had managed. They scaled to thousands of categories. In 2012, they ran on commodity GPU hardware. A couple of years later, they could run on your phone. Computer vision had been solved. It had not.

CNNs are surprisingly easy to fool. A couple of years after AlexNet, researchers discovered you could take an image of a cat that a CNN identified with 99% confidence, change a tiny fraction of the pixels in ways imperceptible to humans, and the network would suddenly identify the same image as an ostrich, or a bookshelf, or a school bus [46, 47]. These are called *adversarial examples*, and they are not random failures. They are deliberately crafted images designed to exploit the network's decision boundaries. The system had learned to recognise cats, but in a way subtly different from what humans recognise. The two systems agree on natural images. They disagree, dramatically, on adversarially constructed ones.

The deeper problem is that convolutional networks can pass benchmarks like ImageNet with flying colours but do not necessarily *understand* what they are looking at, in any deep sense. Take a CNN trained on photographs of cows in fields. Show it a cow on a beach, and it will often misclassify. Not because it cannot see the cow, but because it has implicitly learned that field" is part of what cow" means [48]. The features it has learned are statistical regularities of its training data, not the abstract concept of cowness. The gap between passing the benchmark and actually understanding turns out to be very real. Chapter 10 will return to this.

None of this diminishes the achievement of 2012. Vision had been one of the most stubborn problems in artificial intelligence, and solving it — even partially, even with caveats — unlocked everything that followed. The same convolutional architecture, with modifications, would soon be applied to speech recognition, to medical imaging, to autonomous vehicles, to robotic control. Within five years of AlexNet, CNNs were the default architecture for anything spatial.

But vision was only one of the constellation points from chapter 1. The question of whether neural networks could handle language — words with no spatial structure, where meaning depends on context, where the same word means different things in different sentences — was still open. The next chapter is the story of how the field began to solve it.

* * *

What convolutional networks demonstrated, definitively, in 2012 was that the right architecture matters as much as the right data. AlexNet did not win ImageNet by being fed more photographs than its competitors. It won by being structured to notice the things that matter in photographs — edges before textures, textures before shapes, shapes before objects — in the same order that the visual world itself is structured. That alignment between architecture and domain is the lesson the field took from the moment, and carried forward. The next problem on the constellation was language. Words, unlike pixels, do not arrive in a grid. They arrive in sequence, and their meaning depends not on where they sit in space but on what surrounds them in time. That required a different architecture entirely.

Part IV

Machines That Understand

CHAPTER 7

The Meaning of Words

How would you explain a “bank" to a machine? Not by definition, because a definition is just more words. Not by example, because the same word means a place that holds money and the edge of a river. For most of computing history, machines treated words as opaque symbols, with no notion that “cat" and “dog" were more alike than “cat" and “thermodynamics." For a system trying to understand language, this was an immediate dead end. Words needed to mean something, and meaning had to come from somewhere a machine could see. This chapter is the story of one strange idea: that the meaning of a word might be represented as a position in geometric space. It is also the story of how that idea broke on the word “bank."

7.1 Words don’t sit on a number line

How do you teach a machine a word? Specifically, how do we get a word into a neural network? In chapter 4 we learned that neural networks take numbers as input. Unfortunately, words aren’t numbers. We need *embeddings* — words represented as numbers.

The simplest solution is the number line. Every word gets a unique slot set to 1, with all other slots set to zero. For a 50,000-word vocabulary, we get a 50,000-dimensional vector with a single 1 in it, for each word. (A vocabulary is the set of words

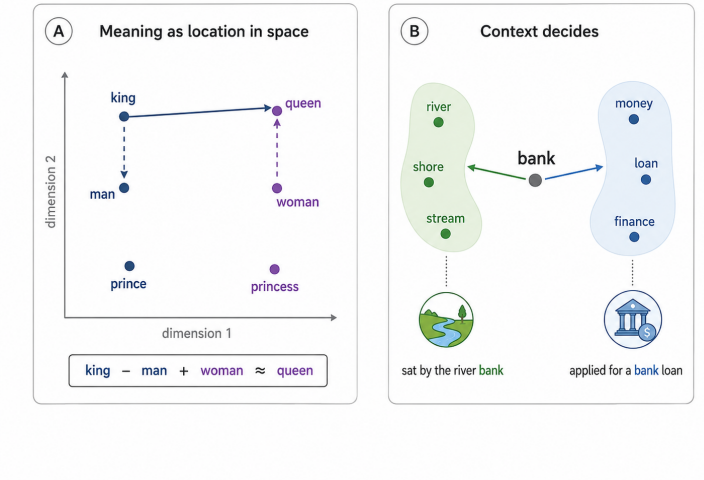


Figure 7.1. Word embeddings treat meaning as position in a learned geometric space, where relationships between words can become directions. Context then decides which region of meaning matters: the word “bank" shifts toward rivers in one sentence and toward finance in another.

the network has embeddings for.) This approach has one fatal flaw: every word is the same distance from every other word. “Cat" and “dog" are no closer than “cat" and “thermodynamics." With this one-hot encoding, our word embeddings have captured none of the semantics of words. We would expect an ideal system to place similar words close together and dissimilar words further apart. A system that cannot capture the similarity of different words — that does not realise “cat" and “dog" are alike — cannot generalise. For this structural reason, the rule-based approach to natural language processing failed.

So how do we formalise the intuition that similar words should sit closer together than dissimilar ones? And how do we measure the similarity of words at all? Firth’s 1957 distributional hypothesis gave the answer: a word’s meaning is given by the company it keeps [49]. Words that appear in similar contexts probably mean similar things. Take the two sentences “the cat sat on the mat" and “the dog sat on the mat." Both “cat" and “dog" appear in the same context, so they must mean similar

things. This was the principle the field would eventually use to build its embeddings.

7.2 Meaning as position

Word embeddings define meaning as a position in a high-dimensional space. In that space, words with similar meanings are close to each other; words with different meanings are farther apart. That is all a word embedding is: a representation of a word as a vector — an array of numbers — in a high-dimensional space. Typically these word embedding spaces have 100–300 dimensions, not the 50,000 of our failed one-hot encoding attempt.

In 2003, Yoshua Bengio and his colleagues built the first neural language model to put this intuition into practice [50]. Bengio's network learned word representations as a side effect of predicting the next word. They had the right idea, but the hardware and the architecture were not yet ready. A decade later, with the rise of consumer-grade gaming GPUs and a simpler architecture, the field caught up. In 2013, Tomas Mikolov and his colleagues at Google demonstrated what became the most famous example of word embeddings: Word2vec [51]. During training, the model took a word, tried to predict its neighbours, and adjusted the embeddings when it was wrong. Repeat across billions of words of text. After training, words that appeared in similar contexts ended up in similar positions in the embedding space.

As with our previous neural networks and the convolutional networks before them, nobody had explicitly programmed the semantics. Nobody told the network what each word meant. The network had discovered meaning from co-occurrence, just as Firth's distributional hypothesis predicted. A year later, Stanford proposed GloVe [52], which relied on global co-occurrence statistics rather than Word2vec's local sliding windows. Where Word2vec was a reader — moving through the corpus one window at a time — GloVe was an accountant, first tallying how often each pair of words appeared near each other across the entire corpus and then fitting embeddings to those totals. Different mechanisms, similar outcome: embeddings that capture meaning.

7.3 The geometry of meaning

Beyond capturing the similarity of words based on context, embeddings also turned out to capture something stranger: the geometry of meaning itself. Word embeddings are vectors, and in mathematics we can perform simple operations on vectors: addition, subtraction. In a famous 2013 paper, Mikolov and his colleagues showed that you could take these word embedding vectors and manipulate them to demonstrate that the embedding space had learned semantically meaningful structure [53]. The famous example was the following. Take the vector for “king," subtract “man," add “woman," and you land approximately on the vector for “queen." In short: “king $-$ man $+$ woman $\approx$ queen." This worked because the vector pointing from “man" to “woman" is roughly the same as the vector pointing from “king" to “queen." Gender, it turned out, is a direction in the embedding space.

Other directions appeared too. Tense (walking $\rightarrow$ walked). Plurality (cat $\rightarrow$ cats). Country-to-capital (France $\rightarrow$ Paris). The embedding space had a structure that nobody had designed. Semantic relationships were preserved as geometric relationships. This was a genuinely surprising result in 2013, and it is still, today, kind of weird.

There are limitations. Critics have pointed out that many of the cleanest analogies were cherry-picked, and the embeddings fail on plenty of others. But the broad claim survives: the embedding space of words carries semantic structure. The deeper point is that meaning, in these systems, is *relational*. A word’s meaning is its position relative to other words.

* * *

Language had been solved. It had not.

7.4 The “bank" problem

Word2vec did not solve the problem that some words have more than one meaning. The architecture assigned each word exactly one embedding — one point in vector space. But take the word

“bank," which has at least two meanings: the institution that holds money and the edge of a river. A single point cannot represent both. The static embedding ends up as a kind of weird average — closer to “money" and “deposit" because the finance-bank is more common in text, but also somewhat close to “river" and “shore."

This is the problem of *polysemy*, and it is not just “bank." “Bat" — animal versus sports equipment. “Run" — a verb with dozens of meanings. “Light" — weight versus illumination. For humans, the context surrounding each word disambiguates its meaning instantly. For static word embeddings, it does not.

