Chapter 9

AI, Neural Networks and the Connectors

A neural network is an abstract set of layered nodes with connections between the nodes of one layer and the next. The base layer is an entry point for relevant data, middle layers (often called hidden layers) are processing layers that are driven by connection parameters, and the final layer gives an output which provides a solution to a specified problem.

The historical references in the preceding chapters describe concrete models, each designed to solve a single well determined problem. The neural network, by contrast, is a general framework: a flexible architecture that can not only be applied to many different tasks, but can be trained to learn how to perform the task through a learning process. This generality—abstract nodes and weighted connections organized into layers—distinguishes neural networks from their predecessors and makes them a central enabling model for artificial intelligence. Actual learning occurs because the neural network adapts its weights as well as a biasing parameter so that it can draw its own conclusions.

This chapter examines the development of the neural network as the central model for AI. The processes and mathematics of the preceding chapters are implicit in this development, but here we focus on the explicit contributions toward the general framework, as summarized in the following table.

Year / Contributor	**Contribution**	**Relevance to Neural Networks**
1780s / Luigi Galvani	Showed that electricity, not fluids, controlled muscles	Founder of neurophysiology
1847–1854 / George Boole	Developed Boolean algebra	Provided the mathematical framework for logical operations; later applied to circuits and neurons
1897 / Charles Sherrington	Coined the term *synapse*; studied reflex arcs, inhibition/excitation, and integrative properties of neurons	Established neurons as discrete functional units; laid foundation for network connectivity and signal integration
1937 / Claude Shannon	Demonstrated that Boolean algebra can be implemented with switching circuits	Showed that networks of binary elements can compute arbitrary logical functions; foundation for digital computation and network analysis

Year / Contributor	Contribution	Relevance to Neural Networks
1943 / McCulloch and Pitts	Modeled neurons as binary switches that generate patterns	Conceived of the brain as a network of neurons that produces and interprets patterns of activity in response to external stimuli
1949 / Donald Hebb	Proposed Hebbian learning principle: "neurons that fire together, wire together"	Provided a biological mechanism for learning and memory; adaptive networks capable of storing and recognizing patterns
1952 / Hodgkin and Huxley	Developed ion-channel model of the action potential in neurons	Provided biophysical realism for neuron models; foundation for time-dependent, spiking network models
1954/1956 / Marvin Minsky	Showed McCulloch–Pitts networks can implement finite-state machines and simulate computation	Formalized universality in neural networks; connected logic networks to computational theory
1958 / Frank Rosenblatt	Invented the perceptron, a trainable neural network	First trainable pattern-recognition machine; practical implementation of Hebbian-inspired learning in artificial networks
1960 / Bernard Widrow and Marcian Hoff	Developed the stochastic gradient method for linear input output systems,	The method allows for scalability of training and is applicable to nonlinear systems. This is ceentral to the learning process of modern day neural networks
1974 / Paul Werbos	Developed the backpropagation algorithm used to train multilayered neural networks.	Introduces a practical algorithm for implementing stochastic gradient descent to multilayered neural networks.

The history of artificial neural networks is not a straightforward march toward artificial intelligence, but rather an evolutionary process that integrates discoveries from neuroscience, mathematics, and engineering. At its core, the field has drawn inspiration from our growing understanding of biological neural systems, while borrowing methods from logic, computation, and circuit design.

Many of the earliest contributors to what we now call "neural network theory" were not concerned with artificial intelligence at all. Researchers such as Luigi Galvani, Charles Sherrington, and Hodgkin and Huxley were motivated by biological questions about the nervous system, uncovering the electrical properties of neurons and the existence of synaptic connections. Mathematicians such as George Boole provided tools for the formal manipulation of logic. Claude Shannon demonstrated that Boolean logic could be physically realized in switching circuits, laying the foundation for digital computation. Even Warren McCulloch and Walter Pitts, who in 1943 presented neurons as logical threshold units, were primarily interested in understanding brain processes rather than building intelligent machines.

This raises an important interpretive question: when, if ever, did the pursuit of artificial intelligence become an explicit goal rather than an unintended byproduct? Some argue that the work of McCulloch and Pitts already hinted at the possibility that networks of artificial neurons could serve as general-purpose computing machines, edging toward the concept of AI. Others note that even Marvin Minsky's early demonstrations of the computational universality of neural networks were rooted more in formal systems theory than in engineering "intelligent" artifacts.

The irony of this history is perhaps clearest in the case of Frank Rosenblatt. When he introduced the perceptron in 1958, he described it primarily as a tool for exploring brain-like learning processes. Yet it was soon reframed as a machine-learning device capable of pattern recognition.

Progress slowed in the 1960s, when the perceptron's limitations—especially its inability to solve certain kinds of problems—became clear. A breakthrough came in 1974, when Paul Werbos showed how multilayer networks could improve themselves by comparing their outputs to the correct answers and adjusting the connection parameters step by step. This process, later named *backpropagation*, allows the network to learn through error corrections that promote the training of complex networks which recognize patterns far beyond the reach of the original perceptron. In this sense, Werbos provided neural networks with their first practical learning strategy.

The development of artificial neural networks cannot be understood as the work of a single discipline or the pursuit of a single goal. Rather, it is the outcome of an interdisciplinary evolution: biological discoveries that inspired mathematical models, mathematical models that inspired engineering implementations, and engineering implementations that were reframed as steps toward AI. At each stage, contributors had different motivations and expectations, and in many cases they would have been surprised to learn that their efforts ultimately paved the way for today's advances in artificial intelligence.

Below, we provide details of this story by specifying the works of contributors as given in Table 8.1. A complete history would also consider hardware developments without which there would be no neural networks. As this book focuses on the data science aspect of neural networks, the hardware description and its relation to developments in neural network design are brief. Following the historical presentation is a case study where we apply the data science program from Chapter 2 to the configuration of a neural network that solves a specified problem.

9.1 Serendipity, Luigi Galvani (1737–1798)

In the late 18th century, electricity was a subject of intense curiosity and experimentation. Scholars were exploring its many manifestations—static sparks generated from friction inspiring Benjamin Franklin's famous kite experiment and Leyden jars that stored charges. The effects of electricity were largely seen as curiosities or demonstrations, with little connection to biological processes. Into this world stepped Luigi Galvani, an Italian physician and anatomist, who was captivated by the mysteries of muscle movement. Remarkably, Galvani considered that electricity might play a direct role in muscular contraction—a bold insight at a time when the mechanisms of the nervous system were unknown.

Galvani was a meticulous observer and tireless experimenter. He reportedly noticed that dissected frog legs occasionally twitched during routine dissections, sometimes associated with contact with metals or nearby sparks. Serendipity sparked these early observations, but Galvani's curiosity drove him to investigate systematically. Using frog legs with exposed nerves, he applied electrical charges in various ways, including contact with metals and static electricity. Consistently, he observed muscular contractions in response to these stimuli, demonstrating that electrical forces could directly provoke movement.

Galvani interpreted these results as evidence of "animal electricity," proposing that living tissue itself generates electrical signals that drive muscle activity. This revolutionary insight—electricity as a mediator of biological signals—would inspire future neuroscientists, including Charles Sherrington a century later and Hodgkin and Huxley nearly two centuries later.

His work also sparked one of the most famous scientific debates of the era. Alessandro Volta argued that the observed effects arose not from the animal tissue but from the metals themselves. This disagreement ultimately led Volta to invent the voltaic pile, the world's first battery, showing how Galvani's experiments

catalyzed advances far beyond their original biological context.

Galvani's dedication to careful observation and methodical experimentation is legendary. He conducted hundreds of frog-leg experiments, documenting even the slightest twitch with remarkable precision. His work earned him widespread recognition during his lifetime, including election to scientific societies and correspondence with leading scientists of the era. Anecdotes of his laboratory often emphasize the "Eureka!" moments that could arise from his careful attention to seemingly trivial phenomena, highlighting the blend of serendipity, curiosity, and rigor that characterized his approach.

Through these investigations, Galvani laid the groundwork for understanding the nervous system as a network of electrically excitable units. His experiments and interpretations established a conceptual bridge between biological function and the later abstraction of neurons as discrete on/off units—a conceptual foundation that would, centuries later, inspire the development of artificial neural networks.

9.2 The Logician, George Boole (1815–1864)

George Boole was a largely self-taught English mathematician whose work laid the foundations for one of the most important conceptual tools in modern computing: Boolean algebra. Unlike many of his contemporaries, who were drawn to grand philosophical systems, Boole focused on the mechanics of thought itself. He asked: if the mind reasons according to logical rules, could those rules be expressed in the precise language of mathematics?

The result was Boolean algebra, a system where logical statements are reduced to simple variables that take only two values: 0 and 1. In this scheme, 0 represents *false* and 1 represents *true*. With only these two symbols, Boole showed how complex chains of reasoning could be represented in algebraic form. This was a radical simplification—stripping logic down to its bare bones—and yet it turned out to be enormously powerful.

At the heart of Boolean algebra are three fundamental operations: **AND**, **OR**, and **NOT**. These operations can be written using algebraic symbols:

- $A \cdot B$ (**AND**): true only if both A and B are true.
- $A + B$ (**OR**): true if at least one of A or B is true.
- $\overline{A}$ (**NOT**): true when A is false.

Boole demonstrated that these simple expressions could encode all possible patterns of logical reasoning. To illustrate, consider the following truth table:

A	B	$A \cdot B$	$A + B$	$\overline{A}$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

This table shows every possible combination of truth values for two statements, A and B, and the results of applying Boole's three logical operators. What is remarkable is how the content of the statements is irrelevant: Boolean algebra cares only about whether each proposition is true or false. To make this more concrete, we can let A and B represent claims from the early history of electricity and physiology. Each row of the truth table can then be read against specific statements:

A	B
Volta proposed that electricity actuates muscular contraction.	Galvani proposed that animal electricity drives muscle contraction.
Volta proposed that electricity actuates muscular contraction.	Volta developed the world's first battery.
Galvani proposed that animal electricity drives muscle contraction.	Galvani developed the world's first battery.
Galvani proposed that animal electricity drives muscle contraction.	Volta developed the world's first battery.

Reading across the two tables together, one sees how Boolean algebra provides a framework for reasoning about propositions. In the first row, where both A and B are 0, neither Volta's nor Galvani's claims are accepted as true. In the last row, where both are 1, the claims of both scientists are true, and the logical operations reflect that joint truth. Whether the subject is electricity, mathematics, or any other domain of knowledge, the structure of reasoning is captured by the same algebraic rules. This leap—from words and ideas to numbers and operations—was Boole's great gift.

Boole's intellectual achievements are made even more striking when one considers his background. Born in 1815 in Lincoln, England, he was the son of a poor shoemaker. His father, though unsuccessful in trade, had a deep amateur interest in science and instruments, which he passed on to George. Because the family could not afford university, Boole left formal schooling at sixteen and began working to support his parents and siblings.

Despite these hardships, Boole pursued knowledge relentlessly. He taught himself Latin, Greek, French, German, and Italian so that he could read advanced mathematics in the original texts of Newton, Laplace, and Lagrange. By nineteen, he had opened a small school of his own, teaching by day and studying by night. His papers on differential equations and the calculus of variations attracted attention from established mathematicians, who were astonished at the quality of work from someone without a university degree.

