Chapter 1

An Informal Introduction to Intelligence

> "*Just as the constant increase of entropy is the basic law of the universe, so it is the basic law of life to be ever more highly structured and to struggle against entropy.*"
>
> – Václav Havel

1.1 Intelligence, Cybernetics, and Artificial Intelligence

The world we inhabit is neither fully random nor completely unpredictable.[1] Instead, it follows certain orders, patterns, and laws that render it largely predictable.[2] The very emergence and persistence of life depend on this predictability. Only by learning and memorizing what is predictable in the environment can life survive and thrive, since sound decisions and actions hinge on reliable predictions. Because the world offers seemingly unlimited predictable phenomena, intelligent beings—animals and humans—have evolved ever more acute senses: vision, hearing, touch, taste, and smell. These senses harvest high-throughput sensory data to perceive environmental regularities. Hence, a fundamental task for all intelligent beings is to

learn and memorize predictable information from massive amounts of sensed data.

[1] If the world were fully random, an intelligent being would have no need to learn or memorize anything.

[2] Some deterministic, some probabilistic.

Before we can understand how this is accomplished, we must address three questions:

- How can predictable information be modeled and represented mathematically?
- How can such information be computationally learned effectively and efficiently from data?
- How should this information be best organized and structured as memory or (empirical) knowledge, in our brain or a machine, to support future prediction and inference?

This book aims to provide some answers to these questions. These answers will help us better understand intelligence, especially the computational principles and mechanisms that enable it. Evidence suggests that all forms of intelligence—from low-level intelligence seen in early primitive life to the highest form of intelligence, the practice of modern science—share a common set of principles and mechanisms. We elaborate below.

Emergence and evolution of intelligence. A necessary condition for the emergence of life on Earth about four billion years ago is that the environment is largely predictable. Life has developed mechanisms that allow it to learn what is predictable about the environment, encode this information, and use it for survival. Generally speaking, we call this ability to learn knowledge of the world *intelligence*. To a large extent, the evolution of life is the mechanism of intelligence at work [Ben23]. In early stages of life, intelligence is mainly developed through two types of learning mechanisms: *phylogenetic* and *ontogenetic* [Wie61].

Phylogenetic intelligence refers to learning through the evolution of species. Species inherit and survive mainly based on memory and knowledge encoded in the DNA or genes of their parents. To a large extent, we may call DNA nature's pre-trained large models because they play a very similar role. The main characteristic of phylogenetic intelligence is that individuals have limited learning capacity. Learning is carried out through a "trial-and-error" mechanism based on random mutation of genes, and species evolve based on natural selection—survival of the fittest—as shown in Figure 1.1. This can be viewed as nature's implementation of what is now known as "reinforcement learning," [SB18] or potentially "neural architecture search" [ZL17]. However, such a "trial-and-error" process can be extremely slow, costly, and unpredictable. From the emergence of the first life forms, about 4.4–3.8 billion years ago [BBH+15], life has relied on this form of evolution.[3]

Ontogenetic intelligence refers to the learning mechanisms that allow an individual to learn through its own senses, memories, and predictions within its specific environment, and to improve and adapt its behaviors. Ontogenetic

[3] Astute readers may notice an uncanny similarity between how early life evolves and how large language models evolve today.

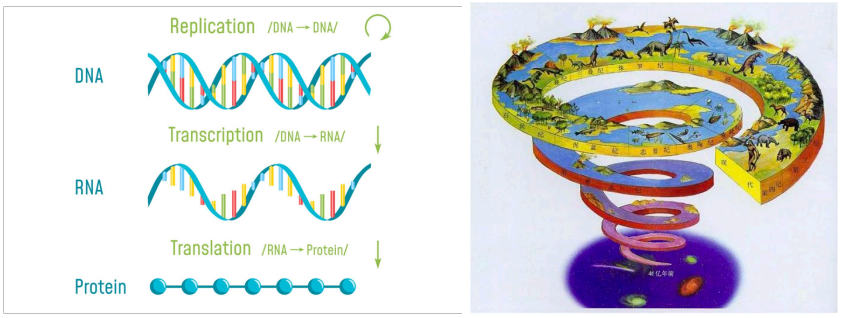


Figure 1.1: Evolution of phylogenetic intelligence: Memory or knowledge of the external world is encoded and passed on via DNA (left), and decoded from DNA to RNA and to proteins. In the early stage of life evolution (right), intelligence develops memory or knowledge at the species level via (random) gene mutation and natural selection—"may the fittest survive"—which can be viewed as a primitive form of reinforcement learning.