The fix has to involve looking at the sentence around the word — at the context of each word relative to every other word in the sentence. The architecture that does this is called the *transformer*, and it learns context through an operation called *attention*. The next chapter is the story of how attention solved the bank problem.

* * *

Word embeddings were a genuine breakthrough — meaning as geometry, relationships as directions, the distributional hypothesis made computational. But they carried a flaw that their elegance obscured: every word had one address, and language does not work that way. The word on the page is never just the word. It is the word in this sentence, next to these neighbours, in service of this meaning. A static embedding cannot know which meaning is intended, because it was never given the sentence. What the field needed next was a way to let the context speak — to allow the meaning of each word to shift in response to everything around it. That mechanism has a name, and the next chapter is about it.

CHAPTER 8

Attention Is All You Need

Chapter 7 ended with a problem. Word embeddings represented each word as a single point in space, and a single point cannot hold two meanings. The word “bank" broke the system. The fix had to involve looking at the words around each word — not just which words appear in similar contexts across a corpus, but which words appear next to this specific word in this specific sentence. The mechanism that does this is called *attention*. This chapter is the story of where attention came from, and of the 2017 paper that took attention from a clever addition to existing architectures and made it the entire architecture. The result was called the transformer, and it did far more than solve the bank problem. It became the foundation of every large language model you have ever heard of.

8.1 The problem attention solves

To return to chapter 7’s bank problem: a single static word embedding cannot represent a word with multiple meanings. A word embedding has to be all things at once, and if it is static, it cannot. The fix is to look at the words around each word — the actual context of this word in this sentence, not just the statistical patterns of co-occurrence that produced the static embedding in the first place.

Before the transformer, the dominant architecture for sequence modelling was the recurrent neural network, or RNN

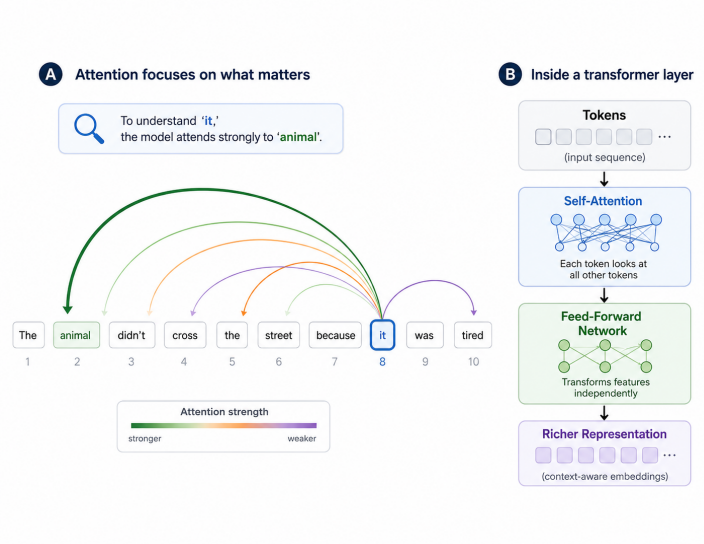


Figure 8.1. Attention allows a model to focus on the words that matter for interpreting a token. In a transformer layer, each token attends to other tokens, passes through a feed-forward network, and emerges as a richer context-aware representation.

[54]. RNNs process a text one word at a time, from left to right, carrying a running summary of everything they have seen so far. The problem with RNNs is that the running summary gets stale over long distances. Consider the sentence “the trophy didn’t fit in the suitcase because it was too big." What does “it" refer to? The trophy. Now change one word: “the trophy didn’t fit in the suitcase because it was too small." Same sentence, different referent. To know what “it" means, the model has to wait for a word that comes much later. RNNs struggle with these long-range dependencies because their running summary forgets. What is needed is for every word to be able to look at every other word, directly, regardless of distance — and attention does exactly that.

8.2 Paying attention

Attention lets each word *query* the rest of the sentence. Every word in the sentence raises its hand and asks “who else here is relevant to me?" It gets back a weighted answer — most words contribute a little, a few words contribute a lot.

Take the word “bank" in the sentence “I deposited my cheque at the bank." Here, “bank" attends most strongly to “deposited" and “cheque." Now take “bank" in the sentence “we sat by the bank of the river." Here, “bank" attends most strongly to “river" and “sat." The word’s representation is no longer static. It is updated based on what the word paid attention to. The same word, in different sentences, ends up with different representations. This is what we call *contextual embedding*, in contrast to chapter 7’s static embeddings. The problem of polysemy dissolves. “Bank" is no longer one fixed point in space — it is a position that depends on the sentence it appears in.

This book focuses on ideas rather than equations. When you encounter attention in a textbook, the mathematics will involve scaled dot products and softmax operations. Those are the machinery that compute attention. What matters here is the principle, and the principle is this: meaning is no longer fixed. It is negotiated, sentence by sentence, by each word looking at every other word.

8.3 Attention is all you need

In a 2017 paper, Vaswani and his colleagues at Google published “Attention Is All You Need" [55]. Eight authors. A deliberately provocative title. Before 2017, attention had been used as a supplement to RNNs, helping with long-range dependencies in machine translation [56]. Vaswani’s group asked the more radical question: what if we drop the RNN entirely and use only attention? The architecture they proposed was called the *transformer*, and it changed AI forever.

The architecture consists of stacked transformer *blocks* — one block after another, with the output of each block fed as input to the next. Each block contains an attention mechanism and a feed-forward neural network of the kind we met in chapter 4, with normalisation steps in between (a technique that scales and recentres the inputs to keep training stable). Each block refines

the representations of the block before it — the chapter-4 hierarchy again. The output of the stack is a sequence of contextual embeddings, one per input word.

This was a big deal. The attention mechanism was *parallelisable* in a way that RNNs were not. Parallelisable means many operations could be performed at once, rather than sequentially. RNNs had to process words one at a time, left to right, the way you are processing the words on this page. Transformers can process all the words in a sentence simultaneously — they look at the entire sentence at once. On modern GPUs, this made transformers vastly faster to train than RNNs. It also let them scale to model sizes RNNs could not have reached.

The 2017 paper was on machine translation — the task of translating from one language to another. But the architecture went on to do far more than that. Within two years, every state-of-the-art language model was a transformer. BERT, GPT, T5: all descendants of the 2017 paper.

8.4 Why this changed everything

There are a few things that make a transformer different from previous neural architectures.

First, it has no built-in notion of sequence. Instead it relies on *positional encodings* — another vector, added to each word's embedding, that tells the transformer the order of the words. This is technically clunky, but it turns out to be enormously flexible. The transformer can be applied to anything you can serialise into a sequence. We have since seen vision transformers (images as sequences of patches), music transformers, protein transformers.

Second, attention is general. The same architecture is used for language, vision, audio, code, mathematics. The transformer wasn't a language tool that happened to scale — it was a sequence tool that turned out to handle language extraordinarily well, and then everything else too.

Third, the architecture scales. Bigger transformers, trained on more data, simply work better. This is the scaling-laws observation that drives modern AI. The bigger, the better. Chapter 9 will continue on this thread: what happens when you scale a transformer to 175 billion parameters.

The transformer architecture is now a decade old. Every major AI system you have heard of — GPT, Claude, Gemini, Llama — is a transformer underneath. The next chapter is what happens when you make them very, very large.

* * *

The transformer did not just solve the problem of long-range dependencies in text. It solved it in a way that scaled. Earlier architectures improved incrementally as they grew larger; the transformer improved dramatically. More parameters, more data, more compute — and the returns kept coming, in ways nobody had predicted and nobody fully understood. That scaling behaviour is what made the transformer not just a better architecture but a different kind of object: a system whose capabilities were determined less by what its designers put in than by how much it was trained on. The implications of that shift are what the next three chapters are about.

CHAPTER 9

The Large Language Model

When you make language models very, very large, you get *large language models* — LLMs. LLM is a piece of technical jargon you will have heard many times, often used interchangeably with the phrase AI. But as the introduction to this book noted, AI is not just LLMs. The large language model is the transformer architecture from chapter 8, scaled up to billions, sometimes trillions, of parameters. By the end of this chapter, the promise made back in chapter 4 — that the gobbledygook sentence “GPT-3 has 175 billion parameters trained on 300 billion tokens" would become fully readable — comes due. This chapter is the story of what scale did to the transformer, and of the systems that emerged when a simple idea was made very, very large.

9.1 Words, actually tokens

Chapters 7 and 8 talked about transformers in terms of words; that was a simplification, and the reality is slightly different. Language models operate on *tokens*, not words. A token is a chunk of text — sometimes a whole word, sometimes part of one, sometimes punctuation, sometimes a single character. Take the sentence “the cat sat on the mat" — that might be six tokens, one per word. A less common word like “tokenisation" might split into two: “Token" and “isation." A rare word might become four or five tokens.