In 1849, Boole's career took a decisive turn when he was appointed the first professor of mathematics at the newly founded Queen's College in Cork, Ireland. There, he wrote his groundbreaking books: *The Mathematical Analysis of Logic* (1847) and *An Investigation of the Laws of Thought* (1854), in which he formalized the algebra of logic.

Tragically, Boole's life was cut short in 1864, at the age of forty-nine, after he caught a severe cold and was treated with the misguided medical practices of the day. Yet in his brief life, he reshaped the landscape of logic and laid the foundations for modern computation.

Boole believed that the processes of reasoning could be represented with the precision of mathematics, a bold idea in his time. His work was initially underappreciated, but its simplicity camouflaged enormous power. Nearly a century later, Claude Shannon recognized that Boolean algebra provided the perfect framework for designing electronic circuits, where 0 and 1 could be represented by low and high voltages and the wiring of switches would implement Boolean operations directly.

In that sense, Boole is one of the intellectual ancestors of modern computing and, by extension, of artificial neural networks. At their simplest, artificial neurons still reduce to binary decisions: fire (1) or remain inactive (0). Boole's algebra gave mathematics a language to describe such decisions, and without it, the digital logic that underpins computers and neural networks could not exist.

9.3 Charles Sherrington (1857-1952): Connecting the Neurons

Charles Scott Sherrington entered the world under uncertain circumstances in 1857. His father died before his birth, leaving his mother to remarry a physician, Caleb Rose, who became the guiding figure of Sherrington's upbringing. From this fragile beginning — a child born into loss and reshaped family ties — Sherrington might have seemed destined for a modest life shadowed by early tragedy. Yet his trajectory was anything but modest. He grew into a man of remarkable physical vigor and extraordinary intellectual stamina, qualities that allowed him to sustain decades of exacting experimental work on the nervous system.

In his laboratory, Sherrington set out to uncover the hidden architecture of the brain. At the time, the prevailing debate in neuroscience was whether the nervous system formed a continuous reticulum or whether it consisted of distinct cellular units. Sherrington's investigations leaned on delicate experiments with both structure and function. He applied chemical dyes to thin slices of brain tissue, allowing neurons to stand out under the microscope with striking clarity. Through this staining technique, he could observe that neurons were not fused into one mass but instead connected across tiny gaps. He gave the name "synapse" to these junctions, recognizing them as the fundamental sites where communication between neurons took place.

Sherrington complemented these microscopic insights with ingenious physiological experiments. By stimulating sensory nerves in animals and recording the resulting muscle responses, he revealed the logic of reflex arcs. He showed that signals converged from multiple sensory inputs onto motor neurons, and that inhibitory as well as excitatory effects determined the final outcome. The brain, in his conception, was not a tangle of undifferentiated fibers but a coordinated network of discrete cells whose interactions gave rise to behavior.

This vision — neurons as individual units bound together into a dynamic network — became the cornerstone of modern neuroscience. Sherrington's intellectual labors were crowned in 1932 with the Nobel Prize in Physiology or Medicine, shared with Edgar Adrian, for their discoveries concerning the functions of neurons. Even after that triumph, Sherrington's mind did not rest. He continued to write and reflect well into his nineties, a striking contrast to his precarious start in life. His longevity — both of body and of intellect — embodied the very resilience and interconnection he revealed in the brain's neural networks.

Sherrington's work also carried significance far beyond physiology. His careful demonstration that the brain is a network of neurons, each combining signals through excitation and inhibition, provided a framework that later thinkers could translate into formal systems. Claude Shannon, working from the perspective of communication and logic, would describe how electrical switches could encode and transmit information. Warren McCulloch and Walter Pitts, directly inspired by Sherrington's conception of the neuron, went further by proposing a mathematical model: each neuron could be represented as a logical unit, with excitatory and inhibitory connections corresponding to weighted inputs, and with a threshold governing whether the neuron fired. In doing so, McCulloch and Pitts gave symbolic form to Sherrington's experimental observations, recasting his living networks of nerve cells as abstract networks of logical operators.

Thus, Sherrington's discovery that the brain is a network of neurons stands as the biological precursor to both Shannon's theory of communication and McCulloch and Pitts' theory of neural networks. Together, these lines of thought reveal a striking continuity: the nervous system, the circuit, and the logical machine can all be understood through the same unifying language of networks.

9.4 Claude Shannon (1916–2001) and his Intellectual Empire

Claude Shannon grew up in Gaylord, Michigan, where from an early age he was known for his fascination with gadgets and problem-solving. As a boy he built model planes, a radio-controlled boat, and even a telegraph line strung between his house and a friend's, using barbed wire as the transmission medium. This love of

tinkering foreshadowed the blend of abstract reasoning and practical ingenuity that would later define his career. Shannon studied electrical engineering and mathematics at the University of Michigan, graduating in 1936, before moving to MIT for graduate work. There he found himself surrounded by the mechanical complexity of telephone relays and early calculating machines—and he saw something no one else had.

In his 1937 master's thesis at MIT, Shannon made a conceptual leap that forever changed engineering and science. He demonstrated that the on–off behavior of electrical relays could be described precisely by the algebra of George Boole. A relay, long used in telephone exchanges, could be either open (0) or closed (1). By wiring relays in different configurations, one could realize the fundamental logical operators **AND**, **OR**, and **NOT**. His thesis, often called the most influential master's thesis ever written, showed that any logical or mathematical expression could be translated into a physical switching circuit. With this insight, Shannon fused symbolic logic with electrical engineering, effectively creating the blueprint for digital computation.

The **AND operator**, for example, can be implemented with two switches placed in series, controlling a circuit that lights a lamp. Current flows (the lamp lights) only if both switches are closed (both inputs are 1). This simple electrical circuit is shown below:

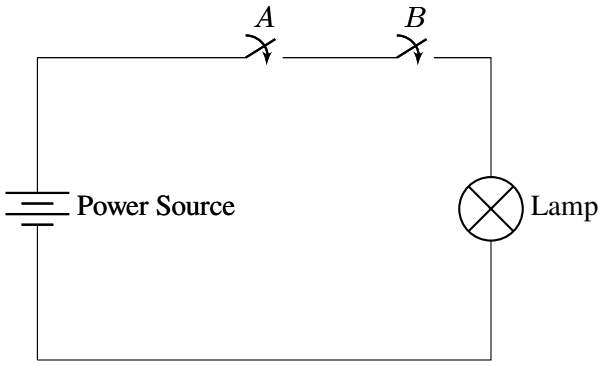


Figure 9.1: Shannon's AND circuit

Shannon extended this approach to other Boolean operations, the **OR operator** (switches in parallel) and the **NOT operator** (a relay contact that opens when the coil is energized). He then showed that more complex combinations of these primitive circuits could implement arithmetic operations, such as binary addition.

For example, the **half-adder**—a circuit that adds two binary digits—can be defined in Boolean terms as:

$$\text{Sum} = A \oplus B \qquad \text{Carry} = A \cdot B$$

where $\oplus$ denotes the exclusive OR. This showed that the fundamental operations of arithmetic could be constructed entirely out of logical building blocks. Shannon had revealed that Boolean algebra was not just an abstract mathematical game but the operating system of electrical circuits.

What makes Shannon's work so remarkable is that the same structural logic is still used in modern microelectronics. Today, the relays of the 1930s have been replaced by transistors, which act as solid-state switches. A CMOS AND gate, for instance, is built from transistors arranged in the same series/parallel logic that Shannon described with relays. The scale has changed—from bulky electromechanical relays to billions of nanoscale transistors on a microchip—but the underlying logic is identical. Shannon did not merely solve a problem in circuit design; he revealed the deep isomorphism between logic and electronics that drives every modern digital device. It was a necessary precursor to the modern computer.

Shannon continued to propel the body of human knowledge. In 1948, he published *A Mathematical Theory of Communication*, founding the discipline of information theory. There, he introduced concepts such as *entropy*, *redundancy*, and *channel capacity*, creating a rigorous framework for understanding communication over noisy channels. Information theory became a universal language, with applications in a remarkable range of fields, including:

- **Telecommunications:** design of coding schemes and error correction.
- **Computer science:** data compression, algorithms, and cryptography.
- **Neuroscience:** modeling neural signaling and information flow in the brain.
- **Genetics:** understanding DNA as a coded information system.
- **Linguistics:** quantitative analysis of language and redundancy.
- **Physics:** statistical mechanics and the thermodynamics of information.
- **Artificial intelligence:** guiding machine learning and neural network theory.

Shannon cultivated an intellectual empire. One would be hard pressed to find a Nobel laureate whose works have been as far reaching as Shannon's. Alongside Newton, Shannon can be counted among the very few thinkers whose work has transformed nearly every branch of science and technology.

Despite his serious contributions, Shannon never lost his playful spirit. At Bell Labs, he built whimsical machines: a mechanical mouse named *Theseus* that could learn to solve a maze, a juggling robot, and even a unicycle he rode through the corridors. These creations reflected his conviction that curiosity and play were inseparable from creativity.

In sum, Shannon's master's thesis and his later work on information theory together established the theoretical and practical foundations of the digital age. Every time a microprocessor computes, a phone transmits data, or a neural network trains on vast datasets, it does so in the shadow of Shannon's revelation: that logic and electricity are two sides of the same coin.

9.5 Biological Switches: Warren McCulloch and Walter Pitts (1943)

Warren McCulloch (1898–1969) was a neurophysiologist and psychiatrist, while Walter Pitts (1923–1969) was a teenage mathematical prodigy with a deep interest in logic and computation. McCulloch was fascinated by the brain's capacity to process information, while Pitts was captivated by formal systems and logical reasoning. Their collaboration began at the University of Chicago, where they sought to combine insights from neurophysiology with the rigor of mathematics.

At the time, understanding of neurons was limited to anatomical observations and qualitative physiological studies. McCulloch and Pitts were motivated by a central question: *can the operations of the brain be modeled mathematically?* In particular, could networks of simple units—neurons—produce complex behavior such as pattern recognition and logical reasoning? Sherrington's work provided the narrative for the endeavor. McCulloch and Pitts would provide the mathematical formalism.

In 1943, they published their landmark paper, "A Logical Calculus of the Ideas Immanent in Nervous Activity," introducing what is now recognized as the first formal model of a neural network. In their model, each neuron is a **binary threshold unit**, functioning as a biological switch. The brain is then seen as a network of such switches, with synaptic connections forming circuits.

The model places the brain directly within the scope of Shannon's electric circuits. Whereas Shannon used physical switches in relays, McCulloch and Pitts described abstract biological switches. Nevertheless, the logical principles are the same: both systems can implement Boolean logic and thus perform computation.

Indeed, following Shannon, McCulloch and Pitts presented neuronal circuitry capable of executing Boolean operations. Below is an illustration of a McCulloch–Pitts neural circuit that implements the AND operation. By changing the threshold one the configuration implements the OR operation.

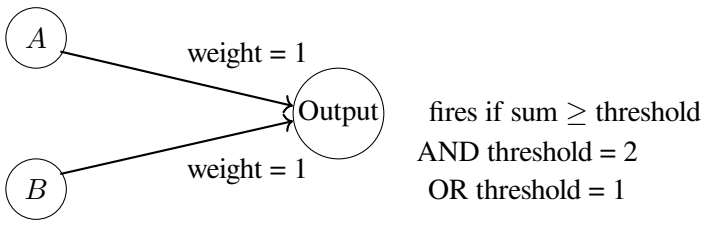


Figure 9.2: McCulloch–Pitts neuron for AND and OR operations.