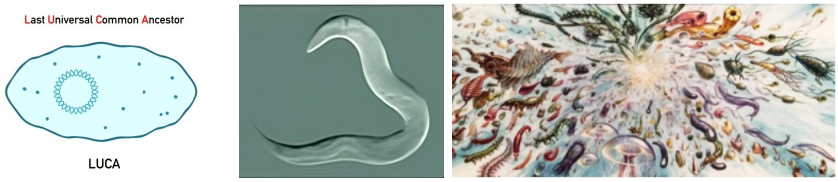


Figure 1.2: Evolution of life, from the ancestor of all life today (LUCA—last universal common ancestor), a single-cell-like organism that lived 3.5–4.3 billion years ago [MÁM+24], to the emergence of the first nervous system in worm-like species (middle), about 550 million years ago [WVM+25], to the explosion of life forms in the Cambrian period (right), about 530 million years ago.

learning became possible after the emergence of the nervous system about 550–600 million years ago (in worm-like organisms) [WVM+25], shown in Figure 1.2 middle. With a sensory and nervous system, an individual can continuously form and improve its own memory about the world, in addition to what is inherited from DNA or genes. This capability significantly enhanced individual survival and contributed to the Cambrian explosion of life forms about 530 million years ago [Par04]. Compared to phylogenetic learning, ontogenetic learning is more efficient and predictable, and can be realized within an individual's resource limits.

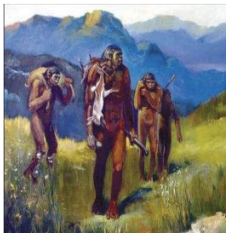
Both types of learning rely on feedback from the external environment—penalties (death) or rewards (food)—applied to a species' or an individual's actions.[4] This insight inspired Norbert Wiener to conclude in his Cybernetics program [Wie48] that all intelligent beings, whether species or individuals, rely on closed-loop feedback mechanisms to learn and improve their memory and knowledge about the world. Furthermore, from plants to fish, birds, and mammals, more advanced species increasingly rely on ontogenetic learning: they remain with and learn from their parents for longer periods after birth, because individuals of the same species must survive in very diverse environments.

Evolution of human intelligence. Since the emergence of *Homo sapiens* about 2.5 million years ago [Har15], a new, higher form of intelligence has emerged that evolves more efficiently and economically. Human societies developed languages—first spoken, later written—as shown in Figure 1.3. Language enables individuals to communicate and share useful information, allowing a human community to behave as a single intelligent organism that learns faster and retains more knowledge than any individual. Written texts thus play a role analogous to DNA and genes, enabling societies to accumulate and transmit knowledge across generations. We may refer to this type of intelligence as *societal intelligence*, distinguishing it from the phylogenetic intelligence of species and the ontogenetic intelligence of individuals. This knowledge accumulation underpins (ancient) civilizations.

About two to three thousand years ago, human intelligence took another major leap, enabling philosophers and mathematicians to develop knowledge that goes far beyond organizing empirical observations. The development of abstract concepts and symbols, such as numbers, time, space, logic, and geometry, gave rise to an entirely new and rigorous language of *mathematics*. In addition, the development of the ability to generate hypotheses and verify their correctness through logical deduction or experimentation laid the foundation for modern *science*. For the first time, humans could proactively and systematically discover and develop new knowledge. This further significantly improved the efficiency of acquiring knowledge about the unknown. We will call this advanced form of intelligence "scientific intelligence" due to its necessity for deductive and scientific discovery.

Hence, from what we can learn from nature, whenever we use the word

[4] Gene mutation of the species or actions made by the individual.



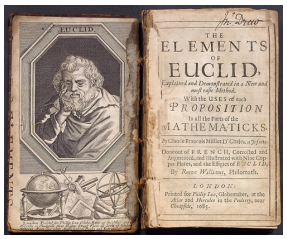


Figure 1.3: The development of verbal communication and spoken languages (70,000–30,000 years ago), written languages (about 3000 BC) [Sch14], and abstract mathematics (around 500–300 BC) [Hea+56] mark three key milestones in the evolution of human intelligence.

"intelligence," we must be specific about which level or form we mean:

$$\textbf{phylogenetic} \implies \textbf{ontogenetic} \implies \textbf{societal} \implies \textbf{scientific intelligence}. \tag{1.1.1}$$

Clear characterization and distinction are necessary because we want to study intelligence as a scientific and mathematical subject. Although all forms may share the common objective of learning and improving memory and knowledge about the world, the computational mechanisms and physical implementations could differ. We believe the reader will better understand and appreciate their commonality and difference after studying this book. Therefore, we leave deeper scientific or philosophical discussions on the relationship among these different forms or levels of intelligence to the end of the book Chapter 9.

Origin of machine intelligence—cybernetics. In the 1940s, spurred by the war effort, scientists inspired by natural intelligence sought to emulate animal intelligence with machines, giving rise to the "Cybernetics" movement championed by Norbert Wiener [Kli11]. Wiener studied zoology at Harvard as an undergraduate before becoming a mathematician and control theorist. He devoted his life to understanding and building autonomous systems that could reproduce animal-like intelligence. Today, the Cybernetics program is often narrowly interpreted as being mainly about feedback control systems, the area in which Wiener made his most significant technical contributions. Yet the program was far broader and deeper: it aimed to understand intelligence as a whole—at least at the animal level—and influenced the work of an entire generation of renowned scientists, including Warren McCulloch, Walter Pitts, Claude Shannon, John von Neumann, and Alan Turing.