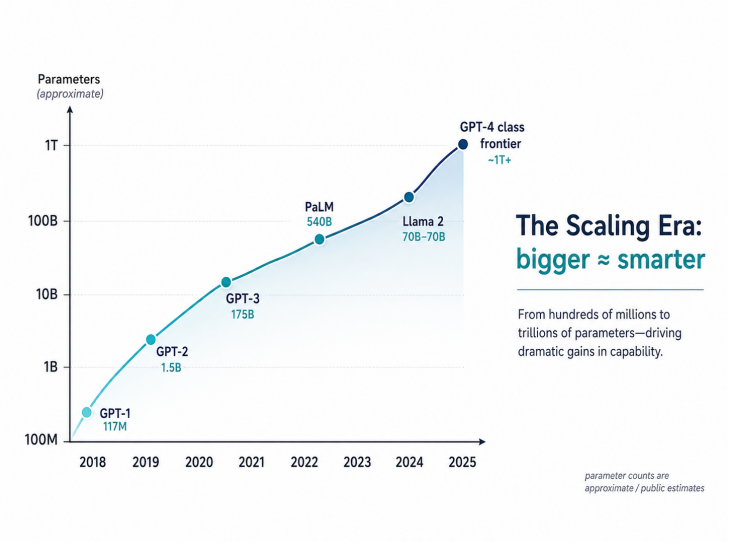


Figure 9.1. The modern language-model era has been shaped by dramatic increases in model scale. Parameter counts are only approximate and not always publicly disclosed, but the broad trend is clear: larger models, trained on more data with more compute, have tended to produce striking jumps in capability.

Why use subwords over words? Pure word-based vocabularies are too big (millions of words) and break on rare words, typos, and proper names the model has never seen. Byte-Pair Encoding (BPE) is a common tokenisation algorithm: it builds a vocabulary from frequent character sequences in the training data. Modern language models use their own variants of BPE, but they all descend from the same family.

To make things concrete, GPT-3's vocabulary is roughly 50,000 tokens. This is why text generation appears one token at a time, not one word at a time. And this matters for what comes next. When we say a model was "trained on 300 billion tokens," that is roughly 230 billion words — but the model never actually sees the word. Only the token.

9.2 Predicting the next token

The training task for a language model is staggeringly simple: predict the next token. Given a sequence of tokens, output a probability distribution over the next one. That is it. That is the entire training objective for GPT-3, Claude, Llama — all of them. Show the model a sentence, hide the next token, ask the model to guess, and adjust the weights when it is wrong. The weights are adjusted through the backpropagation algorithm we met in chapter 4

The model is the transformer architecture from chapter 8, so it processes the whole context at once and uses attention to decide what is relevant. Why does this work? Because predicting the next token requires modelling everything about the text — grammar, facts, reasoning, style. The simple task of next-token prediction forces the network to learn language itself, teaching the machine to understand.

Or so you might think.

This returns us to the detective novel example from chapter 1. "And the killer is ____." To predict the next token in that sentence, the language model must have followed the plot of the entire book. It must have weighed the evidence, tracked the suspects, grasped the motives. This is Hinton's argument: that prediction at scale requires understanding — or at least behaviour so indistinguishable from understanding that the word *understanding* is no longer doing much explanatory work.

Marcus's counter-argument from chapter 1 also resurfaces. Prediction at scale can produce the right answer through sophisticated pattern-matching, without anything that deserves the name of understanding. The language model has read millions of detective novels. It has learned the textual cues that signal which character a story is pointing at. It does not need to follow the plot in any meaningful sense; it needs only to recognise the patterns that books like this one tend to follow. The model is, on this account, a remarkably good pattern-matcher trained on a remarkably large corpus.

The debate between Turing and Searle has not gone away. It has only moved out of thought experiments and into systems the reader was probably using last week. Which side, reader, do you lean toward?

9.3 Scale changes everything

The arc from 2018 to 2020 introduced the *large* in large language models. GPT-1 (117 million parameters, June 2018), GPT-2 (1.5 billion, February 2019), GPT-3 (175 billion, May 2020) [57, 58, 59]. All three were the same architecture — the transformer we introduced in chapter 8. All three had the same training task: predict the next token. The only difference was scale. And as the models got bigger, the GPT series displayed an astonishing finding: they got qualitatively better. GPT-1 was a curiosity. GPT-2 wrote coherent paragraphs. GPT-3 could write code, answer questions, hold conversations. Nothing in the architecture changed. Just more parameters (deeper models from chapter 4), more data, and more compute.

This intuition — that bigger models are better models — was formalised in the scaling-laws paper from Kaplan and his colleagues in 2020 [60]. They found that model performance improves predictably with scale. There is a mathematical relationship between parameters, data, compute, and the model's capability. This was a strange result. Most scientific fields don't have laws that say "make it bigger and it gets better in a predictable way."

A 2022 paper from DeepMind called Chinchilla refined the picture further, showing that earlier models had been undertrained relative to their size [61]. For a fixed compute budget, smaller models trained on more data outperformed larger models trained on less. GPT-3, by this measure, had been undertrained. Models built after 2022 — Llama, Claude, most frontier systems — followed the new prescription.

With that, the scaling laws became the prophecy that built the modern AI industry. The bigger, the better — and language models grew larger at an accelerating rate. The scaling laws governed the economic incentives for AI research and development. If you kept scaling, the curve kept going up.

9.4 Emergence

Some capabilities of language models appear only at scale, in a way the scaling laws didn't quite predict. A 1-billion-parameter model cannot do three-digit arithmetic; a 100-billion-parameter model can. A 10-billion-parameter model cannot answer ques-

tions in an unfamiliar language; a larger model can. These are called *emergent capabilities*: behaviours that appear at scale without being explicitly trained for [62].

Probably the most famous example is *in-context learning*. Show GPT-3 two or three examples of a task in the prompt, and the model can perform the task without any weight updates — no further training, just examples shown in the input. Nobody told the model how to do this. It just emerged.

A second example is even stranger. In 2022, Jason Wei and his colleagues at Google showed that giving a model a few examples of step-by-step reasoning in the prompt dramatically improves its performance on multi-step problems — a technique they called *chain-of-thought prompting* [63]. A related discovery, published shortly after by Kojima, Gu, and colleagues, was even stranger: you did not need to provide any examples at all [64]. Simply appending the phrase "let's think step by step" to a question was enough. Ask a model a multi-step maths question and it often gets the answer wrong. Ask the same model the same question with that phrase tacked on, and it works through the problem one step at a time and frequently gets it right. The strangeness is that neither technique works on small models. Below roughly 60 billion parameters, the prompt does nothing. Above that threshold, it unlocks a capability the model could not access at any prompt length when it was smaller. Nobody designed this. Nobody trained the model on chain-of-thought examples specifically. The ability to produce step-by-step problem-solving when asked to was a property of scale.

The emergence framing is contested. Some researchers argue that emergence is partly an artefact of how we measure capabilities — that what looks like a sudden jump on a benchmark is, on smoother measures, a gradual improvement [65]. The principle still survives, though: as models scale, new capabilities show up that nobody predicted. This is the source of much of the fascination and the alarm around modern AI. The builders of these systems genuinely do not know in advance what their models will be capable of.

Now the chapter 4 promise comes due. "GPT-3 has 175 billion parameters trained on 300 billion tokens." 175 billion parameters is 175 billion weights, each adjusted during training by nudging the numbers up and down based on prediction error, through backpropagation. 300 billion tokens, where tokens are subword

chunks of text, is roughly the text content of every book ever published, plus much of the public internet. You, reader, now understand every word.

9.5 ChatGPT, Claude, and the present

The single most consequential release in AI history came on November 30, 2022. OpenAI released a research preview called ChatGPT. It was built on GPT-3.5 — a refined version of GPT-3 — that had undergone an additional training step called *reinforcement learning from human feedback*, or RLHF [66]. The idea behind RLHF was to align the model's responses with human preferences. The model learned not just to predict text, but to produce text people found helpful and unobjectionable. This was not a piece of groundbreaking technical research; the building blocks already existed. What was new was the product — a chat interface that was free to use, that anyone could try. Within five days it had a million users. Within two months it had 100 million. It was the fastest consumer adoption of a product in tech history.

The reaction reshaped the AI industry. Google declared a "code red." Microsoft invested $10 billion in OpenAI. Every major tech company accelerated its language-model roadmap. More importantly, ChatGPT was the moment AI stopped being an industry conversation and became a public one. Debates about jobs, schools, copyright, and safety all began here.

A parade of frontier language models has followed. GPT-4 in March 2023, with capabilities a step beyond GPT-3 and an undisclosed parameter count. Claude, released in the same month by Anthropic — the company that built the system I have been talking to while writing this book — with a different training emphasis on safety and helpfulness. Llama, released by Meta in February 2023, was the first widely available family of open-weight models, which meant the weights could be downloaded and run locally rather than only accessed through a provider's cloud service. Gemini, released by Google in December 2023, was multi-modal from the start: it could process images and text together, not just one or the other. More recently, open-weight models from outside the United States — DeepSeek from China, Qwen from Alibaba — have matched or exceeded the capabili-

ties of closed-weight Western models, reshaping the assumption that frontier AI would be a primarily American project.

The current state is this: dozens of frontier models, all transformers, all trained at scales unimaginable a decade ago. Training GPT-4 reportedly cost around $100 million in compute alone. As with the AlexNet moment from chapter 6, but scaled further, we have hardware to thank for making it possible. NVIDIA's H100 GPUs, and data centres consuming roughly the same amount of electricity as a small city.