In this figure, A and B are binary signals (0 or 1), coming from external stimuli or other neurons. The signals are multiplied by their connection's corresponding weight to become inputs for the output neuron. For the AND operation, the output neuron activates if the sum of inputs reaches the threshold of 2, producing an output of 1. Lowering the threshold to 1 results in an OR operation.

More generally, each neuron has an arbitrary number of incoming signals with corresponding weights and an activation threshold that is a positive integer. Each neuron sums its weighted inputs and compares the result to its threshold to determine whether or not it activates; summed inputs beneath the threshold are inactive, otherwise the neuron activates.

In McCulloch–Pitts' original model, weights were either +1 (excitatory) or -1 (inhibitory). For expository purposes, one can imagine a third conceptual value, $-\infty$, which ensures the neuron never fires. Excitatory inputs push the sum toward the threshold, while inhibitory inputs prevent firing. The AND operation uses only excitatory weights (+1), a NOT operation uses an inhibitory input (-1), and $-\infty$ can be viewed as a conceptual weight that inactivates a neuron regardless of other inputs.

After establishing the behavior of a single neuron, McCulloch and Pitts explored networks of neurons. Their key insight was that by wiring many neurons together, one could build circuits that generate patterns of activation in response to both external and internal stimuli. The brain, in this view, both generates and interprets patterns.

The number of possible activation patterns across a large network is staggering: 2^N, where N is the number of neurons. The human brain is estimated to have roughly 86 billion neurons. In principle, such a system could represent a number with more than 25 billion digits—so large that it defies conventional naming. To illustrate scale, if this number were printed in 12-point type on a ribbon, the ribbon would stretch far enough to encircle the Earth more than once.

A single McCulloch–Pitts neuron is simple: it sums its inputs and compares the total to a threshold. Its operation can be described using elementary arithmetic, making the model accessible to an elementary school student. Yet networks of these simple units can produce complexity on an unimaginable scale.

McCulloch and Pitts' work inspired further research in both physiology and artificial intelligence. For neuroscience, their model proposed a testable hypothesis: is the neuron essentially a switch? For AI, their ideas opened the door to designing neural networks with simple computational units that could later be realized on digital computers.

9.6 Donald Hebb: But how do we learn?

Donald Olding Hebb (1904–1985) entered the study of the mind at a moment when both physiology and psychology seemed to be circling around the same unanswered question: how does the brain change when it learns? Hebb was trained in psychology at McGill University and later worked under Wilder Penfield at the Montreal Neurological Institute, where neurosurgical interventions gave him firsthand exposure to the effects of brain lesions on behavior. Unlike many of his contemporaries, Hebb was not content with describing learning in purely behavioral terms, as behaviorists such as B. F. Skinner did, nor did he settle for only anatomical descriptions of neurons. His motivation lay in bridging the two: to explain learning as a biological process grounded in the physiology of the brain but expressed as changes in behavior.

In 1949, Hebb published his landmark book *The Organization of Behavior*, which became the cornerstone of modern neuropsychology. In this work, he advanced what has since become known as the Hebbian theory of learning. His central idea was simple yet profound: when one neuron repeatedly helps to fire another, the connection between them grows stronger. In his own concise formulation, "cells that fire together, wire together." Hebb reasoned that such strengthening of connections provided a physiological mechanism for learning, whereby repeated co-activation of neurons would produce lasting changes in the structure and function of the neural network.

The model proposed by Hebb was not an abstract mathematical construction like that of McCulloch and Pitts. Instead, it was a biological hypothesis: synaptic efficacy changes with experience. Still, his work presupposed the availability of a formal network framework such as theirs. Once neurons could be represented mathematically as nodes connected by weighted links, Hebb's rule supplied a principle for how those weights might evolve over time with repeated stimulation. In other words, McCulloch and Pitts described how neurons could compute; Hebb described how neurons could learn.

Hebb extended his model beyond the single synapse to propose the idea of cell assemblies — groups of neurons that, through repeated co-activation, become wired into functional units. When such an assembly was activated, it could evoke the entire network of activity, creating a stable representation of a perception, an idea, or a memory. Overlapping assemblies allowed complex associations to be formed, giving the brain the flexibility to integrate new experiences with old.

This vision was motivated not only by laboratory work but also by Hebb's observations of patients with brain injuries. He noted that deficits in perception or cognition were often not absolute; patients could sometimes recover functions, suggesting that new circuits could form or strengthen to compensate. Such plasticity, he believed, was evidence for a learning mechanism at the neural level.

Hebb's proposal of synaptic modification as the foundation of learning represented a major intellectual leap. It shifted the explanation of learning from vague mental faculties or behavioral conditioning to concrete biological changes in neural circuits. Later discoveries of long-term potentiation (LTP) in the hippocampus in the 1970s would provide striking experimental confirmation of Hebb's hypothesis.

Thus, Hebb provided the missing piece of the story that began with Sherrington, Shannon, and McCulloch and Pitts. Sherrington showed that the brain is a network of neurons; Shannon and McCulloch and Pitts showed that such networks can compute; and Hebb showed how such a network could adapt and learn from experience. As with Sherrington, Hebbs' work was narrative. Modern AI would have to await a mathematical formalism that would introduce learning in the manner that Hebbs describes.

9.7 It's for Real: Hodgkin and Huxley (1949)

By 1949, the state of neuroscience was the following. Sherrington had provided a descriptive account of the brain as a vast network of neurons connected by synapses. McCulloch and Pitts gave this network mathematical structure, showing how interconnected "neurons" could perform logic and generate patterns of thought. It made for a compelling story — but was there reality behind it? Were neurons truly switches that could carry out such operations, or was this only a clever metaphor?

The answer came from two British physiologists, Alan Hodgkin and Andrew Huxley, who undertook a series of bold experiments to uncover how neurons actually transmit signals. To do this, they turned to the giant axon of the squid, one of the largest nerve fibers in the animal kingdom. Its size — nearly a millimeter in diameter — allowed them to insert delicate electrodes into the axon itself, making direct measurements of the electrical potential inside the cell possible for the first time.

A key innovation was the voltage clamp technique, which they developed to hold the axon's membrane at a fixed voltage while they measured the currents flowing through it. By controlling the voltage like a laboratory knob, they could separate and quantify the contributions of different ions (sodium and potassium) to the action potential. This gave them unprecedented experimental precision: not just watching the nerve fire, but dissecting the dynamics of how it fired.

Their results were revolutionary. They showed that the resting potential of the neuron is maintained by a delicate balance of ions: sodium ions (Na^+) are kept outside, potassium ions (K^+) inside, creating a voltage difference across the cell membrane much like the terminals of a battery. When the neuron fires, voltage-sensitive ion channels open: sodium rushes inward, briefly making the inside positive, then potassium flows outward to restore the balance. This rapid flip in voltage — the action potential — propagates along the axon like a wave, carrying the nerve signal forward.

Hodgkin and Huxley did not stop at description. They constructed a mathematical model, a set of nonlinear differential equations, that reproduced the action potential in quantitative detail. Their model showed that the neuron is not just like a switch — it is a switch, implemented by nature through ion channels and voltage differences. For this, they were awarded the 1963 Nobel Prize in Physiology or Medicine (shared with John Eccles, who studied synaptic transmission).

McCulloch and Pitts (1943) had proposed an abstract neuron: it sums its inputs, applies weights (which could be positive or negative), and fires if the total exceeds a threshold. This logical neuron could implement Boolean operations like AND, OR, and NOT, and thus serve as the building block of a computational network.

Hodgkin and Huxley, by contrast, discovered that real neurons operate using biophysical mechanisms, not algebraic rules. Their model allowed only positive ion currents: sodium flowing in, potassium flowing out. There is no direct equivalent of a "negative weight" in the biophysics of a single axon. However, when one considers networks of neurons and especially inhibitory synapses, negative weights reappear in biological form: inhibitory neurons release transmitters that make the postsynaptic cell less likely to fire, effectively subtracting from the input.

Thus, the Hodgkin–Huxley neuron is more constrained than the McCulloch–Pitts abstraction if taken alone. But the essential requirement — a threshold-based device capable of switching on and off — is present. And once both excitatory (positive) and inhibitory (negative) influences are allowed, the biological network has all the machinery needed to implement the same Boolean functions envisioned by Shannon, McCulloch, and Pitts.

In sum, Hodgkin and Huxley supplied the missing piece of the story. Sherrington had described the brain as a network. McCulloch and Pitts had shown that such a network could compute. Hodgkin and Huxley proved

that neurons are actual biological switches, powered by the chemistry of ions and membranes. With their work, the brain could finally be understood as a biochemically organized computational system.

While the computational aspect of neural processing had been placed on a firm foundation, Hebbian learning through adaptable synaptic connections remained purely conjectural. Nevertheless Hebbian learning was useful. More to the point, no other proposal for learning had been presented. So the theoreticians did what theoreticians do, they marched with it just to see where it would take them.

9.8 Pushing Further Through the Maze: Marvin Minsky (1951)

Marvin Minsky's early life and education laid a strong foundation for his pioneering work in artificial intelligence. Born in 1927 in New York City, Minsky demonstrated an early aptitude for mathematics and science. His service in the U.S. Navy during World War II exposed him to radar technology and electronics, fostering a hands-on understanding of complex systems. After completing his undergraduate studies at Harvard University, Minsky pursued a Ph.D. in mathematics at Princeton University, where he began to explore the intersection of neural networks and learning systems.

In 1951, while a graduate student, Minsky developed the Stochastic Neural Analog Reinforcement Calculator (SNARC), a machine designed to simulate learning processes through a network of interconnected neurons. Inspired by the McCulloch and Pitts' work, Minsky conceived of the project during his undergraduate days at Harvard. Minsky sought to create the physical machinery that would transition McCulloch and Pitts' work from a theoretical proposition to an actual electro-mechanical neural network. That end goal in its own was quite audacious, but Minsky wanted the device to do more than execute preprogrammed circuits that perform a specified task. Minsky also wanted to implement a learning procedure; the network would teach itself to perform its task by adjusting its circuity. Minsky, along with Dean Edmonds, a fellow graduate student who contributed practical knowledge of electronics, constructed SNARC.

Comporting with the work of McCulloch and Pitts, SNARC's basic building blocks were "neurons" implemented with vacuum tubes, each acting as a simple on-off switch. Wiring between the neuronal units acted as synaptic connections. Potentiometers that controlled voltage differentials across the wirings acted as weights.

To train SNARC, Minsky blended two approaches, reinforcement learning along with Hebb's 'neurons that fire together, wire together' learning mechanism. B. F. Skinner, a behavioral psychologist was an advocate of reinforcement learning. In later interviews Minsky relates Skinners influence upon him during his undergraduate years at Harvard.

Reward and punishment mechanisms are central to Skinner's reinforcement learning. When a living organism responds appropriately to an environmental challenge it receives a reward. Alternatively, an inappropriate response entails a punishment.