Wiener was arguably the first to study intelligence *as a system*, rather than focusing on isolated components or aspects of intelligence. His comprehensive views appeared in the celebrated 1948 book *Cybernetics: or Control and Communication in the Animal and the Machine* [Wie48]. In that book and its second edition published in 1961 [Wie61] (see Figure 1.4), he attempted to identify several necessary characteristics and mechanisms of intelligent systems, including

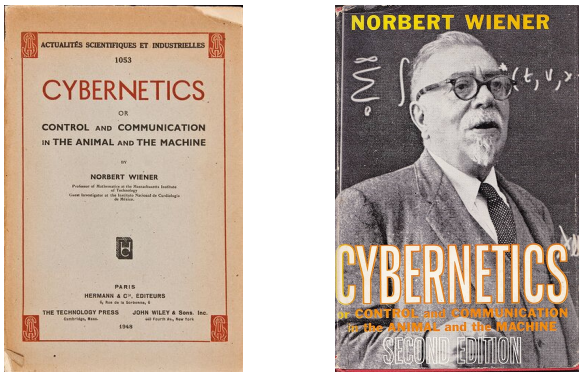


Figure 1.4: Norbert Wiener's book "Cybernetics" (1948) [Wie48] (left) and its second edition (1961) [Wie61] (right).

(but not limited to):

- How to *measure and store* information (in the brain) and how to communicate with others.[5] This insight led Claude Shannon to formulate *information theory* in 1948 [Sha48].

- How to *correct errors* in prediction and estimation based on existing information. Wiener himself helped formalize the theory of (closed-loop) feedback control in the 1940s.

- How to learn to *make better decisions* when interacting with a non-cooperative or even adversarial environment. John von Neumann formalized this as *game theory* in 1944 [NMR44].

In 1943, inspired by Wiener's Cybernetics program, cognitive scientist Warren McCulloch and logician Walter Pitts jointly formalized the first computational model of a neuron [MP43], called an *artificial neuron* and illustrated later in Figure 1.15. Building on this model, Frank Rosenblatt constructed the Mark I Perceptron in the 1950s—a physical machine containing hundreds of such artificial neurons [Ros57]. The Perceptron was the first physically realized artificial neural network; see Figure 1.17. Notably, John von Neumann's universal computer architecture, proposed in 1945, was also designed to facilitate the goal of building *computing machines* that could physically realize the mechanisms suggested by the Cybernetics program [Neu58].

Astute readers will notice that the 1940s were truly magical: many fundamental ideas were invented and influential theories formalized, including the mathematical model of neurons, artificial neural networks, information theory, control theory, game theory, and computing machines. Figure 1.5 portrays some

[5] Wiener was the first to point out that "information" is neither matter nor energy, but an independent quantity worthy of study.

Figure 1.5: Pioneers of theoretical and computational foundations for intelligence: Norbert Wiener (cybernetics and control theory), Claude Shannon (information theory), John von Neumann (game theory), and Alan Turing (computing theory).

of the pioneers. Each of these contributions has grown to become the foundation of a scientific or engineering field and continues to have tremendous impact on our lives. All were inspired by the goal of developing machines that emulate intelligence in nature. Historical records show that Wiener's Cybernetics movement influenced nearly all of these pioneers and works. To a large extent, Wiener's program can be viewed as the true predecessor of today's "embodied intelligence" program; in fact, Wiener himself articulated a vision for such a program with remarkable clarity and concreteness [Wie61].

Although Wiener identified many key characteristics and mechanisms of (embodied) intelligence, he offered no clear recipe for integrating them into a complete autonomous intelligent system. From today's perspective, some of his views were incomplete or inaccurate. In particular, in the last chapter of the second edition of *Cybernetics* [Wie61], he stressed the need to *deal with nonlinearity* if machines are to emulate typical learning mechanisms in nature, yet he provided no concrete, effective solution for this difficult issue. In fairness, even the theory of linear systems was in its infancy at the time,[6] and nonlinear systems seemed far less approachable.

Nevertheless, we cannot help but marvel at Wiener's prescience about the importance of nonlinearity. As this book will show, the answer came only recently: nonlinearity can be handled effectively through progressive linearization and transformation realized by deep networks and representations (see Chapter 5). Moreover, we will demonstrate in this book how all the mechanisms listed above can be naturally integrated into a complete system which exhibits certain characteristics of an autonomous intelligent system (see Chapter 6).

Origin of artificial intelligence. The subtitle of Wiener's *Cybernetics—Control and Communication in the Animal and the Machine*—reveals that 1940s research aimed primarily at emulating animal-level intelligence. As noted ear-

[6] Wiener's famous achievement, the Wiener filter [Wie42; Wie49], which we will introduce later, was only for linear systems.

lier, the agendas of that era were dominated by Wiener's Cybernetics movement.

Alan Turing was among the first to recognize this limitation. In his celebrated 1950 paper "*Computing Machinery and Intelligence*" [Tur50], Turing formally posed the question of "can machines think?" In other words, whether machines could imitate human-level intelligence, to the point of machine intelligence being indistinguishable from human intelligence—now known as *the Turing test*.