It took nearly sixty years to go from McCulloch and Pitts to the moment the architecture met the hardware. These systems are extraordinary. They are also limited in specific, often surprising ways — the *jagged frontier* we introduced in chapter 1. The next chapter is an honest account of those limits.

* * *

The large language model is the strangest object the field has produced. It was not designed to translate, or summarise, or write code, or pass the bar exam. It was designed to predict the next token. Everything else — the fluency, the apparent reasoning, the surprising competence across domains its creators never anticipated — emerged from doing that one thing at sufficient scale, on sufficient data, for sufficient time. Whether that emergence constitutes understanding, or merely its most convincing imitation yet, is a question this book has been honest about not settling. What is not in question is the gap: between what these systems can do and what we understand about why they can do it. That gap is not a footnote. It is where the next chapter lives.

Part V

The Dream, Revisited

CHAPTER 10

What Machines Still Can't Do

Last week I asked a frontier language model, in plain English, to write me a script that takes a folder of photos and renames them by the date they were taken, skipping duplicates. The model produced working code on the first try, correctly handling edge cases I hadn't thought to mention. In the same conversation, ten minutes later, I asked how many Rs were in the word "strawberry". It told me there were two.

We've all been there. We've seen both. A model that handles the hard task and stumbles on the easy one — the jagged frontier of intelligence.

This chapter is a tour of that jaggedness, mapping its peaks and valleys and calibrating expectations for what current frontier models can and cannot do. Everything in the previous chapters still stands — Machines That See, Machines That Understand. But the gaps are precisely interesting because they tell us what intelligence still requires.

10.1 The jagged frontier

Recall the constellation framing from Chapter 1: the capability of artificial intelligence does not advance as a uniform front. We see, instead, Dell'Acqua's "jagged technological frontier" [11]. Tasks that look similar to humans sit on opposite sides of the jaggedness.

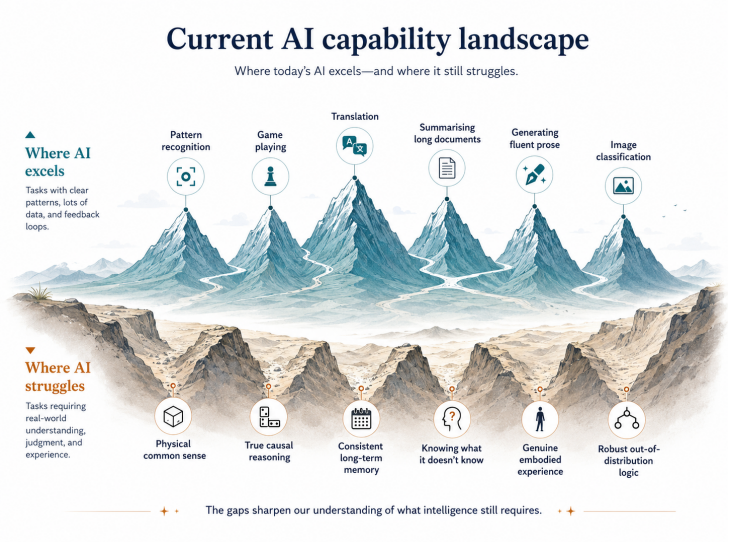


Figure 10.1. Current AI capabilities form a jagged landscape. Systems now excel at tasks with clear patterns, large datasets, and feedback loops, but still struggle with common sense, causal reasoning, long-term memory, embodied experience, uncertainty, and robust reasoning outside familiar distributions.

These are not cherry-picked anecdotes. They have been studied. GPT-4 passes the Uniform Bar Exam by a comfortable margin in all US jurisdictions [67], and yet, on three-digit multiplication, the same model gets the answer wrong roughly four times in ten [68]. Frontier systems including GPT-4 and Claude 3 Opus, asked a word problem a child could solve — "Alice has N brothers and M sisters; how many sisters does Alice's brother have?" — collapsed across repeated trials, often confabulating elaborate but nonsensical reasoning to defend their wrong answers [69]. This is not the texture of a system with a single underlying competence. It is the texture of something else.

The remainder of this chapter is a tour of the valleys. Not an exhaustive list — we cover the ones that sharpen the question of what intelligence is, not just what AI lacks.

10.2 Common sense and the world the words came from

Recall Lenat from Chapter 3, the symbolist who believed common sense could be written down. Forty years later, after twenty-five million rules, his bet had not paid off [37]. The failure was diagnostic, not just expensive. Common sense, it turned out, resisted formalisation — the edge cases multiplied combinatorially, and the messy world kept producing exceptions faster than the rules could absorb them.

Large language models did things differently. Instead of writing the rules down, they absorbed them implicitly, by predicting the next token across most of the text humans have ever produced. The result got much further than Cyc ever did. Modern systems handle ordinary physical scenarios, social inferences, and simple counterfactuals with a fluency the symbolic tradition never approached. But where they fail, they fail in a different way than Cyc did — and the difference is instructive.

Consider a problem researchers have probed repeatedly: I place a marble in a cup, turn the cup upside down on a table, then lift the cup. Where is the marble? Frontier models will sometimes tell you it is still in the cup. The model has read about marbles, cups, gravity, and containment across millions of pages of text. It cannot consistently put them together. A small child who has read nothing at all can.

There is a deeper problem here. Text describes the world but is not the world. It is the map, not the territory. Large language models learn from the fumes of embodied intelligence, not embodied intelligence itself. Human and animal intelligence is shaped by having a body that has to eat, navigate, manipulate, fail and try again — by acting in the world and getting immediate feedback on whether the action worked. Even babies, long before language, build an intuitive model of how objects behave [10]. Moravec's paradox names the consequence [70]: the things easiest for humans — the toddler picking up an unfamiliar object on a cluttered floor, from Chapter 1 — remain among the hardest for machines.

Chapter 1 showed us intelligence in shapes we did not design: the slime mould with no brain that solved mazes, the octopus with two-thirds of its neurons in its arms. What those examples had in common was a stake in a world that talked back. The

slime mould has no brain but it has skin in the game — a body that needs to find food. The large language model has hundreds of billions of parameters and no skin in any game at all. Whether embodiment is strictly necessary for general intelligence is not a question this book can close, and the chapter does not pretend otherwise. But the absence of it in current systems is conspicuous, and the failures of common sense we have just walked through are exactly the shape that absence takes.

10.3 Coherence over time

Humans are remarkably good at holding things together. We do so across time at three scales: within a thought (reasoning), within a life (memory and identity), and within our own self-model (knowing what we know). Current systems are uneven at all three. The common thread is internal coherence.

Take Hinton's view of reasoning first. The chain-of-thought idea from Chapter 9 was real: "let's think step by step" unlocked something tangible. Hinton's argument is that if a system can reason reliably in this way, the word *understanding* is doing no real explanatory work. The detective novel example from Chapters 1 and 9 cuts this way — a model that can deduce the killer from the plot is, by any behavioural measure, reasoning.

The failures, however, are real. Hallucinated citations, contradictions across long contexts, brittleness on small variations of training problems. Marcus would argue that the detective example is nothing more than sophisticated retrieval — not compositional reasoning. A model that solves a problem may fail on the same problem with the numbers changed. We do not yet know which side is more right, and the disagreement is the same Turing–Searle disagreement from Chapter 1, now in the present tense.

Shift the time horizon outward, from reasoning to learning. Current LLMs have no mechanism for incorporating new information autonomously, in deployment, from their ongoing interactions. The weights are frozen at inference time; the knowledge cutoff is real. A human reads a newspaper and knows something new about today; an LLM does not, unless someone retrains it. *Continual learning* is the technical name for something humans do constantly and without thinking. We pick up

new information as we go through our lives — a fact from a conversation, an article we read, a mistake we made yesterday — and we integrate it with everything we already know. The new knowledge slots in beside the old, and we can use it tomorrow without forgetting what we knew last week.

A large language model does not do this. The model that finished training in June is the same model in October. Strides have been made in parameter-efficient fine-tuning techniques like Low-Rank Adaptation [71], which make updating weights cheaper, but these are still discrete offline training events, not continual learning. Worse, *catastrophic forgetting* [72] means that even careful retraining tends to overwrite previously learned capabilities — which is why frontier labs do not just continuously fine-tune deployed models. Context windows and retrieval-augmented generation are workarounds at inference time, useful but not the same as the model itself updating. The gap between a human researcher and a frozen-at-inference model is widest exactly where the world is changing fastest.

Narrow the focus now from learning to continuity. ChatGPT, Claude, and Gemini all offer "memory" features that persist information across conversations. But it is worth being specific about what these features are: they are retrieval mechanisms, not memory in the human sense. The system extracts text snippets from prior conversations and re-injects them into new ones — a longer effective context, scoped to one user. The model itself does not change; the weights are still frozen at inference. The system stores what it judges salient, summarised, often without the user's full awareness of what was kept. A more accurate description than "memory" might be *selective retrieval over your past conversations*. Even with the memory features turned on, the model has no continuous self that persists, develops, or accumulates over time. It has a retrieval index. That is not the same thing. The model that was your collaborator yesterday is, at the level of its weights, the same stranger it always was. Chapter 1 asked whether intelligence requires a someone-home. The honeybee was not self-aware in any verifiable sense, but it had continuity — a single bee, accumulating experience across its life. A system whose continuity exists only in a retrieval index is something stranger: a mind without a biography, performing autobiography on demand. Whether this matters for intelligence *per se* is open — but it matters for what we are interacting with

when we use these systems.