SNARC executed reinforcement learning through a Hebbian mechanism that is most easily understood within the context of the task that Minsky expected SNARC to perform. For the task, Minsky looked to Shannon's Theseus. Recall that Shannon designed and built an artificial mouse, named Theseus, that learned to navigate its way through a maze. In a simulated environment, SNARC would act as a mouse whose objective was to work its way to the maze's exit.

The patterns of the neuronal firings indicated the directional choice of SNARC. Random motion through a stochastic simulator guided SNARCS unlearned directional choices [1]. When the stochastic choice guided SNARC toward the exit, SNARC received a Hebbian reward by adjusting the potentiometers on the activated

[1] Random processes are called stochastic which explains the the S in SNARC

neurons strengthening the signaling between those neurons. Literally neurons that fired together were wired together. Alternatively, when the stochastic choice led SNARC astray, the punishment was an adjustment in the potentiometers that weakened the signaling between the neurons. Capacitors across the neuron circuits acted as short-term memory elements, recording which neurons had fired together during a trial. A mechanical process then adjusted the potentiometers associated with the charged capacitors.

Minsky describes successes as well as failures. By successfully navigating the maze, SNARC demonstrated that neural networks were able to learn how to perform tasks using reinforcement learning with Hebbian rewards. In the aforementioned latter day interview, Minsky relates limitations of the learning process in that SNARC could not perform other tasks included in Minsky's experiments. The interview does not specify these tasks.

After his experimentation with SNARC Minsky turned to more theoretical issues. In his 1954 dissertation at Princeton, under the guidance of John Tukey, Minsky provides theoretical foundations for understanding network stability, feedback, and pattern representation. Central to his dissertation was the question of whether or not neural networks could perform higher level logical operations.

The issue had been taken up by Shannon who demonstrated that circuits of boolean operators could perform higher level logical operations as well as computations. Nevertheless, the class of operations that such a network could perform was unknown. Minsky stepped in; he did this through a device known as a finite state machine.

An abstract description of a finite state machine is a system of N objects each having its own set of states along with a set of inputs. The objects transition from one state to another in accordance with inputs and a transition rule for each input. The finite state machine actuates if/then statements in the following sense. For each object, if it is in a given state, say A, and it receives an input, say I, then it transitions to state B. (It is possible that state B is the same as state A.)

Finite state machines can implement complex logic. One can describe a traffic light as a finite state machine with timers providing inputs to the red, yellow, and green signals. There is no single correct way to implement a traffic light as a finite-state machine; multiple designs can produce the same observable behavior. This is a general statement; different finite state machines may implement logic with the same outcomes.

By definition,a McCulloch-Pitts neural network is a finite state machine. Each neuron is an object with two states, active or inactive. Through the summing and threshold mechanism, inputs determine the transitions of the neurons state. In the case of the neural network, the state of one neuron may act as an input to another neuron. Minsky showed that for any given finite state machine, a McCulloch-Pitts neural network could be designed to implement its logic. This establishes an equivalence between McCulloch-Pitts neural networks and finite state machines implying that all the capabilities and limitations of finite state machines apply to McCulloch-Pitts neural networks.

In the summer of 1956, Minsky participated in the seminal Dartmouth Summer Research Project on Artificial Intelligence, organized by John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon. This gathering, now widely regarded as the official birth of AI, brought together leading thinkers to explore the possibility of machines exhibiting human-like intelligence. Minsky contributed both his experimental insights from SNARC and his theoretical work on neural networks. He emphasized the potential of networks of simple units to learn through adaptation, but also offered critical observations. He warned that while neural networks could simulate certain learning processes, they were limited in scalability and computational efficiency given the hardware constraints of the time. Minsky recommended pursuing hybrid approaches, combining rule-based symbolic reasoning with network-based learning—foreshadowing the split in AI between symbolic and connectionist paradigms.

Although Minsky constructed the world's first known neural network, by the early 1960s he became more skeptical of their potential and published work that illustrated neural network's limitations. Little progress on

the issues raised during the 1956 conference had been made. Several decades of slow progress indicate that Minsky's analysis was prescient. It wasn't until 2014 that neural networks cemented their position as leading candidates toward true AI. Hardware had finally become sufficiently powerful to overcome the obstacles of scalability and implement training algorithms with sufficient speed. Had he lived long enough to witness the success of neural networks, Minsky's actual biological neural network would have received its well deserved reward.

9.9 The Architect: Frank Rosenblatt(1958)

Frank Rosenblatt was born in 1928 in New Rochelle, New York, into a family that encouraged both music and science. As a boy he played the violin and loved to draw, an early hint of the creative streak that would later fuel his scientific imagination. After a brief period of service in the U.S. Navy at the end of World War II, Rosenblatt entered Cornell University, where he majored in psychology. His formal training remained rooted in experimental psychology and neurobiology, but he also displayed a natural gift for mathematics. Professors and colleagues later recalled that he moved comfortably between the language of neurons and the abstractions of geometry, an ability that allowed him to prove precise mathematical results even while thinking like a psychologist about the nature of learning.

Rosenblatt's deepest interest lay in understanding how the brain processes information. He wanted to know how the visual system turns raw sensory data into organized perceptions and how experience reshapes the connections among neurons. His early research was not aimed at building practical computers. Instead, he envisioned machines as laboratories for exploring neural principles, experimental testbeds that could reveal how networks of simple elements might learn. Only later, when his ideas began to attract public attention, did he suggest that these same devices might one day serve as general-purpose tools for pattern recognition or artificial intelligence.

The most famous of these machines was the perceptron, which Rosenblatt conceived as a mathematical model of neural processing and then realized as a working device. The perceptron consisted of two layers of artificial neurons: an input layer, which captured information from the environment, and an output layer, which produced a decision. Each input unit detected a basic feature of a stimulus—for example, whether a particular spot on a grid of lights was illuminated. Every input neuron was connected to every output neuron by a set of adjustable weighted wires. Information flowed in a single direction: signals entered through the input layer, passed along the weighted connections, and were summed by each output neuron. If the total input to an output neuron exceeded a preset threshold, that neuron fired. The collective firing pattern of the output layer constituted the machine's answer to the stimulus.

Although Rosenblatt supplied the theoretical architecture and the learning algorithm, he was not an electrical engineer and did not construct the perceptron with his own hands. The hardware realization of his design—the Mark I Perceptron built at the Cornell Aeronautical Laboratory in Buffalo—was carried out by a team of engineers and technicians. Rosenblatt worked closely with this technical staff, translating his conceptual blueprint into engineering specifications. The Cornell team then used hundreds of photocells, adjustable potentiometers, and relays to create a physical device that matched Rosenblatt's theoretical model. In this way, Rosenblatt's ideas moved from an abstract psychological construct to a functioning machine; a two layered neural network with 512 neurons in the input layer and 8 neurons in the output layer.

Training the perceptron began with a task and a training set of example inputs. In one well-known experiment, the Mark I was asked to distinguish between two shapes—say, triangles and squares—presented as patterns of light on its input grid. Each example pattern was labeled with the correct category. The machine started with random weights on its connections. During each trial a labeled pattern was projected onto the input layer,

producing a tentative decision in the output layer. If the perceptron's decision matched the correct label, no change was made. If it erred, the connections were adjusted according to Rosenblatt's learning rule:

- Connections from active input neurons that should have contributed more to the correct output were strengthened, making it easier for them to drive the output neuron in the right direction next time.
- Connections that incorrectly encouraged the wrong output were weakened, reducing their influence on future decisions.

By repeatedly presenting examples and making these incremental corrections, the perceptron gradually settled on a stable set of weights that produced the correct output for every pattern in the training set.

Once trained, the perceptron was tested with new patterns not included in the training set, such as a triangle shifted to a different position. Successful classification of these novel patterns showed that the machine had captured the underlying rule separating the categories rather than merely memorizing specific examples.

Rosenblatt's mathematical talent allowed him to go beyond demonstrations to formal guarantees. He proved what is now known as the perceptron convergence theorem: if the categories in the training set are linearly separable—that is, if a straight-line boundary (or its higher-dimensional equivalent) exists that can divide the categories—then the perceptron learning rule will, in a finite number of steps, find a set of weights that perfectly separates them. Later mathematicians refined his argument and supplied sharper bounds, but the essential proof was Rosenblatt's and gave his work scientific legitimacy.

The perceptron's strengths were striking for the late 1950s. It could learn from experience, adapt to new inputs, and discover decision boundaries without explicit programming. Yet the same simplicity that made it elegant also imposed limits. A single-layer perceptron could not solve problems that required combining features in more complex, non-linear ways, such as the famous XOR problem [2]. Rosenblatt recognized this and proposed extending the perceptron to include multiple layers of neurons, which is the modern day architecture for neural networks. But he did not provide a practical method for training such networks. This was the next phase of development.

9.10 An Essential Midpoint: Widrow and Hoff (1960)

ADALINE: The Adaptive Linear Neuron

Bernard Widrow and Marcian Hoff were pioneering engineers at Stanford University in the late 1950s and early 1960s. Widrow, a professor fascinated by adaptive systems, combined rigorous mathematical insight with practical electronics, while Hoff, one of his graduate students, contributed exceptional computational skill and a talent for turning theoretical ideas into working systems. Together, they developed ADALINE (ADAptive LInear NEuron) in 1960, a model capable of adjusting its internal connections to match desired outputs and learn from experience. Their collaboration bridged theory and hands-on implementation, reflecting both Widrow's engineering intuition and Hoff's facility with computation.

The problem that Widrow and Hoff addressed is the same as the least-squares problem that Legendre and Gauss had solved at the beginning of the 19th century: finding the set of weights connecting inputs to outputs that minimizes the sum of squared differences between predicted and observed outputs. However, the scale of ADALINE's problem, while theoretically amenable to Gauss and Legendre's solution method, made the direct solution methods of Gauss and Legendre infeasible. ADALINE networks could involve far more inputs than those that Legendre and Gauss explicitly solved. Datasets were also orders of magnitude larger, containing

[2] XOR stands for exclusive or. It requires one to identify an exlusive outcome among an arbitrary number of outcomes; i.e. a statement is A xor B xor C is true if one and only one of the statements is true,

thousands of datapoints. Solving the corresponding matrix equation as Legendre and Gauss did would require storing and processing enormous matrices, far beyond the memory and speed limitations of 1960s computers. Thus, while the mathematical form was identical, practical solution methods needed to be different.

Structurally, ADALINE can be viewed as a perceptron without thresholding: each input is multiplied by a weight and directly summed to produce the output. Conceptually, the network performs a linear regression, adjusting weights to minimize the difference between its predicted output and the true output from the training data. The challenge was efficiently finding these weights in such a high-dimensional system.

To solve this problem, Widrow and Hoff employed stochastic gradient descent (SGD). Gradient descent relies on the concept of the gradient, a computational analog of feeling the slope of a hillside. Imagine a blind person standing on a hill, trying to reach the lowest point. By probing the ground with a cane, they can detect the steepest downward direction and take a step in that direction. Similarly, the gradient indicates which direction each weight should move to most effectively reduce the error. By repeatedly calculating the gradient and making small adjustments, the network incrementally moves closer to the optimal weights.

For very large datasets, computing the gradient over all datapoints at once would still be time- and memory-intensive. Widrow and Hoff solved this by updating weights after each datapoint and cycling repeatedly through the dataset—a method now known as stochastic gradient descent. This incremental approach dramatically reduces computational requirements while ensuring that ADALINE steadily approaches the optimal solution.