Around 1955, a group of ambitious young scientists sought to break away from the then-dominant Cybernetics program and establish their own legacy. They accepted Turing's challenge of imitating human intelligence and proposed a workshop at Dartmouth College to be held in the summer of 1956. Their proposal stated [MMR+06]:

> "*The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves.*"

They aimed to formalize and study the higher-level intelligence that distinguishes humans from animals. Their agenda included abstraction, symbolic methods, natural language, and deductive reasoning (causal inference, logical deduction, etc.), i.e., scientific intelligence. The organizer of the workshop, John McCarthy—then a young assistant professor of mathematics at Dartmouth College—coined the now-famous term "Artificial Intelligence" (AI) to describe machines that exhibit scientific intelligence in order to formally describe the problems and goals of the workshop.

Renaissance of "artificial intelligence" or "cybernetics"? Over the past decade, machine intelligence has undergone explosive development, driven largely by deep artificial neural networks, sparked by the 2012 work of Geoffrey Hinton and his students [KSH12]. This period is hailed as the "Renaissance of AI." Yet, in terms of the tasks actually tackled (recognition, generation, prediction) and the techniques developed (reinforcement learning, imitation learning, encoding, decoding, denoising, and compression), we are largely emulating mechanisms common to the intelligence of early life and animals. Even in this regard, as we will clarify in this book, current "AI" models and systems have not fully or correctly implemented all necessary mechanisms for intelligence at the animal level which were known to the 1940s Cybernetics movement.

Strictly speaking, the recent advances of machine intelligence in the past decade do not align well with the 1956 Dartmouth AI program. Instead, what has been achieved is closer to the objectives of Wiener's classic 1940s Cybernetics program. It is probably more appropriate to call the current era the "Renaissance of Cybernetics." The recent rise of so-called "Embodied AI" for autonomous robots and agents aligns even more closely with the goals of the

Cybernetics program. Only after we fully understand, from scientific and mathematical perspectives, what we have truly accomplished can we determine what problems remains open and which directions lead to the true nature of intelligence.[7] That is one of the main purposes of this book.

1.2 What to Learn?

1.2.1 Predictability

Data that carry useful information manifest in many forms. In their most natural form, they can be modeled as sequences that are predictable and computable. The notion of a predictable and computable sequence was central to the theory of information [Sha48] and computing and largely led to the invention of computers [Tur36]. The role of predictable sequences for (inductive) inference was studied by Ray Solomonoff, Andrey Kolmogorov, and others in the 1960s [Kol98] as a generalization of Claude Shannon's classic information theory [Sha48]. To help readers understand the concept of predictable sequences, we begin with some concrete examples.

Scalar case. The simplest predictable discrete sequence is arguably the sequence of natural numbers:

$$S = 1, 2, 3, 4, 5, 6, \ldots, n, n+1, \ldots \tag{1.2.1}$$

in which the next number x_{n+1} is defined as the previous number x_n plus 1:

$$x_{n+1} = x_n + 1. \tag{1.2.2}$$

One may generalize the notion of predictability to any sequence $\{x_n\}_{n=1}^{\infty}$ with $x_n \in \mathbb{R}$ if the next number x_{n+1} can always be computed from its predecessor x_n:

$$x_{n+1} = f(x_n), \quad x_n \in \mathbb{R},\ n = 1, 2, 3, \ldots \tag{1.2.3}$$

where $f(\cdot)$ is a *computable* (scalar) function.[8] Alan Turing's seminal work in 1936 [Tur36] gives a rigorous definition of computability. In practice, we often further assume that f is efficiently computable and has nice properties such as continuity and differentiability. The necessity of these properties will become clear later once we understand more refined notions of computability and their roles in machine learning and intelligence.

Multivariable case. The next value may also depend on two predecessors. For example, the famous *Fibonacci sequence*

$$S = 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots \tag{1.2.4}$$

[7] So we will resume our discussion at the end of this book in the last Chapter 9.

[8] Here we emphasize that the function $f(\cdot)$ itself is computable, meaning it can be implemented as a program on a computer.

satisfies

$$x_{n+2} = x_{n+1} + x_n, \quad x_n \in \mathbb{R},\ n = 1, 2, 3, \ldots \tag{1.2.5}$$

More generally, we may write

$$x_{n+2} = f(x_{n+1}, x_n), \quad x_n \in \mathbb{R},\ n = 1, 2, 3, \ldots \tag{1.2.6}$$

for any computable function f that takes two inputs. Extending further, the next value may depend on the preceding d values:

$$x_{n+d} = f(x_{n+d-1}, \ldots, x_n), \quad x_n \in \mathbb{R},\ n = 1, 2, 3, \ldots \tag{1.2.7}$$

The integer d is called the *degree* of the recursion. The above expression (1.2.7) is called an *autoregression*, and the resulting sequence is *autoregressive*. When f is linear, we say it is a *linear autoregression*.