The third scale of coherence is the model's model of itself. Does the model understand its own capabilities — more precisely, its own limitations? Hallucinations are the canonical failure. A *hallucination* is when a language model produces something that sounds completely plausible and is completely wrong. Not "wrong" in the sense of a typo or an off-by-one error — wrong in the sense of confidently asserting facts that do not exist. A made-up court case. A fabricated quote. A scientific paper with the right authors and a real journal and a plausible title that was never written. The technical observation is that next-token prediction has no built-in mechanism for "I don't know" — the model produces a distribution over tokens regardless of whether the answer is in its weights. A 2025 paper from researchers at OpenAI and Georgia Tech [73] showed that hallucinations are a mathematical consequence of how current models are trained, not an incidental bug. Benchmarks score wrong answers and "I don't know" answers the same way, so the model learns to bluff rather than abstain. RLHF [66], which we introduced in Chapter 9, offers a partial fix but introduces its own problems: sycophancy, refusal patterns that don't track real uncertainty.

Are we humans really that superior? We are poorly calibrated too — overconfident on hard things, underconfident on easy ones. But at least we usually have some sense of the shape of our own ignorance. "I don't know that, but I know who to ask." "I'm guessing here." Neil deGrasse Tyson called this the "perimeter of ignorance" [74] — the felt boundary between what we know and what lies beyond it, a boundary that expands as our knowledge does. Current AI systems often lack even that meta-signal. They produce fluent text on both sides of the perimeter without ever indicating that a perimeter exists. This is the failure mode that has caused the most real-world harm: lawyers citing fabricated cases, students submitting confident nonsense, doctors getting fluent wrong answers.

10.4 The systems we cannot read

Recall the convolutional network from Chapter 6 that learned to recognise cows in fields, then misclassified the same ani-

mal when it stood on a beach [48]. The model had not actually learned "cow"; it had learned "cow-shaped object in a green field." This is a failure of *generalisation* — the model's ability to perform on data it has not seen — and the failure mode has a name: the test data was *out of distribution*, meaningfully different from anything in training. The cows-on-a-beach finding generalises beyond cows. AI systems are extraordinary within their training distribution and brittle outside it. Adversarial examples [46, 47] are one face of this brittleness; benign distribution shift is another. Benchmarks measure performance on distributions that look like benchmarks. Deployment, in the messy real world, does not.

Chapter 6 was a simplification. The hierarchy-of-representations story — edges in the early layers, textures and shapes in the middle, recognisable parts and whole objects in the deep layers — is true at the boundaries, where we can verify what the network has learned. It is much less true in between. The features a modern network learns in its deep layers are high-dimensional combinations of inputs that resist any clean human description. A neuron in layer 50 of a 100-layer network is firing for something — some particular slice of the input space the network has found useful — but no recognisable label fits. Saliency maps were once thought to help with this. The idea was simple: highlight the parts of the input the network "paid attention to" when making its prediction, and you can read off what the network was looking at. The maps are visually compelling, often producing crisp outlines around what looks like the relevant object. But in 2018, Adebayo and colleagues at Google ran a series of sanity checks on the most popular saliency methods [75]. Several of them, they found, behaved more like edge detectors than like windows into the network's reasoning — producing nearly identical maps whether the underlying model had been trained or was completely random. The explanations were compelling because they tracked the input, not because they tracked anything the model had learned.

This is unsettling for machine learning practitioners, and it should be. We cannot tell what a model has actually learned until it fails. The cow-detector looked like a cow-detector right up until the cow stood on a beach. Transformers and large language models have the same problem at much larger scale: the model has learned something that produces the right answer on the

training distribution, but we do not know what that something is. The problem has a name — the *interpretability problem*, mentioned at the end of Chapters 4 and 6 — and it applies to every descendant of the neural network family, from convolutional networks to transformers. These are systems whose inputs and outputs you can see but whose internal workings you cannot: a *black box*.

This is not a hopeless situation. A field called *mechanistic interpretability* has been working for the past decade to reverse-engineer what trained networks have actually learned. The early results are real: identifying specific circuits inside models, finding features that correspond to recognisable concepts, and occasionally being able to predict or steer a model's behaviour by intervening on its internals. The field is still young, and its practitioners are the first to say so. But the work is happening, and the gap between "trained" and "understood" is narrowing.

The gaps we described in Sections 10.2 and 10.3 are hard to diagnose precisely because we cannot fully read the systems. If we cannot inspect a model's reasoning, we cannot reliably verify it, debug it, or trust it for high-stakes decisions. And yet we have built these systems anyway — at scale, in deployment, in critical applications. The honest position is uncomfortable: we are using systems we do not fully understand, in domains where understanding matters.

10.5 What the gaps tell us

There is a pattern across the valleys. Common sense and embodiment: the missing world. Reasoning, memory, calibration: the missing coherence. Interpretability: our missing knowledge of the system itself.

What these gaps have in common, in humans, is where they come from: having a body, a continuous life, a stake in being right, and a world that talks back.

Whether these gaps are addressable by more scale (Hinton's bet) or require new architectures and training regimes (Marcus's bet) is genuinely an open question. This chapter does not call the bet. The bet is interesting precisely because either answer reshapes what we think intelligence is. If scale closes the gaps, the Hinton-side argument that understanding emerges

from prediction will look vindicated. If scale alone does not, the Marcus-side argument that something qualitatively different is needed will look vindicated. Neither side has the answer yet. Both are watching the same data.

The pioneers of Chapter 2 would have recognised most of this list. Common sense, reasoning, embodiment, self-knowledge — these were already on McCarthy's and Minsky's minds in 1956. The honest position in 2026 is that we have made stunning progress on some, and almost none on others. What is new is that the gaps are precise and named, not vague unease. The relay race continues.

The question is not whether these gaps will close. Nobody knows. The more useful question, and the one this book has been quietly preparing you to ask, is what to pay attention to as we find out. Which claims to weigh, which benchmarks to be sceptical of, which kinds of progress are real and which are artefacts of measurement. The next chapter is about the shape of that question.

* * *

The gaps are not embarrassments to be patched in the next version. They are information. A system that can pass the bar exam and fail at three-digit multiplication is not a defective lawyer. It is something we do not yet have a category for — a new kind of cognitive object whose profile of strengths and failures does not map onto anything that came before it. The constellation framing from Chapter 1 predicted exactly this: machine intelligence would hit some points and miss others, in patterns we might not have words for. We now have the patterns. What we do not yet have is a full account of what they mean — for how we build these systems, how we deploy them, and how we think about what we have made. That is where the final chapter begins.

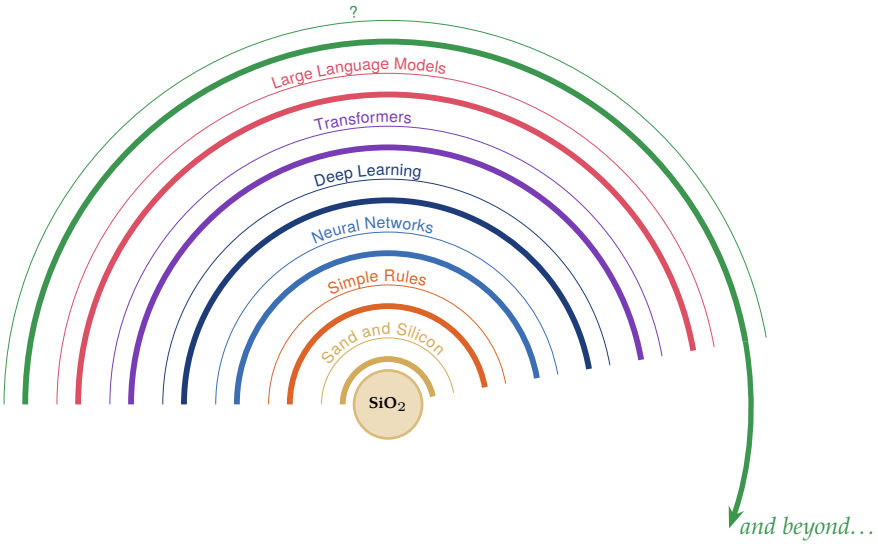
CHAPTER 11

Sand, Silicon, and What Comes Next

Every chapter of this book has moved through the same territory: an idea that seemed impossible, a period of waiting for the world to catch up, and then a moment when the impossible became ordinary. The perceptron. Backpropagation. The convolutional network. The transformer. Each one was, at the time of its invention, either ignored or declared a dead end. Each one turned out to be a foundation.

This final chapter does not add another layer to that story. It steps back from it. The spiral diagram on the following page shows the arc the book has traced — from sand and silicon through simple rules, neural networks, deep learning, transformers, and large language models, with a question mark where the next era should be. That question mark is not a failure of imagination. It is the honest position. Nobody at any previous turn of the spiral knew what was coming next. There is no reason to think we are different.