Widrow and Hoff demonstrated the effectiveness of stochastic gradient descent through both synthetic and real-world tasks. In synthetic tasks, ADALINE was trained on known input-output pairs, showing that even from arbitrary initial weights, repeated updates could reduce the error to near zero. In practical applications, ADALINE was applied to adaptive signal processing problems, such as predicting slowly varying signals, canceling echoes in communication channels, or filtering noise from electrical signals. In all cases, the network adjusted weights incrementally, steadily improving performance without requiring the full-matrix computations that were impossible on early computers.

While limited to linear problems—and thus unable to model complex nonlinear decision boundaries—ADALINE introduced several conceptual and computational innovations that were critical for the next generation of neural networks. These included iterative weight adjustment, gradient-based learning, and methods to handle massive datasets efficiently. In this sense, ADALINE serves as a crucial midpoint in the evolution from early perceptrons to the deep neural networks that power modern artificial intelligence.

9.11 Layering on the Missing Piece: Paul Werblos (1974)

Paul Werbos occupies a pivotal place in the history of neural networks, bridging the early perceptrons of Rosenblatt and the linear learning methods of ADALINE to the powerful multi-layer networks of modern AI. Building on the ideas pioneered by Widrow and Hoff with ADALINE, Werbos recognized that the same principle of adjusting weights to minimize error could, in principle, be extended to networks with multiple layers of neurons.

Werbos's key insight was that training a multi-layer network required an efficient method to determine how changes in each connection would affect the overall error. While ADALINE could calculate the gradient for a single-layer network, multi-layer networks introduced additional complexity: each neuron in a hidden layer contributes indirectly to the output error through its influence on subsequent layers. Directly computing the effect of each weight on the total error seemed intractable.

Werbos solved this problem by formulating a systematic backpropagation algorithm. The method calculates the gradient of the error function efficiently by propagating error signals backward from the output layer toward

the input layer, one layer at a time. Using these gradients, the network can adjust each weight slightly in the direction that reduces the error, extending the principle of stochastic gradient descent from ADALINE to multi-layer networks.

Backpropagation transformed neural networks from simple linear classifiers into a general-purpose learning method capable of discovering internal representations in hidden layers. Its strength lies in its generality: it can be applied to any network with differentiable activation functions and provides a repeatable, structured way to reduce error. However, the method also has limitations. Gradient-based learning can converge to suboptimal solutions in error landscapes with many local minima, it is sensitive to learning rates, and large networks may still require substantial computational resources.

Werbos first introduced these ideas in his 1974 Ph.D. dissertation, and he revisited and refined them in the 1980s as computational power increased, exploring practical applications and demonstrating how the approach could scale to larger networks and more complex tasks. In this period, Werbos focused on problems that involved predicting time-varying signals, control systems, and pattern recognition tasks. For example, one study used a dataset consisting of input-output sequences representing dynamic system behavior, with the goal of training a network to predict future outputs based on past observations. The network architecture included an input layer representing current and past system states, one or more hidden layers to capture internal patterns, and an output layer producing the predicted signals.

To train the network, Werbos applied his backpropagation algorithm iteratively, adjusting weights to minimize the difference between the network's predictions and the actual outputs. By testing the network on sequences not included in the training set, he demonstrated that the network could generalize beyond the examples it had seen, producing accurate predictions for new data. This showed that backpropagation was not merely a theoretical construct but a practical learning method capable of handling real-world, time-dependent problems.

In interviews, Werbos reflected on the challenges he faced during the early years: "Back then, I had to deal with computers that had limited memory and speed. Even thinking about networks with a few dozen neurons was a stretch," he recalled. Yet, by deriving a method to propagate error information backward, Werbos overcame these early computational constraints and created a framework that remains central to artificial intelligence today.

Werbos' backpropagation algorithm was popularized by David Rumelhart, Geoffrey Hinton, and Ronald Williams in their landmark 1986 paper. From that time on, refinements of Werbos' method have been central to the training of neural networks, enabling the deep learning architectures that power modern AI applications.

9.12 The Other Half

Modern neural networks run almost entirely on digital computers, and their success depends on two critical components: the software that defines how networks learn, and the hardware capable of performing the immense number of computations required. With the development of Paul Werbos' backpropagation algorithm in the late 1970s, the software side of neural networks was ready for experimentation and further development. However, the hardware side, if judged by the standard needed for modern day neural networks, was pathetic.

To put the challenge in perspective, the human brain contains roughly 85 billion neurons and trillions of synapses. Emulating even a small fraction of this scale in an artificial system requires enormous numbers of synthetic neurons and adjustable weights. Frank Rosenblatt's 1958 Mark I perceptron contained 520 artificial neurons connected by 4,096 adjustable synaptic weights. It was trained on a dataset of 10,000 example images. At this scale it was only able to perform the simplistic task of distinguishing between a triangle and a square. Multi-layered networks of far larger scale along with significantly larger datasets would be required to

enable neural networks to learn and perform more consequential tasks.

In 2012, a neural network called AlexNet, created by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, won the ImageNet Large Scale Visual Recognition Challenge. This competition required entries to categorize images across 1,000 object categories. AlexNet achieved a top-5 classification error of 15.3 percent—more than ten percentage points lower than the next-best competitor—a dramatic improvement in a field where even single-percentage gains are significant. This success demonstrated that, with sufficient data and computational resources, deep neural networks could outperform all previously known methods, sparking the deep learning revolution and cementing neural networks as the leading paradigm in artificial intelligence.

AlexNet's architecture exemplifies the scale of computation required for state-of-the-art networks. It contained roughly 650,000 neurons across eight learned layers and approximately 60 million adjustable synaptic weights in addition to bias terms (bias terms are discussed later in this chapter). The network was trained on 1.2 million images, with each image processed approximately 90 times over the course of training, requiring a total of about 4.6×10^{17} floating-point operations (that is, 460 quadrillion individual calculations). A single backward propagation pass on one image requires roughly 4.3 billion floating-point operations [3]

To appreciate the computational leap from early computers, consider that in 1961 the IBM 7030 Stretch,the world's fastest and most expensive computer, The Stretch performed roughly 600,000 floating-point operations per second. At that rate, training AlexNet would have taken approximately 24,500 years. By contrast, using two NVIDIA GTX 580 GPUs, the AlexNet team completed training in roughly nine days, highlighting the dramatic advances in both processing speed and parallelism that made deep learning feasible.

The evolution of hardware that enabled these advances can be traced through a series of pivotal innovations:

Year / Contributor	**Contribution**	**Relevance to Neural Networks**
1801 / Joseph Marie Jacquard	Invented the Jacquard loom using punched cards to control weaving patterns	Early concept of programmable machinery; inspiration for storing and controlling instructions in machines
1837 / Charles Babbage	Designed the Analytical Engine, a general-purpose mechanical computer	Conceptual foundation for programmable computation and separation of data and instructions
1936 / Alan Turing	Defined the Universal Turing Machine	Established the theoretical basis for any computational device, including networks simulating computation
1937 / Claude Shannon	Demonstrated that Boolean algebra can be implemented with switching circuits	Showed that networks of binary elements can compute arbitrary logical functions; foundation for digital computation and network analysis
1939–1945 / Konrad Zuse	Built the Z3, the first programmable digital computer	Demonstrated practical implementation of binary computation using electromechanical relays

[3] A floating-point operation is an arithmetic operation performed on rational numbers using decimal expressions in a base 2 numerical system.

Year / Contributor	Contribution	Relevance to Neural Networks
1943–1945 / Colossus	First programmable electronic computer (UK, wartime codebreaking)	Early use of electronics for high-speed computation; demonstrated advantage over mechanical systems
1945 / ENIAC (John Mauchly, J. Presper Eckert)	First general-purpose electronic digital computer using vacuum tubes	Enabled practical high-speed computation necessary for large-scale neural network simulations
1947 / Bell Labs	Invention of the transistor	Replaced vacuum tubes, drastically increasing reliability, speed, and miniaturization of circuits
1958 / Jack Kilby and Robert Noyce	Developed the integrated circuit (IC)	Allowed multiple transistors to be fabricated on a single chip, foundational for compact, fast, and scalable computers
1960s–1970s / Early minicomputers	DEC PDP series and others	Provided accessible computing platforms for scientific experiments and early neural network simulations
1970s–1980s / Microprocessors	Intel 4004, 8080, and successors	Brought general-purpose computation to compact, affordable platforms suitable for implementing small neural networks
1980s / Parallel computers	Connection Machine, SIMD and MIMD architectures	Enabled simultaneous processing of multiple units, critical for large neural networks and later for backpropagation training
1990s / Graphics Processing Units (GPUs)	Initially for graphics rendering; later adapted for general-purpose computing	High-throughput parallel computation allows fast training of large neural networks; essential for deep learning
2000s / Multi-core CPUs and clusters	Commodity multi-core processors and networked clusters	Scalable computing infrastructure for larger neural networks, distributed training, and research experimentation
2010s / GPU-accelerated deep learning	NVIDIA CUDA-enabled GPUs, Tensor Cores	Key hardware enabling practical training of deep neural networks and modern AI applications
2020s / Specialized AI hardware	TPUs, neuromorphic chips (IBM TrueNorth, Intel Loihi)	Dedicated architectures for neural computation, improving energy efficiency and performance for inference and learning

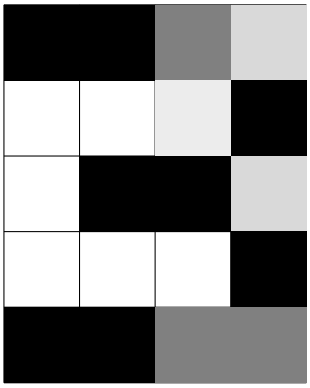
9.13 Common Neural Network Architectures

This section presents details behind the architecture and training of a neural network. The presentation introduces the three most common designs, fully connected, convolution neural network (CNN), recurrent neural

network (RNN). The presentation combines graphics that provide visual representations with narrative that expands upon the visual representation. For concreteness, the presentation centers on examples.

Fully Connected

Consider a neural network that is designed to determine a handwritten number 0 through 9. A preprocessing program converts the handwriting to a grey scale for 30 pixels aligned along 5 rows and 4 columns. Preprocessing centers the handwritten digit into a 5 by 4 rectangle and associates each pixel with a unique 1 by 1 square. For a given square, if there is no handwritten marking, the grey scale for the corresponding is zero. Alternatively, if the square is completely filled, the corresponding entry is 255; entries are always integer, there are $2^8 = 256$ possible entries. The figure below illustrates the preprocessing outcome on a handwritten sample of the number 3.



255	255	200	140
0	0	100	255
0	255	255	140
0	0	0	255
0	255	200	200

Preprocessed Grid with Handwritten 3 Corresponding Grayscale Values

Figure 9.3

Let us consider a perceptron that one might construct to determine the number. The perceptron has only two layers. an input and an output layer. The input layer must accept all of the possible inputs, one neuron for each input. As there are 20 inputs, the perceptron has 20 input neurons. There are ten possible outcomes, 0 through 9. Align the neurons in a column with the intention that an input only activates a single output neuron and the activated neuron indicates the number; i.e. when the number three is the input, only the third neuron activates.