Vector case. To simplify notation, we define a vector $\boldsymbol{x} \in \mathbb{R}^d$ that collects d consecutive values in the sequence:

$$\boldsymbol{x}_n \doteq [x_{n+d-1}, \ldots, x_n]^\top, \quad \boldsymbol{x}_n \in \mathbb{R}^d,\ n = 1, 2, 3, \ldots \tag{1.2.8}$$

With this notation, the recursive relation (1.2.7) becomes

$$\boldsymbol{x}_{n+1} = g(\boldsymbol{x}_n) \in \mathbb{R}^d, \quad n = 1, 2, 3, \ldots \tag{1.2.9}$$

where $g(\cdot)$ is uniquely determined by the function f in (1.2.7) and maps a d-dimensional vector to a d-dimensional vector. In different contexts, such a vector is sometimes called a "state" or a "token." Note that (1.2.7) defines a mapping $\mathbb{R}^d \to \mathbb{R}$, whereas here we have $g\colon \mathbb{R}^d \to \mathbb{R}^d$.

Controlled prediction. We may also define a predictable sequence that depends on another predictable sequence as input:

$$\boldsymbol{x}_{n+1} = f(\boldsymbol{x}_n, \boldsymbol{u}_n) \in \mathbb{R}^d, \quad n = 1, 2, 3, \ldots, \tag{1.2.10}$$

where $\{\boldsymbol{u}_n\}$ with $\boldsymbol{u}_n \in \mathbb{R}^k$ is a (computable) predictable sequence. In other words, the next vector $\boldsymbol{x}_{n+1} \in \mathbb{R}^d$ depends on both $\boldsymbol{x}_n \in \mathbb{R}^d$ and $\boldsymbol{u}_n \in \mathbb{R}^k$. In control theory, the sequence $\{\boldsymbol{u}_n\}$ is often referred to as the "control input" and $\boldsymbol{x}_n$ as the "state" or "output" of the system (1.2.10). A classic example is a linear dynamical system:

$$\boldsymbol{x}_{n+1} = \boldsymbol{A}\boldsymbol{x}_n + \boldsymbol{B}\boldsymbol{u}_n, \quad \boldsymbol{A} \in \mathbb{R}^{d\times d}, \boldsymbol{B} \in \mathbb{R}^{d\times k}, \tag{1.2.11}$$

which is widely studied in control theory [CD91].

Often the control input is given by a computable function of the state $\boldsymbol{x}_n$ itself:

$$\boldsymbol{u}_n = h(\boldsymbol{x}_n), \quad n = 1, 2, 3, \ldots \tag{1.2.12}$$

As a result, the sequence $\{\boldsymbol{x}_n\}$ is given by composing the two computable functions f and h:

$$\boldsymbol{x}_{n+1} = f(\boldsymbol{x}_n, h(\boldsymbol{x}_n)), \quad n = 1, 2, 3, \ldots \tag{1.2.13}$$

In this way, the sequence $\{\boldsymbol{x}_n\}$ again becomes an autoregressive predictable sequence. When the input $\boldsymbol{u}_n$ depends on the output $\boldsymbol{x}_n$, we say the resulting sequence is produced by a "closed-loop" system (1.2.13). As the closed-loop system no longer depends on any external input, we say such a system has become *autonomous*. It can be viewed as a special case of autoregression. For instance, if we choose $\boldsymbol{u}_n = \boldsymbol{F}\boldsymbol{x}_n$ in the above linear system (1.2.11), the closed-loop system becomes

$$\boldsymbol{x}_{n+1} = \boldsymbol{A}\boldsymbol{x}_n + \boldsymbol{B}\boldsymbol{u}_n = \boldsymbol{A}\boldsymbol{x}_n + \boldsymbol{B}\boldsymbol{F}\boldsymbol{x}_n = (\boldsymbol{A} + \boldsymbol{B}\boldsymbol{F})\boldsymbol{x}_n, \tag{1.2.14}$$

which is a linear autoregression.

Continuous processes. Predictable sequences have natural continuous counterparts, which we call predictable processes. The simplest such process is time itself, $x(t) = t$.

More generally, a process $\boldsymbol{x}(t)$ is *predictable* if, at every time t, its value at $t + \delta t$ is determined by its value at t, where δt is an infinitesimal increment. Typically, $\boldsymbol{x}(t)$ is continuous and smooth, so the change $\delta\boldsymbol{x}(t) = \boldsymbol{x}(t+\delta t) - \boldsymbol{x}(t)$ is infinitesimally small. Such processes are typically described by (multivariate) differential equations

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t)), \quad \boldsymbol{x} \in \mathbb{R}^d. \tag{1.2.15}$$

In systems theory [CD91; Sas99], the equation (1.2.15) is known as a state-space model. A controlled process is given by

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)), \quad \boldsymbol{x} \in \mathbb{R}^d, \boldsymbol{u} \in \mathbb{R}^k, \tag{1.2.16}$$

where $\boldsymbol{u}(t)$ is a computable input process.