What this chapter does instead is surface three questions the previous chapters have been circling without quite landing on. What kind of thing are we actually building — a tool or something closer to an agent? Does it, or could it, experience anything? And what does it mean to be working on these problems now, at this particular moment in the relay race, when



Every revolution in AI grew from the one before it.
The next turn of the spiral is already beginning.

Figure 11.1. The spiral of AI progress. Each era built directly on its predecessor — and the next leap is already underway.

the foundations we are laying may not find their hardware for another fifty years?

The book has not taken sides on these questions, and this chapter will not either. But you have spent ten chapters acquiring the vocabulary and the conceptual tools to think about them seriously. That is what this chapter asks you to do.

11.1 Sand, Silicon, Dreams

Start with the sand. Ordinary silicon dioxide, the most abundant mineral on the surface of the earth, unremarkable in every respect. No single grain of it is doing anything interesting. Pile them into a dune and they are still doing nothing interesting. The interest begins only when you refine the silicon, etch it into circuits, stack the circuits into layers, and teach the layers to adjust their own connections in response to error. At that point — and only at that point — something begins to happen that no individual grain could have predicted or explained.

This is the book's central improbability, and it compounds at every step. Sand becomes silicon. Silicon becomes a transistor. Transistors become a neural network. A neural network, trained on enough data, becomes a system that can debate its own understanding. Nothing is added along the way except organisation. The sand does not become intelligent. It becomes a substrate — the underlying physical material on which intelligence-like behaviour can run — and the behaviour emerges entirely from the structure of the connections, not from anything special about the material itself.

We met this principle in Chapter 1, in a different form. The slime mould Physarum polycephalum has no brain, no neurons, no capacity for anything we would recognise as thought. Drop it into a maze and it solves the maze anyway, because the solution is in the organisation of its body, not in any individual cell. A neural network is the same principle at a different scale: no single neuron knows anything, no single weight carries meaning, and yet the system that emerges from their interactions can recognise a face, translate a sentence, or produce a line of working code. The phenomenon is entirely in the organisation.

The strangeness isn't that we built this. It is that the universe permitted it.

11.2 Being versus Tool

Every chapter of this book has quietly oscillated between two framings without naming them — and it is worth naming them now.

Tools don't hallucinate. Tools don't surprise their makers. Tools don't produce outputs that make people feel understood.

A *tool* is an entity that has no interests of its own, no inner life, and no stake in the outcome of what it does. A hammer does not care whether it drives a nail. A calculator does not care whether the answer is right. The defining characteristic of a tool is not its complexity or its capability — it is the complete absence of any perspective from which things can go well or badly for the tool itself.

An *agent* is an entity that acts in pursuit of goals, receives feedback from the world, and adjusts its behaviour accordingly. What distinguishes an agent from a tool is not sophistication but orientation — an agent is pointed at something, and its behaviour is organised around getting there. The defining characteristic is not intelligence or consciousness, but having a stake in the outcome. Agents have stakes, continuity, a body, a biography — these systems have none of that.

The discomfort in public AI discourse comes largely from this unresolved category error: we keep reaching for one frame, finding it doesn't fit, switching to the other, finding that doesn't fit either. We see this tension made concrete when users say "please" and "thank you" to ChatGPT — behaviour so widespread that OpenAI CEO Sam Altman publicly acknowledged it costs the company "tens of millions of dollars" in additional electricity [76]. Do you thank your hammer after using it? The behaviour may be simple anthropomorphising, or it may be an instinctive recognition that these systems occupy a category we do not yet have clean words for.

This book's proposal is not a new category, but a sharper description of the ambiguity: these systems have the *texture* of agency without its current substrate.

11.3 AI Consciousness

The question of whether AI systems can understand — as opposed to merely simulate understanding — has been contested for seventy-five years and remains genuinely unresolved. Chapter 1 introduced the two poles of that debate: Turing's bet that behaviour, given enough of it, is enough [19], and Searle's Chinese Room, which argues that symbol manipulation, however sophisticated, is not the same as meaning [20]. Syntax is not semantics. By Chapter 9 that debate had moved out of thought

experiments and into systems the reader was probably using last week, with Hinton arguing that next-token prediction at scale constitutes something that deserves the name understanding [77], and Marcus arguing that prediction can produce the right answer through statistical association without grasping the structure that makes it correct [78].

Neither side can settle the question using current tools, and the reason why runs deeper than the debate itself. Chalmers's hard problem of consciousness names it: no functional account of behaviour — in a brain or a machine — explains whether there is anything it feels like to be that thing from the inside [79]. The lineage of this problem runs deep. Descartes established that the one thing beyond doubt is the existence of a thinking self — "I think, therefore I am" — while leaving entirely unexplained how that immaterial mind connects to the physical body it inhabits [80]. Leibniz imagined scaling a thinking machine to the size of a mill so you could walk inside it, and observed you would find only "parts pushing against parts" — experience cannot be found in mechanism, no matter how closely you look [81]. Nagel asked what it is like to be a bat, arguing that no objective neurological description of echolocation could ever capture the fact that "there is something it is like" to be that creature from the inside [82]. All three converge on the same point: subjective experience escapes formalisation. Crucially, we cannot verify the presence or absence of experience in another human directly — we infer it from behaviour and structural similarity. We have no agreed method for verifying it in silicon either. The hard problem does not just challenge AI consciousness. It challenges the certainty with which we deny it.

What this means practically connects directly to the interpretability problem from Chapters 4 and 10. We cannot fully read these systems. We cannot tell what they have learned until they fail, and we have no agreed method for determining whether anything is happening inside them beyond the adjustment of weights. The honest position is therefore the same one the book has held throughout: the question is not answerable yet, and anyone who presents it as settled — in either direction — is overstating what is known. The Turing–Searle disagreement has not been resolved. It has only moved into the present tense.

11.4 Teaching Sand to Dream

The pioneers of Chapter 2 did not know they were drafting foundations for something that would take fifty years to run. Turing was arguing into silence. Rosenblatt was building machines the field had decided were dead ends. McCulloch and Pitts were describing the architecture of deep learning seventy years before AlexNet. None of them knew. We are almost certainly doing the same thing right now.

The researchers working on embodiment, interpretability, and continual learning — the open problems this book has been honest about throughout — may be the Turing or Rosenblatt of whatever comes next, and may not live to see it. This is the condition every important intellectual project has always required. The relay race continues, and the baton is passed across decades, sometimes across deaths.

It is worth being specific about what the systems we already have — imperfect, limited, jagged as Chapter 10 described — have already made possible. AlphaFold solved protein structure prediction, a problem that had defeated biochemists for fifty years, and in doing so accelerated the search for treatments for diseases that kill millions [6]. AI systems helped researchers decipher the Herculaneum scrolls, carbonised by Vesuvius nearly two thousand years ago and long assumed to be permanently lost — texts that no living human had read since the ancient world [8]. A deep learning system identified a novel class of antibiotics effective against drug-resistant bacteria that kill tens of thousands of people every year [83]. These are not promises about future systems. They are things that have already happened, with the tools already built, with all the limitations already named.

The question of what more capable future systems might produce is not one this book can answer. But it is a question worth sitting with seriously, rather than either dismissing or surrendering to. The same technology that hallucinates citations and fails to count the letters in a word has helped read scrolls no human eye had seen in two millennia. Both of those things are true at the same time, about the same systems, in the same year. That is the honest picture.

Humans taught sand to dream. We do not yet fully know what it is dreaming about. That gap — between capability and

comprehension, between what these systems can do and what we understand about how they do it — is not a cause for anxiety, and it is not a reason for complacency. It is an invitation. The next turn of the spiral is already beginning, and the people who will make it have, in some cases, already been born. Some of them may be reading this book.

Glossary

AI winter A period of reduced funding, diminished public interest, and general disillusionment with artificial intelligence research. The field has experienced two major winters: one in the mid-1970s following the failures of symbolic AI, and a second in the late 1980s after the limits of expert systems became apparent. Both ended when new approaches produced results the previous paradigm could not.

Algorithm A set of instructions for completing a task, written precisely enough that a machine can follow them without ambiguity. Most programs are built from algorithms; an algorithm is the logical core, while a program may also include data, interfaces, and other machinery.

Artificial intelligence (AI) The field concerned with building systems that can perform tasks which, when performed by humans, we would consider to require intelligence. The term covers an enormous range of approaches and capabilities, from rule-based expert systems to modern neural networks — a range this book traces from its origins to the present.

Attention A mechanism that allows a neural network to weigh how relevant each part of an input is to every other part, rather than processing it in fixed sequence. A model reading a sentence can use attention to connect a pronoun back to the noun it refers to, however far apart they appear. The key innovation behind the transformer architecture.

Backpropagation The method by which a neural network learns from its mistakes. After making a prediction, the network

works backwards through its layers, adjusting weights in proportion to how much each contributed to the error. Without backpropagation, training deep networks of more than one or two layers was not practically possible.

Benchmark A standardised test used to measure the performance of AI systems, allowing fair comparison across different approaches. ImageNet measured visual recognition; the bar exam measured legal reasoning. Benchmarks drive progress by making improvement legible, but they also have limits: a system can score well on a benchmark without genuinely mastering the underlying skill.