The gray levels in the first row of figure 8.3b are input into neurons n_1 through n_4 of figure 8.4. The second row of gray levels from figure 8.4b are input into neurons n_5 through n_8 of figure 8.4 and so on. Each input neuron has a connection to every output neuron. There are a total of 200 connections, each with its own weight.

The perceptron is a neural network with two layers. Neural networks are numerical processors; input numbers are numerically processed into output numbers. The processing is as follows.

$$\begin{aligned} r_j = \sum i = 0^2 0 w_{ij} x_i + b_j \\ y_j = f(r_j) \end{aligned} \tag{9.1}$$

where

- j represents the j^th output neuron

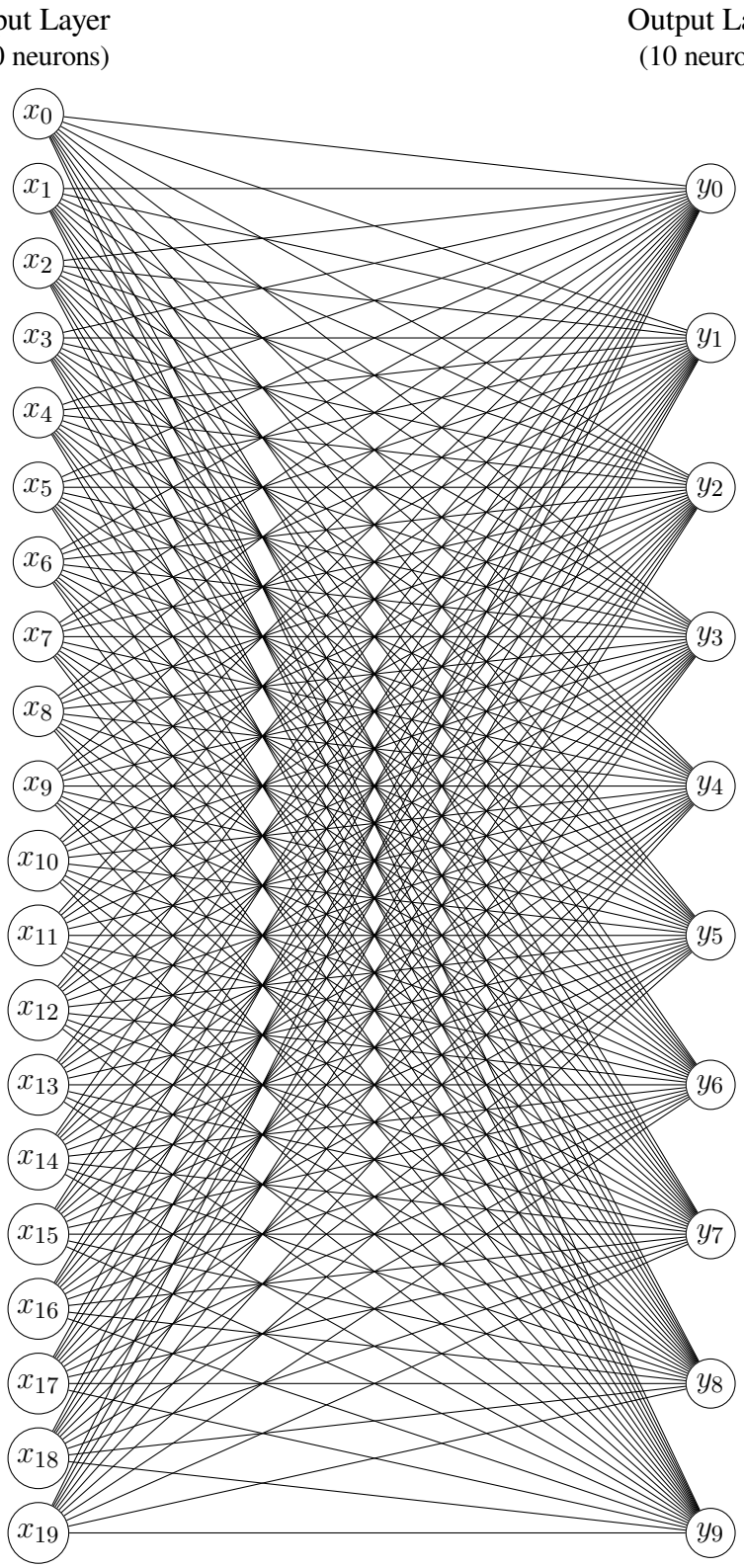

Input Layer
(20 neurons)
Output Layer
(10 neurons)
x_0
x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}
x_{11}
x_{12}
x_{13}
x_{14}
x_{15}
x_{16}
x_{17}
x_{18}
x_{19}
y_0
y_1
y_2
y_3
y_4
y_5
y_6
y_7
y_8
y_9

Figure 9.4

- w_{ij} is the weight of the connection between the i^{th} input neuron and j^{th} output neuron
- b_j is a regression bias term
- y_j is the output of the j^{th} neuron
- $f(r_j)$ is a function applied to r_j that is known as the activation function.

Summarizing, the output of each neuron is an activation function applied to a linear regression on the inputs. A commonly used activation function is the following.

$$\begin{aligned} f(r_j) &= 0 \text{ whenever } r_j < 0 \\ &= r_j \text{ whenever } r_j >= 0 \end{aligned} \tag{9.2}$$

The outputs of the activation function are real valued nonnegative numbers. The neuron with the highest activation indicates the network's identification of the input number. In the above case, note that the biasing term, b_j, can be thought of as a changing the threshold where activation begins.

The above two layered network does not work well. Considerable improvement is possible by adding more layers as in the figure below.

The network of figure 8.5 has four layers, In addition to the input and output layer, there are two additional layers often referred to as hidden layers. There are no theoretical restrictions on the number of hidden layers or number of neurons in each hidden layer. However, practical considerations apply. More neurons in a given layer require more weights, which in turn requires more training data and training time. There is little theory that indicates how one should scale a neural network. Practitioners rely upon experience.

In a multilayered neural network, the output of a hidden layer becomes the input of its subsequent neighboring layer. Equation 8.1 gives the numerical processing between the input and output of any two neighboring layers.

From Eyeballs to Convolution Neural Networks (CNNs)

When scientists first began building artificial neural networks, they imagined a machine that could learn like a human brain. But early models—what we now call fully connected networks—were clumsy when faced with images. A fully connected network treats every pixel as equally related to every other pixel, as if each tiny square of an image must communicate with every other square. For tasks like reading handwriting, this quickly became overwhelming: the number of connections exploded, and the network struggled to capture the simple fact that patterns in an image tend to be local.

Inspiration for a better design came not from mathematics alone but from biology. In the 1950s and 1960s, neurophysiologists David Hubel and Torsten Wiesel explored the visual cortex of cats and monkeys. They discovered that neurons in the early visual areas of the brain are arranged in layers, and that different groups of neurons respond only to small patches of the visual field—a corner here, an edge there. Some cells detect simple features like orientation or edges, while others combine those features to recognize more complex shapes. This layered, locally focused organization provided a powerful clue: perhaps artificial networks should also process images through local receptive fields and share the same set of feature detectors across the visual field.

Yann LeCun, a French computer scientist with a passion for both mathematics and biological models of vision, absorbed these ideas in the 1980s. After finishing his doctorate in Paris, he joined AT&T Bell Laboratories in Holmdel, New Jersey, a legendary research center where innovations in computing and telecommunications flourished. At Bell Labs, LeCun set out to create a neural network that would mimic the layered processing of the visual cortex while remaining computationally efficient.

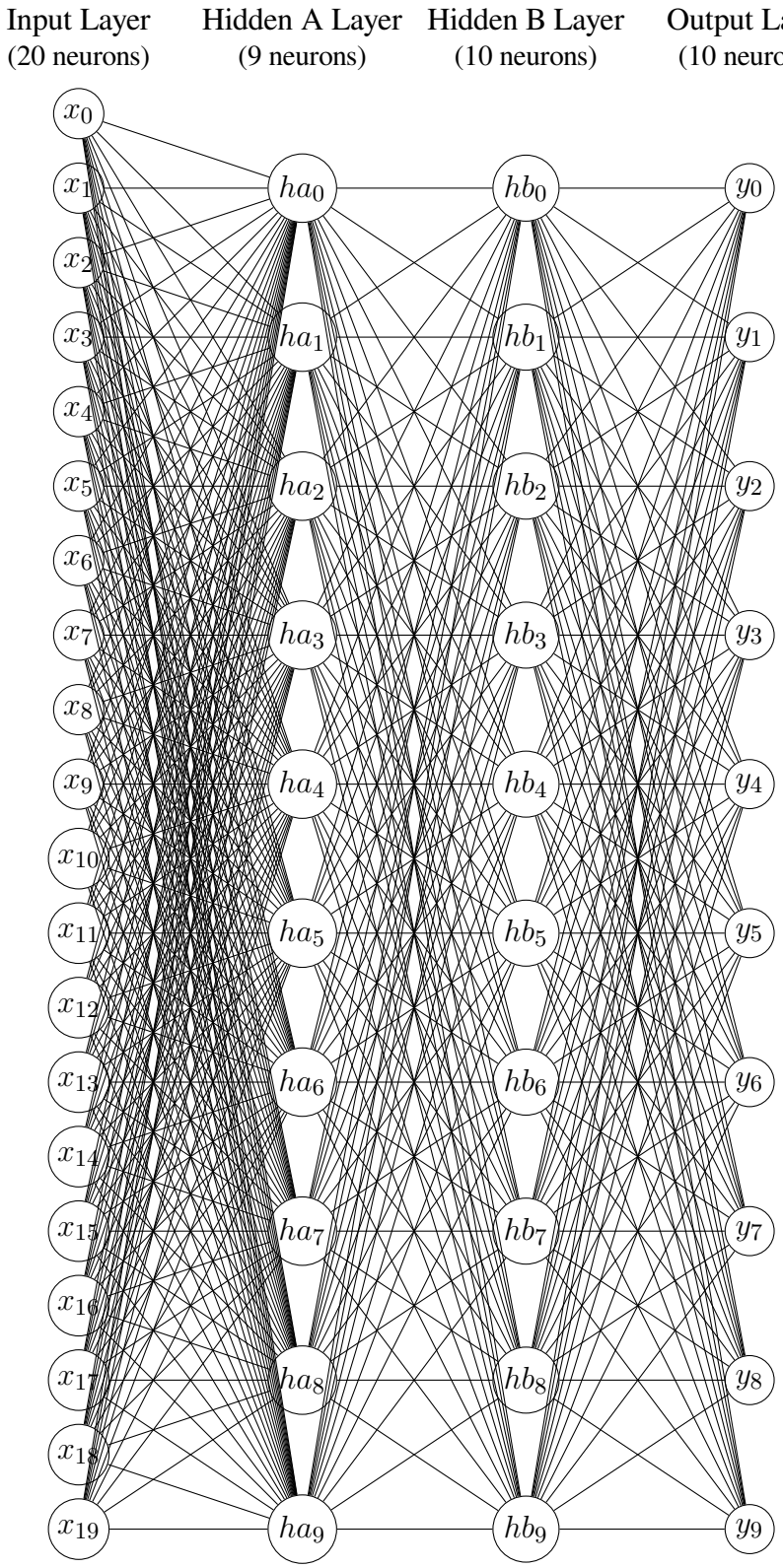

Input Layer
(20 neurons)
Hidden A Layer
(9 neurons)
Hidden B Layer
(10 neurons)
Output Layer
(10 neurons)
x_0
x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}
x_{11}
x_{12}
x_{13}
x_{14}
x_{15}
x_{16}
x_{17}
x_{18}
x_{19}
ha_0
ha_1
ha_2
ha_3
ha_4
ha_5
ha_6
ha_7
ha_8
ha_9
hb_0
hb_1
hb_2
hb_3
hb_4
hb_5
hb_6
hb_7
hb_8
hb_9
y_0
y_1
y_2
y_3
y_4
y_5
y_6
y_7
y_8
y_9

Figure 9.5

His answer was the convolutional neural network, or CNN. In LeCun's design, small filters—mathematical "feature detectors"—slide across an image, looking for patterns such as edges or corners. The result is a convolution layer with localized input-output connection that dramatically reduce the number of weights compared with a fully connected network. After each convolution layer comes a subsampling (or pooling) layer, which compresses information and provides a degree of position invariance, much like how the brain tolerates small shifts in the position of a visual stimulus.