Example 1.1. Newton's second law predicts the trajectory $\boldsymbol{x}(t) \in \mathbb{R}^3$ of a moving object under a force $\boldsymbol{F}(t) \in \mathbb{R}^3$:

$$m\ddot{\boldsymbol{x}}(t) = \boldsymbol{F}(t). \tag{1.2.17}$$

When there is no force, i.e., $\boldsymbol{F}(t) \equiv \boldsymbol{0}$, this reduces to Newton's first law: the object moves at constant velocity $\boldsymbol{v} \in \mathbb{R}^3$:

$$\ddot{\boldsymbol{x}}(t) = \boldsymbol{0} \iff \text{there exists } \boldsymbol{v} \in \mathbb{R}^3 \text{ such that } \dot{\boldsymbol{x}}(t) = \boldsymbol{v}. \tag{1.2.18}$$

■

1.2.2 Low Dimensionality

Learning to predict. Now suppose you have observed or have been given many sequence segments:

$$\{S_1, S_2, \ldots, S_i, \ldots, S_N\} \tag{1.2.19}$$

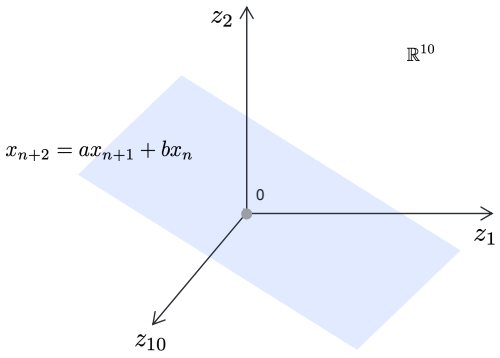


Figure 1.6: A two-dimensional subspace in a ten-dimensional ambient space.

drawn from a predictable sequence $\{x_n\}_{n=1}^{\infty}$. Without loss of generality, assume each segment has length $D \gg d$, so

$$S_i = [x_{j(i)}, x_{j(i)+1}, \ldots, x_{j(i)+D-1}]^\top \in \mathbb{R}^D \tag{1.2.20}$$

for some $j \in \mathbb{N}$. You are then given a new segment S_t and asked to predict its future values.

The difficulty is that the generating function f and its order d are unknown:

$$x_{n+d} = f(x_{n+d-1}, \ldots, x_n). \tag{1.2.21}$$

The goal is therefore to learn f and d from the sample segments $S_1, S_2, \ldots, S_N$. The central task of learning to predict is:

Given many sampled segments of a predictable sequence, how can we effectively and efficiently identify the function f?

Predictability and low-dimensionality. To identify the predictive function f, we may notice a common characteristic of segments of any predictable sequence given by (1.2.21). If we take a long segment, say of length $D \gg d$, and view it as a vector

$$\boldsymbol{x}_i = [x_i, x_{i+1}, \ldots, x_{i+D-1}]^\top \in \mathbb{R}^D, \tag{1.2.22}$$

then the set of all such vectors $\{\boldsymbol{x}_i\}$ is far from random and cannot occupy the entire space $\mathbb{R}^D$. Instead, it has at most d degrees of freedom—given the first d entries of any $\boldsymbol{x}_i$, the remaining entries are uniquely determined. In other words, all $\{\boldsymbol{x}_i\}$ lie on a d-dimensional surface. In mathematics, such a surface is called a submanifold, denoted $\mathcal{S} \subset \mathbb{R}^D$.

In practice, if we choose the segment length D large enough, all segments sampled from the same predicting function lie on a surface with intrinsic dimension d, significantly lower than that of the ambient space D. For example,

if the sequence is given by the linear autoregression

$$x_{n+2} = ax_{n+1} + bx_n, \tag{1.2.23}$$

for some constants $a, b \in \mathbb{R}$, and we sample segments of length $D = 10$, then all samples lie on a two-dimensional plane in $\mathbb{R}^{10}$, as illustrated in Figure 1.6. Identifying this two-dimensional subspace fully determines the constants a and b in (1.2.23).

More generally, when the predicting function f is linear, as in the systems given in (1.2.11) and (1.2.14), the long segments always lie on a low-dimensional linear subspace. Identifying the predicting function is then largely equivalent to identifying this subspace, a problem known as principal component analysis. We will discuss such classic models and methods in Chapter 2.

This observation extends to general predictable sequences: if we can identify the low-dimensional surface on which the segment samples lie, we can identify the predictive function f.[9] We cannot overemphasize the importance of this property: *All samples of long segments of a predictable sequence lie on a low-dimensional submanifold.* To a large extent, modern science and mathematics are precisely trying to identify and describe such low-dimensional structure of observed phenomena. This explains why the most important and famous scientific discoveries about our physical world are often described in terms of equations, as Figure 1.7 illustrated, because equations restrict the otherwise independent observed quantities onto lower-dimensional manifolds.

According the String Theory that unifies almost all physical laws we know about the world, the entire universe evolves in a space of 10 dimension, starting from the Big Bang, see Figure 1.8. As we will see in this book, all modern learning methods also exploit the low-dimensionality property in data, either implicitly or explicitly.

Remark 1.1 (Entropy for Physics versus that for Intelligence). In Figure 1.7, we see that the only physical law that is *not* described in the form of an equation is the Second Law of Thermodynamics which states that the entropy of any closed physical system, like the whole universe, always *increases*. We will see in this book that the goal of intelligence (and science) is to identify structures in observed phenomena that can *decrease* the entropy in our prediction. This will be studied rather extensively in Chapter 3 and Appendix B.

In real-world scenarios, observed data often come from multiple predictable sequences. For example, a video sequence may contain several moving objects. In such cases, the data lie on a mixture of low-dimensional linear subspaces or nonlinear submanifolds, as illustrated in Figure 1.9.