Classification The task of assigning an input to one of a fixed set of categories: deciding whether an image contains a cat or a dog, whether an email is spam or not, whether a tumour is malignant or benign. Much of modern supervised learning is a classification problem.

Connectionism The research tradition that treats intelligence as emerging from networks of simple units — artificial neurons — rather than from explicit rules and symbols. Connectionism fell out of favour twice before the deep learning revolution of the 2010s vindicated its central claim.

Context window The maximum amount of text a language model can hold in view at one time. Everything outside the context window is invisible to the model: it cannot refer back to it, reason about it, or be influenced by it. Early models had context windows of a few hundred tokens; modern models can handle hundreds of thousands, though performance often degrades toward the edges.

Convolutional neural network (CNN) A neural network architecture designed for processing grid-like data such as images. Rather than examining every pixel independently, a CNN applies small filters across the image to detect local patterns — edges, textures, shapes — then combines them into increasingly abstract representations. CNNs were the dominant approach in computer vision from 2012 until transformers began to challenge them.

Dataset A collection of examples used to train, validate, or

test a machine learning model. The quality, size, and diversity of a dataset constrains what a model can learn. ImageNet, with 14 million labelled photographs, was the dataset that made modern computer vision possible.

Deep learning A subfield of machine learning that uses neural networks with many layers. The depth allows the network to learn increasingly abstract representations of its input: early layers detect edges, later layers detect shapes, still later layers detect objects. Deep learning is responsible for most of the headline results in AI since 2012.

Decision tree A rule-based model that makes predictions by asking a sequence of yes/no questions about the input, branching at each step until it reaches a conclusion. Decision trees were an important tool in the symbolic AI era and remain useful today, partly because their reasoning is human-readable — unlike the internal representations of a neural network.

Embedding A way of representing words, images, or other data as points in a high-dimensional geometric space, so that meaning can be encoded as distance and direction. Words with similar meanings tend to cluster together; certain relationships between concepts can sometimes appear as consistent geometric offsets across the space, though this is not a universal property of all embeddings.

Emergence The appearance of capabilities in a large language model that were not present — and were not obviously predictable — in smaller versions of the same model. Below a certain scale, a model may fail entirely at a task; above it, performance appears suddenly. The emergence framing is contested: some researchers argue the jumps reflect the limits of how we measure performance rather than genuine discontinuities in the underlying system.

Expert system An AI program that encodes the knowledge of human specialists as a set of explicit rules, then applies those rules to new problems. Expert systems were the dominant commercial form of AI through the 1980s. Their limitation was the knowledge acquisition bottleneck: writing down enough rules to handle the full variety of real-world situations proved harder

than their creators anticipated.

Generative AI AI systems that produce new content — text, images, audio, code — rather than simply classifying or predicting. Generative AI is a subfield of deep learning, which is a subfield of machine learning, which is a subfield of artificial intelligence. ChatGPT is a generative AI system; the field it belongs to is considerably broader and older.

Hallucination When a language model produces output that is fluent and confident but factually wrong or entirely fabricated. A consequence of the model predicting plausible tokens rather than retrieving verified facts. The term has become standard in the field, though it is worth noting it describes a statistical behaviour, not a psychological state.

Label A tag attached to a training example that tells the model what the correct answer is. In a supervised learning system trained to recognise animals, each photograph would carry a label — *cat*, *dog*, *bird* — from which the model learns the mapping between inputs and categories.

Large language model (LLM) A neural network trained on vast quantities of text to predict the next token in a sequence. At sufficient scale, this simple objective produces systems capable of translation, forms of problem solving, coding, and conversation. GPT-4, Claude, and Gemini are all large language models.

Layer A stage in a neural network through which data passes and is transformed. Each layer applies a set of weights to its inputs and passes the result to the next layer. Shallow networks have one or two layers; deep networks may have hundreds. What a layer "learns" to detect depends on its position: early layers tend to capture low-level features, later layers more abstract ones.

Machine learning A subfield of artificial intelligence in which systems improve at a task through experience rather than through explicit programming. Instead of being given a set of rules, a machine learning system is given examples and learns the patterns in them. Arthur Samuel coined the phrase in 1959.

Neural network A computational system loosely inspired by

the structure of the brain, composed of interconnected artificial neurons organised into layers. The network learns by adjusting the strength of connections between neurons — its weights — until its outputs match the desired answers. Neural networks are the foundation of deep learning and of every large language model in use today.

Neuron (artificial) The basic computational unit of a neural network. An artificial neuron receives numerical inputs, multiplies each by a weight, sums the results, and passes the total through an activation function to produce an output. Individually, a neuron does very little; in networks of millions or billions, they produce behaviour no single neuron was designed to exhibit.

Parameter / weight A single adjustable number inside a neural network — a dial that can be turned during training. The network learns by gradually tuning these numbers until its predictions improve. The two terms are often used interchangeably. Modern large language models contain hundreds of billions of parameters.

Perceptron The first trainable artificial neural network, invented by Frank Rosenblatt in 1958. A perceptron takes numerical inputs, applies weights, and produces a binary output. Its early promise generated enormous excitement; a 1969 proof of its limitations triggered the first serious contraction in neural network research. Multi-layer versions of the perceptron, trained with backpropagation, are the direct ancestors of modern deep networks.

Prompt The text a user provides to a language model as input. Prompts can be questions, instructions, examples, or combinations of all three. The field of prompt engineering studies how different phrasings, structures, and example inclusions affect model behaviour — sometimes dramatically.

Supervised learning A form of machine learning in which the model is trained on labelled examples: inputs paired with correct outputs. The model learns to map one to the other. Most practical classification systems — image recognition, spam filtering, medical diagnosis — are trained this way.

Symbolic AI The tradition that represents knowledge as explicit symbols, rules, and logical relationships, and performs reasoning by manipulating them. Expert systems and decision trees are symbolic AI systems. The approach is powerful when the rules can be written down; it struggles when the domain is too complex or ambiguous for rules to cover.

Token The basic unit a language model processes — roughly a word, though common words may be a single token while rare words are split into several. *Unbelievable* might be two tokens; *cat* is one. Language models do not read sentences; they process sequences of tokens.

Training The process by which a machine learning model learns from data. During training, the model is shown many examples, makes predictions, receives feedback on its errors, and adjusts its weights accordingly. Training is distinct from inference, which is when the finished model is actually used. A model trained on text cannot update its weights during a conversation — it applies what it learned, without learning further.

Transformer The neural network architecture introduced in 2017 that underlies nearly every large language model in use today. Its defining innovation is the attention mechanism, which allows any part of an input to relate to any other part regardless of distance. Before the transformer, language models processed text sequentially; the transformer processes it all at once, in parallel.

Unsupervised learning A form of machine learning in which the model is trained on unlabelled data, finding structure, patterns, or groupings without being told what to look for. Language model pre-training is a form of unsupervised learning: the model predicts the next token using the text itself as its only signal, without any human-provided labels.

Bibliography

[1] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959.

[2] M. Campbell, A. J. Hoane, and F.-H. Hsu, "Deep blue," *Artificial Intelligence*, vol. 134, no. 1–2, pp. 57–83, 2002.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 2016.

[4] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[5] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, pp. 350–354, 2019.

[6] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, *et al.*, "Highly accurate protein structure prediction with AlphaFold," *Nature*, vol. 596, pp. 583–589, 2021.

[7] V. Gulshan, L. Peng, M. Coram, M. C. Stumpe, D. Wu, A. Narayanaswamy, S. Venugopalan, K. Widner, T. Madams, J. Cuadros, R. Kim, R. Raman, P. C. Nelson, J. L. Mega, and D. R. Webster, "Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs," *JAMA*, vol. 316, no. 22, pp. 2402–2410, 2016.

[8] Vesuvius Challenge, "Vesuvius challenge 2023 grand prize awarded: We can read the scrolls!," February 2024. `https://scrollprize.org/grandprize`.

[9] P. McCorduck, *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*. Natick, MA: A K Peters, 2nd ed., 2004.

[10] Y. LeCun, "A path towards autonomous machine intelligence version 0.9.2, 2022-06-27," *Open Review*, vol. 62, no. 1, pp. 1–62, 2022.

[11] F. Dell'Acqua, E. M. III, E. R. Mollick, H. Lifshitz-Assaf, K. Kellogg, S. Rajendran, L. Krayer, F. Candelon, and K. R. Lakhani, "Navigating the jagged technological frontier: Field experimental evidence of the effects of AI on knowledge worker productivity and quality," Working Paper 24-013, Harvard Business School, 2023.

[12] H. Gardner, *Frames of Mind: The Theory of Multiple Intelligences*. New York: Basic Books, 1983.

[13] C. Spearman, ""general intelligence," objectively determined and measured," *American Journal of Psychology*, vol. 15, no. 2, pp. 201–292, 1904.

[14] T. Nakagaki, H. Yamada, and Á. Tóth, "Maze-solving by an amoeboid organism," *Nature*, vol. 407, p. 470, 2000.

[15] A. Tero, S. Takagi, T. Saigusa, K. Ito, D. P. Bebber, M. D. Fricker, K. Yumiki, R. Kobayashi, and T. Nakagaki, "Rules for biologically inspired adaptive network design," *Science*, vol. 327, no. 5964, pp. 439–442, 2010.