LeCun's first CNNs appeared in 1989 and gradually evolved into LeNet-5, published in 1998. This architecture featured two convolutional layers, each followed by a pooling layer, and finally a set of fully connected layers for classification. But the design alone would not have been enough; CNNs also needed training data and computational power.

In the late 1980s the U.S. Postal Service and major banks faced a costly bottleneck: millions of handwritten checks and envelopes arrived every day, and humans had to read and sort them. AT&T Universal Card Services and the Postal Service were eager for an automated solution. Bell Labs researchers, including LeCun, began collaborating with the U.S. National Institute of Standards and Technology (NIST) to create a massive dataset of handwritten digits for training and testing recognition systems.

NIST collected tens of thousands of handwritten samples from two distinct populations: U.S. Census Bureau employees and American high-school students. The combination captured a wide range of writing styles, from careful clerks to energetic teenagers. LeCun and colleagues carefully split the data so that training and testing sets came from different writers, ensuring that the network would learn to generalize rather than memorize. To make experimentation easier for researchers worldwide, LeCun later cleaned, normalized, and combined these samples into a more compact dataset called MNIST (Modified NIST), which eventually contained 60,000 training images and 10,000 test images, each a 28×28 grayscale picture of a single digit.

Today, a laptop can train a LeNet model in minutes. In the early 1990s, training required days of computation on the powerful workstations of the era—DECstation and Sun SPARCstation UNIX machines. These were top-of-the-line RISC computers for research, but they lacked the specialized graphics processors that now accelerate deep learning. LeCun's team wrote custom C code to implement backpropagation and manage the convolution operations efficiently.

Despite these limitations, the results were remarkable. LeNet achieved error rates around 0.8 percent on the NIST digit set, far surpassing traditional methods such as k-nearest neighbors or handcrafted feature extraction. For the first time, a machine could read human handwriting at a level suitable for industrial deployment.

AT&T put LeNet into production in the early 1990s for its check-reading systems, and the U.S. Postal Service incorporated similar networks into automated mail-sorting machines. By the mid-1990s, these CNNs were processing millions of checks and envelopes every day, proving that neural networks were not just academic curiosities but practical tools.

Although CNNs would later be overshadowed for a time by support vector machines and other techniques, LeNet quietly laid the foundation for the deep-learning revolution. When researchers returned to neural networks in the 2010s, they rediscovered LeCun's ideas almost intact. AlexNet—the 2012 model that reignited deep learning—was essentially a scaled-up LeNet running on modern GPUs and trained on a far larger image dataset.

Meanwhile, the MNIST dataset became the "Hello World" of machine learning. Students, hobbyists, and researchers still use it to test new ideas, from novel optimization algorithms to exotic network architectures. Nearly every practitioner of deep learning has, at some point, trained a model to recognize the digits in MNIST—a quiet tribute to the work that LeCun and his collaborators began decades ago.

RNN: On The Merry Go Round

The recurrent neural network (RNN) is a network that generates a sequence of outputs having patterns that are extracted from a training set. After training, to generate a sequence, one enters a subsequence into the input. The subsequence initializes the full sequence that the RNN generates.

A fully connected network or perhaps a CNN or another architecture then processes the input (initial subsequence) into both an output and supplemental information. The supplemental information, which the rnn creates is deemed important for making a prediction of the next output. The supplemental information is often called a hidden state.

To create an input for the next element, remove the first element, slide the remaining elements over by one position and add the hidden state into the final position. The neural network will then generate a new element and a new hidden state. The process continues until a stopping criteria is met; i.e. produce a series of 1000 elements and then stop. Figure 8.6 illustrates the process using an example that predicts the next character of text.

Robotics provides a use-case for RNNs. Using mathematical models, one can train the neural network to determine the state of a robot (position and speed) at after a specified interval of time if its current state is known and the status of the control motors over the specified interval is known. Once the RNN is trained, one can generate a sequence of motor control that drives the robot from an initial state to a desired state.

Language models that generate text one word at a time also use RNNs. The next chapter discusses more details concerning language models. A rather quirky experiment toward generating text was Anrei Karpathy's efforts to generate sequences one character, rather than one word at a time (figure 8.6).

In 2015–2016 Karpathy, a Stanford graduate student published a series of experiments he called char-RNN. A publication that garnered much attention was entitled *The Unreasonable Effectiveness of Recurrent Neural Networks*.[4]

Karpathy's motivation was twofold. First, he wanted to show that an RNN could learn long-range structure—punctuation, indentation, rhyme, even the formatting of plays—without ever being told what a word is. Second, by stripping language down to raw characters he could sidestep the need for a predefined vocabulary and demonstrate the raw pattern-finding power of recurrent networks. The work built on decades of RNN research, beginning with Paul Werbos's backpropagation-through-time in the 1970s and refined by researchers such as Jürgen Schmidhuber and Sepp Hochreiter, but Karpathy presented it with modern GPUs, better optimization, and a gift for storytelling that captured the imagination of the AI community.

His training sets were wonderfully eclectic: the complete works of Shakespeare, source code from the Linux kernel, Paul Graham's startup essays, a database of baby names, and more. Each dataset was fed to the RNN as an unbroken stream of characters. The network's task was simple but demanding: predict the next character given all the previous ones. After many hours of GPU training the RNN learned not just spelling but also higher-order patterns—stage directions, programming syntax, even the cadence of prose.

The results were often startlingly convincing. From the Shakespeare corpus, for example, the network produced passages such as:

KING HENRY IV:
O gentlemen, the time of life is short;
To spend that shortness basely were too long,

[4]Karpathy may have been influenced by Eugene Wigner's famous 1960 paper, *The Unreasonable Effectiveness of Mathematics in the Natural Sciences*

If life did ride upon a dial's point,
Still ending at the arrival of an hour.

LADY PERCY:
My lord, I did deny no prisoners.
But I remember, when the fight was done,
When I was dry with rage and extreme toil,
Breathless and faint, leaning upon my sword,
Came there a certain lord, neat and trimly dressed,
Fresh as a bridegroom; and his chin new reaped
Showed like a stubble land at harvest home.

Notice the coherent meter, the invented but plausible character "Lady Percy", and the way the network sustains a poetic rhythm over many lines.

Karpathy's playful char-RNN became a viral example of what RNNs could do, inspiring countless follow-up projects and serving as a hands-on introduction for students of sequence modeling. More importantly, it foreshadowed the power of modern language models: if a relatively small network could capture Shakespearean drama character by character, what might larger networks learn when trained on the entire internet?

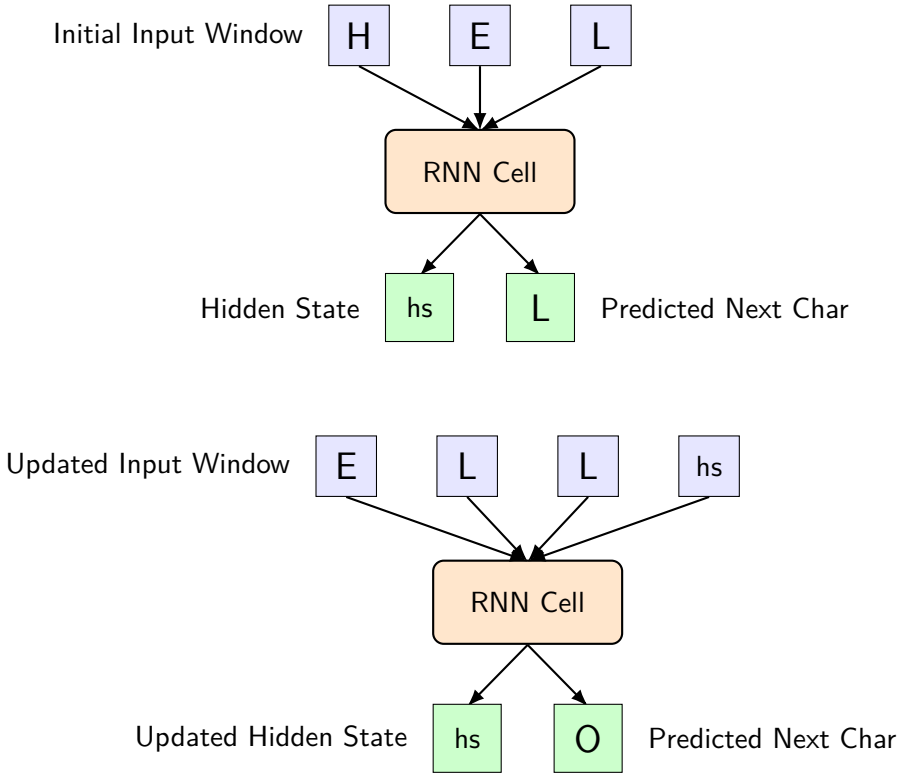


Figure 9.6: Iterative character-by-character generation in an RNN. The model consumes a sliding window of characters (here starting with "HEL"), predicts the next letter ("L"), then appends it and shifts the window to continue the sequence. In addition, the model outputs a hidden state that is used as an input in the next iterate.

9.14 The Penguin Map: RNN's Discovery From Data

Suppose there is a sequence of points in the plane generated as follows. Choose an initial point and designate it as the current point. Then apply a function to the current point to obtain the next point. Relabel the next point as the current point and apply the function to attain a new point. Continue iteratively until a stopping condition is in effect. Physical systems, such as the weather operate in such a manner. The weather tomorrow depends upon today's condition. The laws of physics determine the function that yields tomorrow's weather from today's conditions.

Now let us suppose that such a sequence of points is available, but the function that generates the sequence is unknown. The challenge is to create a function, sometimes called a map, that replicates the data. Knowing the laws of physics and with enough information about current conditions around the world, one can make a model that inputs the current conditions and outputs a forecast of tomorrow's weather. Perhaps, given enough input information, an RNN could produce an accurate forecast. Below is a much more modest challenge.

Figure 8.7 is a graph of a sequence that we call the penguin map. Using the data alone, an RNN replicates the figure, somewhat like the outline of a penguin. Figures 8.8 and 8.9 are different plots of the RNN model against the true model at different stages of training. Training blocks are often described by epochs where one epoch indicates that the entire training dataset has been placed through the training program one time.