Properties of low-dimensionality. Temporal correlation in predictable sequences is not the only reason data are low-dimensional. For example, the space

[9] Under mild conditions, there is a one-to-one mapping between the low-dimensional surface and the function f. This fact has been exploited in problems such as system identification, which we will discuss later.

17 Equations That Changed the World
by Ian Stewart

1.	**Pythagoras's Theorem**	$a^2 + b^2 = c^2$	Pythagoras,530 BC
2.	**Logarithms**	$\log xy = \log x + \log y$	John Napier, 1610
3.	**Calculus**	$\frac{\mathrm{d}f}{\mathrm{d}t} = \lim_{h \to 0} \frac{f(t+h) - f(t)}{h}$	Newton, 1668
4.	**Law of Gravity**	$F = G\frac{m_1 m_2}{r^2}$	Newton, 1687
5.	**The Square Root of Minus One**	$i^2 = -1$	Euler, 1750
6.	**Euler's Formula for Polyhedra**	$V - E + F = 2$	Euler, 1751
7.	**Normal Distribution**	$\Phi(x) = \frac{1}{\sqrt{2\pi\rho}} e^{\frac{(x-\mu)^2}{2\rho^2}}$	C.F. Gauss, 1810
8.	**Wave Equation**	$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$	J. d'Almbert, 1746
9.	**Fourier Transform**	$f(\omega) = \int_{\infty}^{\infty} f(x) e^{-2\pi i x \omega} \mathrm{d}\,x$	J. Fourier, 1822
10.	**Navier-Stokes Equation**	$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}\right) = -\nabla p + \nabla \cdot \mathbf{T} + \mathbf{f}$	C. Navier, G. Stokes, 1845
11.	**Maxwell's Equations**	$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_*}$ $\quad \nabla \cdot \mathbf{H} = 0$ $\nabla \times \mathbf{E} = -\frac{1}{c}\frac{\partial \mathbf{H}}{\partial t}$ $\quad \nabla \times \mathbf{H} = \frac{1}{c}\frac{\partial E}{\partial t}$	J.C. Maxwell, 1865
12.	**Second Law of Thermodynamics**	$dS \geq 0$	L. Boltzmann, 1874
13.	**Relativity**	$E = mc^2$	Einstein, 1905
14.	**Schrodinger's Equation**	$i\hbar\frac{\partial}{\partial t}\Psi = H\Psi$	E. Schrodinger, 1927
15.	**Information Theory**	$H = -\sum p(x) \log p(x)$	C. Shannon, 1949
16.	**Chaos Theory**	$x_{t+1} = kx_t(1 - x_t)$	Robert May, 1975
17.	**Black-Scholes Equation**	$\frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} + \frac{\partial V}{\partial t} - rV = 0$	F. Black, M. Scholes, 1990

Figure 1.7: Famous 17 equations that changed the world. Most of these equations describe physical laws of the universe. For intelligence, the equation 15 of Entropy will play a very special role, as we will explain why in this book, especially in Chapter 3.

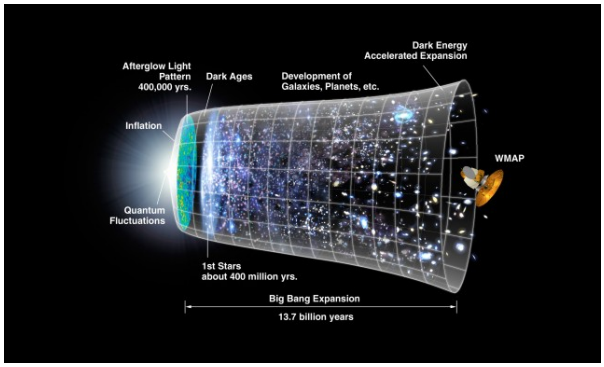


Figure 1.8: The evolution of the Universe that is, based on the best physics theory so far, believed to be in a space of 10 dimensions.

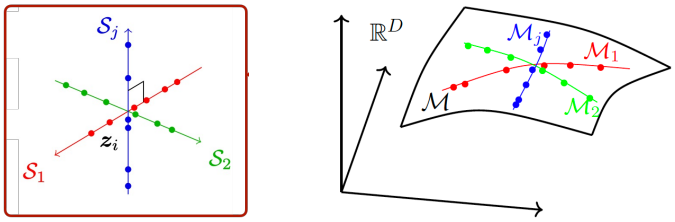


Figure 1.9: Data distributed on a mixture of (orthogonal) subspaces (left) or submanifolds (right).