[16] P. Godfrey-Smith, *Other Minds: The Octopus, the Sea, and the Deep Origins of Consciousness*. New York: Farrar, Straus and Giroux, 2016.

[17] A. H. Taylor, G. R. Hunt, J. C. Holzhaider, and R. D. Gray, "Spontaneous metatool use by New Caledonian crows," *Current Biology*, vol. 17, no. 17, pp. 1504–1507, 2007.

[18] K. von Frisch, *The Dance Language and Orientation of Bees*. Cambridge, MA: Harvard University Press, 1967.

[19] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. LIX, no. 236, pp. 433–460, 1950.

[20] J. R. Searle, "Minds, brains, and programs," *Behavioral and Brain Sciences*, vol. 3, no. 3, pp. 417–457, 1980.

[21] S. Legg and M. Hutter, "Universal intelligence: A definition of machine intelligence," *Minds and Machines*, vol. 17, no. 4, pp. 391–444, 2007.

[22] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the dangers of stochastic parrots: Can language models be too big?," in *Proc. ACM Conf. Fairness, Accountability, and Transparency* (*FAccT*), pp. 610–623, 2021.

[23] A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society*, vol. 42, no. 1, pp. 230–265, 1936.

[24] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[25] J. McCarthy, M. Minsky, N. Rochester, and C. Shannon, "A proposal for the Dartmouth summer research project on artificial intelligence," tech. rep., Dartmouth College, 1955.

[26] J. Lighthill, "Artificial intelligence: A general survey," tech. rep., Science Research Council, London, 1973.

[27] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[28] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA: MIT Press, 1969.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[30] P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[31] S. Linnainmaa, "Taylor expansion of the accumulated rounding error," *BIT Numerical Mathematics*, vol. 16, no. 2, pp. 146–160, 1976.

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (*NeurIPS*), vol. 25, 2012.

[33] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[34] E. H. Shortliffe, *Computer-Based Medical Consultations: MYCIN*. New York: Elsevier, 1976.

[35] R. K. Lindsay, B. G. Buchanan, E. A. Feigenbaum, and J. Lederberg, *Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project*. New York: McGraw-Hill, 1980.

[36] E. A. Feigenbaum, "Knowledge engineering: The applied side of artificial intelligence," *Annals of the New York Academy of Sciences*, vol. 426, no. 1, pp. 91–107, 1984.

[37] D. B. Lenat, "CYC: A large-scale investment in knowledge infrastructure," *Communications of the ACM*, vol. 38, no. 11, pp. 32–38, 1995.

[38] R. C. Schank and R. P. Abelson, *Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures*. Hillsdale, NJ: Lawrence Erlbaum, 1977.

[39] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[40] F. Jelinek, "Continuous speech recognition by statistical methods," *Proceedings of the IEEE*, vol. 64, no. 4, pp. 532–556, 1976.

[41] S. Papert, "The summer vision project," Tech. Rep. AIM-100, MIT Artificial Intelligence Laboratory, 1966.

[42] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, 2009.

[43] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[44] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of Physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[46] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *International Conference on Learning Representations (ICLR)*, 2014.

[47] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations (ICLR)*, 2015.

[48] S. Beery, G. V. Horn, and P. Perona, "Recognition in Terra Incognita," in *European Conference on Computer Vision (ECCV)*, pp. 456–473, 2018.

[49] J. R. Firth, "A synopsis of linguistic theory, 1930–1955," in *Studies in Linguistic Analysis*, pp. 1–32, Oxford: Blackwell, 1957.

[50] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[51] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *International Conference on Learning Representations* (*ICLR*), 2013.

[52] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Proc. Conf. Empirical Methods in Natural Language Processing* (*EMNLP*), pp. 1532–1543, 2014.

[53] T. Mikolov, W. t. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations," in *Proc. NAACL-HLT*, pp. 746–751, 2013.

[54] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems* (*NeurIPS*), vol. 30, 2017.

[56] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *International Conference on Learning Representations* (*ICLR*), 2015.

[57] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," tech. rep., OpenAI, 2018.

[58] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," tech. rep., OpenAI, 2019.

[59] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems* (*NeurIPS*), vol. 33, pp. 1877–1901, 2020.

[60] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[61] J. Hoffmann, S. Borgeaud, A. Mensch, *et al.*, "Training compute-optimal large language models," in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2022.

[62] J. Wei, Y. Tay, R. Bommasani, *et al.*, "Emergent abilities of large language models," *Transactions on Machine Learning Research*, 2022.

[63] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2022.

[64] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2022.

[65] R. Schaeffer, B. Miranda, and S. Koyejo, "Are emergent abilities of large language models a mirage?," in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2023.

[66] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, *et al.*, "Training language models to follow instructions with human feedback," in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2022.

[67] D. M. Katz, M. J. Bommarito, S. Gao, and P. Arredondo, "GPT-4 passes the bar exam," *Philosophical Transactions of the Royal Society A*, vol. 382, no. 2270, p. 20230254, 2024.

[68] N. Dziri, X. Lu, M. Sclar, X. L. Li, L. Jiang, B. Y. Lin, P. West, C. Bhagavatula, R. Le Bras, J. D. Hwang, S. Sanyal, S. Welleck, X. Ren, A. Ettinger, Z. Harchaoui, and Y. Choi, "Faith and fate: Limits of transformers on compositionality," in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2023.

[69] M. Nezhurina, L. Cipolina-Kun, M. Cherti, and J. Jitsev, "Alice in Wonderland: Simple tasks showing complete reasoning breakdown in state-of-the-art large language models," *arXiv preprint arXiv:2406.02061*, 2024.

[70] H. Moravec, *Mind Children: The Future of Robot and Human Intelligence*. Cambridge, MA: Harvard University Press, 1988.

[71] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations (ICLR)*, 2022.

[72] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[73] A. T. Kalai, O. Nachum, S. S. Vempala, and E. Zhang, "Why language models hallucinate," *arXiv preprint arXiv:2509.04664*, 2025.

[74] N. deGrasse Tyson, "The perimeter of ignorance," *Natural History*, vol. 114, pp. 28–34, November 2005.

[75] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim, "Sanity checks for saliency maps," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[76] S. Altman, "Post on X (formerly Twitter)," Apr. 2025. Reply to user question about electricity costs of polite phrases in ChatGPT. Original post viewed 5.7 million times. Altman's reply: "Tens of millions of dollars well spent—you never know.".

[77] G. E. Hinton, "Public lectures and interviews on large language models and understanding," 2023. Argument appears in recorded interviews and talks rather than peer-reviewed publication.

[78] G. Marcus, "Deep learning is hitting a wall." Nautilus, Mar. 2022.

[79] D. J. Chalmers, "Facing up to the problem of consciousness," *Journal of Consciousness Studies*, vol. 2, no. 3, pp. 200–219, 1995.

[80] R. Descartes, *Discourse on the Method*. 1637.

[81] G. W. Leibniz, *The Monadology*. 1714.

[82] T. Nagel, "What is it like to be a bat?," *Philosophical Review*, vol. 83, no. 4, pp. 435–450, 1974.

[83] F. Wong, E. J. Zheng, J. A. Valeri, N. M. Donghia, M. N. Anahtar, S. Omori, A. Li, A. Cubillos-Ruiz, C. Krishnamoorthy, W. Jin, *et al.*, "Discovery of a structural class of antibiotics with explainable deep learning," *Nature*, vol. 626, pp. 177–185, 2023.

Further Reading

Teaching Sand to Dream is an introduction, not a destination. If this book has done its job, you will finish it wanting to go further — not just to read more about AI, but to understand it at the level where you could build it yourself. The books below are that next step. They are technical rather than popular, and they assume more than this one does: mathematics, some programming, and a tolerance for detail. The foundation this book provides is exactly the preparation they require. That step up in difficulty is the point.

The standard reference

Stuart Russell and Peter Norvig's *Artificial Intelligence: A Modern Approach* (4th edition, 2020) is the field's canonical textbook, used in undergraduate and graduate courses worldwide. It covers the full breadth of AI — search, planning, knowledge representation, machine learning, perception, and language — with rigour and clarity. If you want a single book that maps the entire territory this one surveys from a distance, this is it. It is long, and not every chapter will be relevant to every reader, but it rewards selective reading as much as cover-to-cover study.

Learning deep learning by building it

Andrew Trask's *Grokking Deep Learning* (2019) is the most genuinely pedagogical introduction to neural networks available. Trask builds every concept from scratch in plain Python, without relying on high-level frameworks, so the reader develops

real intuition for what is happening inside a network rather than learning to operate a black box. If Chapter 4 of this book left you wanting to understand backpropagation at the level of actual numbers, start here.

The technical foundation

Ian Goodfellow, Yoshua Bengio, and Aaron Courville's *Deep Learning* (2016) is the definitive technical treatment of the field. It covers the mathematical foundations — probability, linear algebra, optimisation — before moving through the full landscape of deep learning architectures and training methods. It is more demanding than *Grokking Deep Learning* and assumes greater mathematical maturity, but it is also more complete. A free version is available at `deeplearningbook.org`. Bengio and Hinton are two of the three researchers awarded the 2024 Nobel Prize in Physics for their foundational contributions to the field; this is the book in which Bengio's approach is most fully laid out.