Figure 8.8 is the result of placing the dataset through the training program 9950 times, 9950 epochs. Figure 8.9 is the result after 25000 epochs. In general, each training epoch fine tunes the networks weights and biases in a manner that reduces the error between the true result and the prediction.

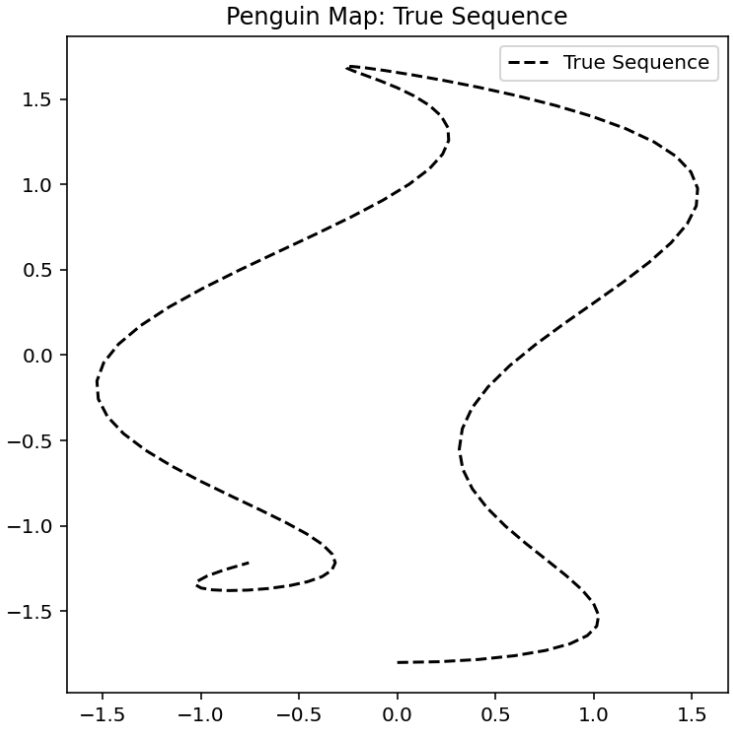


Figure 9.7

After 25000 epochs, the RNN is able to accurately predict the next point, even though it does not know the function that governs where the next point lies. What's more, the RNN uses its prediction of the next point

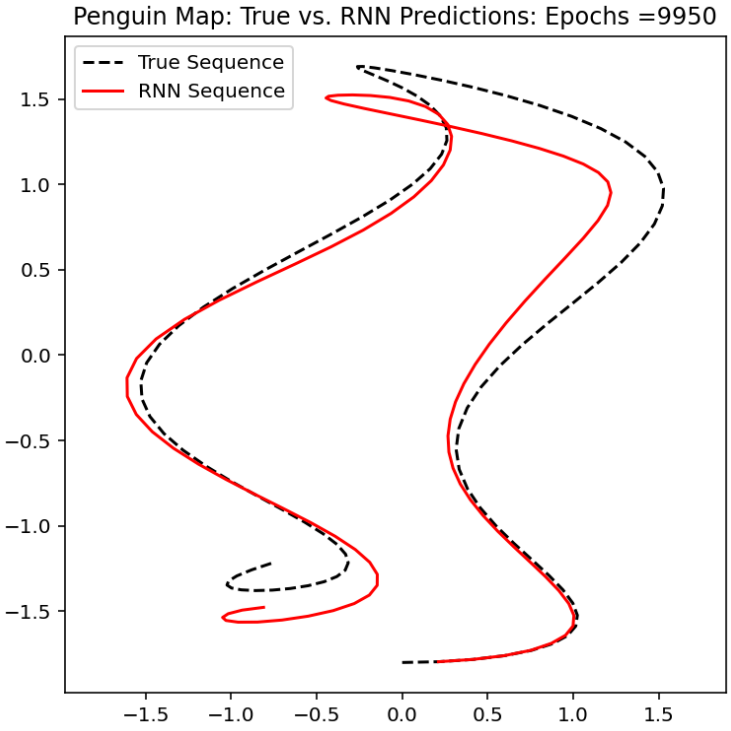


Figure 9.8

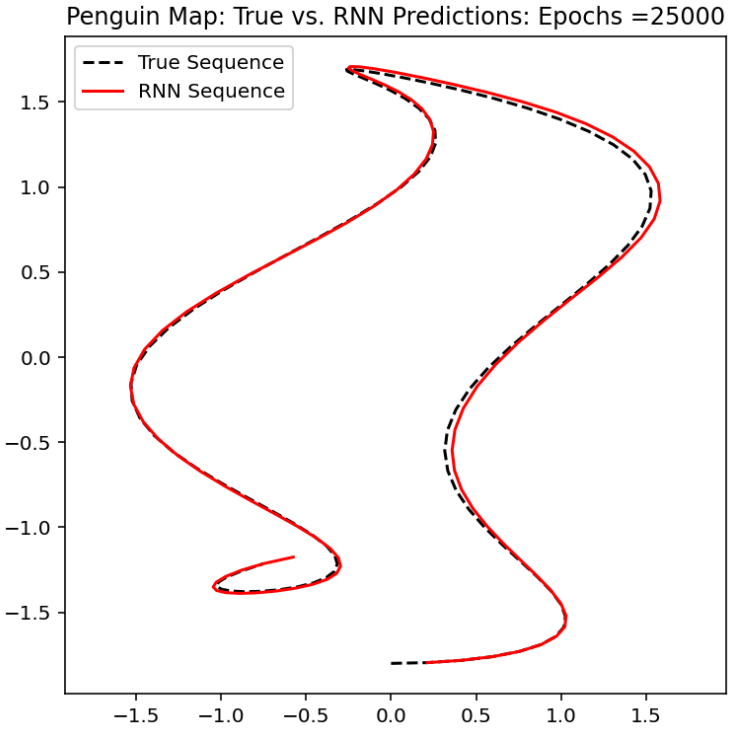


Figure 9.9

to then predict the following point in the penguin map's sequence. If the predictions were slightly off, the prediction errors would accumulate and the RNN's trace would stray from that of the penguin as happens in figure 8.7. But the predictions are very accurate and the RNN stays on the correct path, even though it was only presented with the initial point and is using its own internally created model to predict the entire penguin sequence.

9.15 The Penguin Map and the the Data Scientist

This section presents the development of the penguin map mimicking RNN through the process of a data scientist. The presentation is brief. The curious reader can find details of the RNN structure, the function that produces the penguin map, the training and prediction process in the Python code given in the appendix. The Python code relies upon Pytorch libraries.[5]

Define the problem.

From a dataset, reproduce the function that creates the data.

Propose an input-output parametric model of the system.

Propose an RNN with specific structure as given in the appendix.

Identify the required data.

An initial point and output from the penguin map. The function creating the data is available in the Python code in the appendix.

Collect and organize data as inputs and outputs.

Using the function from the aforementioned Python code, a dataset yielding the outline of a penguin is available. Points in the dataset are both inputs and outputs.

Define a metric that quantifies performance.

Each prediction produces an error between the actual next point from the penguin map and the RNN's forecast. The metric is the sum of square prediction errors over the entire dataset.

Apply an optimization routine to adjust the parameters and minimize the error.

The Python code uses an implementation of stochastic gradient descent.

Validate results against additional data.

There is a time dependency in the penguin map that confines the process to the outline of the penguin. Because of the time dependency the data provides no information addressing the map's behavior outside of the penguin outline. Accordingly, the task is solely to reconstruct the penguin without considering additional data.

[5] Pytorch is a library of functions that one can use to configure and train a neural network. After training, Pytorch functionality allows one to output predictions from the neural network.

9.16 Final Thoughts

As described artificial neural networks are independent neurons that connect with one another to create interpretable patterns in response to external or internal simulation. The developers who intentionally sought to produce AI connected and contributed with previous developments to produce neural networks. The process of making connections with previous individuals and refining their output is a metaphor for how a neural network operates. Perhaps the metaphor runs deeper and provides insight into how organized systems evolve and operate.

Organized evolution, whether through biological processes, physical processes, or human design, are evident throughout the book. One perspective on evolution is that it is an input-output process. The answer to Aristarchus' quest to determine the relative sizes of the Sun and Earth evolves from inputs and outputs flowing through sets of equations that yield the final answer. The process of biological evolution as conceived by Darwin inputs a lifeform in a changed environment and outputs adaptations of the lifeform to the new environment. Henry Ford's assembly line may be thought of as an input output process. Each stage of the line receives an input from the previous stage and outputs additions to the next stage. The inputs and outputs evolve into an automobile. The neural network passes its inputs and outputs through layers until they evolve into a final response. As noted in the previous paragraph, contributors to the design of neural networks received inputs from their predecessors and output new knowledge that eventually evolved into neural networks.

Perhaps this perspective on evolution provides nothing more than a definition: *evolution is a series of inputs and outputs chained together through time*. But this definition has broad application. Viewing processes like *intelligence* through the lens of evolution yields the neural network. Understanding nature's successful evolutionary processes; how nature transforms inputs to outputs, may lead to enhanced human designs. Once again, the neural network provides an example.

The preceding chapters present historical works that lay the groundwork for AI. The groundwork evolved without any intent or planning. This chapter marks the beginning of a directed effort to create AI focusing on the most successful outcome of the effort, neural networks. With this background the next chapter describes next efforts that result in ChatGPT.

9.17 Summary Poem: The Connectors

From Galvani's frogs that twitched with light,
To sparks that danced through nerves at night,
He showed that life and lightning blend—
A living current without end.

Boole then cast the mind in signs,
With ones and zeros, truth aligns.
He proved that thought, so fluid, grand,
Could rest on logic's steady hand.

Sherrington peered through tinted glass,
And found where living signals pass.
He named the gap—the synapse small—
Where neurons whisper, rise, and fall.

Then Shannon came with circuits bright,
Where bits of truth replaced the light.
He wired the world in Boolean art,

And taught machines to think with heart.

McCulloch, Pitts—two minds entwined—
Mapped neurons into forms of mind.
Each node a switch, each path a plan,
The birth of thought within a span.

Hebb gave these models life's own tone:
"Cells that fire together grow."
Learning's seed, in synapse laid,
Let networks change with what they've made.

Hodgkin, Huxley, proved it real—
With squid and current, pulse and steel.
They showed the ion's rhythmic flow,
The neuron's truth from head to toe.

Then Minsky built his learning maze,
With SNARC that groped through wired haze.
He dreamed of minds both strict and free,
Finite states in circuitry.

Rosenblatt's perceptron took the stage,
A model child of this new age.
It learned to see, to weigh, to tell,
The patterns where our meanings dwell.

Widrow and Hoff refined the way—
They let each weight adjust, obey.
Errors whispered, gradients steered,
So networks learned as doubts were cleared.

Werbos returned with insight deep,
Backpropagation's promised leap.
Errors reversed through hidden lines,
So layered minds could now refine.

Then came the builders—LeCun's view,
From vision's cells to pixels new.
His filters slid where edges lie,
And taught machines to read the sky.

Karpathy's words, in sequence spun,
Brought text to life from code begun.
Through loops that learned, in rhythm bound,
He found where sense and syntax sound.

Now deep within the code we trace
Each thinker's spark, each hand, each face.
From nerve to net, from thought to scheme,
Their work became the learning dream.

Each weight we tune, each line we write,
Still hums with echoes of their light.

From living cells to silicon—
The road to ChatGPT runs on.