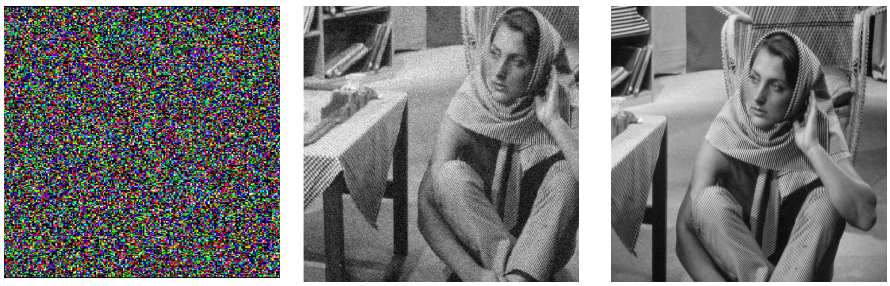


Figure 1.10: An image of random noise (left) versus a noisy image (middle) and the original clean image (right).

of all images is vast, yet most of it consists of structureless random images, as shown in Figure 1.10 (left). Natural images and videos, however, are highly redundant because of strong spatial and temporal correlations among pixel values. This redundancy allows us to recognize easily whether an image is noisy or clean, as shown in Figure 1.10 (middle and right). Consequently, the distribution of natural images has a very low intrinsic dimension relative to the total number of pixels.

Because learning low-dimensional structures is both important and ubiquitous, the book *High-Dimensional Data Analysis with Low-Dimensional Models: Principles, Computation, and Applications* [WM22] begins with the statement: "*The problem of identifying the low-dimensional structure of signals or data in high-dimensional spaces is one of the most fundamental problems that, through a long history, interweaves many engineering and mathematical fields such as system theory, signal processing, pattern recognition, machine learning, and statistics.*"

By constraining the observed data point $\boldsymbol{x}$ to lie on a low-dimensional surface or submanifold, we make its entries highly dependent on or correlated to one another and, in a sense, "predictable" from the values of other entries. For example, if we know the data are constrained to a d-dimensional surface in $\mathbb{R}^D$, we can perform several useful tasks beyond prediction:

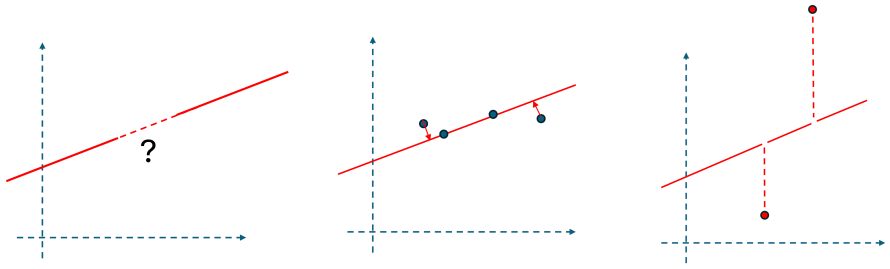


Figure 1.11: Illustration of properties of a low-dimensional (linear) structure: it enables completion (left), denoising (middle), and error correction (right).

- **completion**: given more than d entries of a typical sample $\boldsymbol{x}$, the remaining entries can usually be uniquely determined;[10]
- **denoising**: if the entries of a sample $\boldsymbol{x}$ are perturbed by small noise, the noise can be effectively removed by projecting $\boldsymbol{x}$ back onto the surface;
- **error correction**: if a small number of unknown entries of $\boldsymbol{x}$ are arbitrarily corrupted, they can be efficiently corrected.

Figure 1.11 illustrates these properties using a low-dimensional linear structure—a one-dimensional line in a two-dimensional plane.

Under mild conditions, these properties generalize to many other low-dimensional structures in high-dimensional spaces [WM22]. As we will see, these useful properties—completion and denoising, for example—inspire effective methods for learning such structures.

For simplicity, we have so far used the deterministic case to introduce the notions of predictability and low-dimensionality, where data lie precisely on geometric structures such as subspaces or surfaces. In practice, however, data always contain some uncertainty or randomness. In this case, we may assume the data follow a probability distribution with density $p(\boldsymbol{x})$. A distribution is considered "low-dimensional" if its density concentrates around a low-dimensional geometric structure—a subspace, a surface, or a mixture thereof—as shown in Figure 1.9. This structure is also called the "support" of the distribution. Once learned, such a density $p(\boldsymbol{x})$ serves as a powerful prior for estimating $\boldsymbol{x}$ from partial, noisy, or corrupted observations:

$$\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{n}, \tag{1.2.24}$$

by computing the conditional estimate $\hat{\boldsymbol{x}}(\boldsymbol{y}) = \mathbb{E}(\boldsymbol{x} \mid \boldsymbol{y})$ or by sampling the conditional distribution $\hat{\boldsymbol{x}}(\boldsymbol{y}) \sim p(\boldsymbol{x} \mid \boldsymbol{y})$.[11]

[10] Prediction becomes a special case of this property.

[11] Modern generative AI technologies, such as conditioned image generation, rely heavily on this principle, as we will discuss in Chapter 7.

Many prior works have studied the properties of data sets whose distributions are supported on a low-dimensional surface or submanifold, e.g., [ACM12; LMR17]. We here would like to single out its ultimate importance and claims it as the single assumption on which this book bases its deductive approach to understanding deep networks and intelligence in general:

> **The main assumption:** *Any intelligent system or learning method should (and can) rely on the predictability of the world; hence, the distribution of observed high-dimensional data samples has low-dimensional support.*

The remaining question is how to learn these low-dimensional structures correctly and computationally efficiently from high-dimensional data. As we will see, parametric models studied in classical analytical approaches and non-parametric models such as deep networks, popular in modern practice, are simply different means to the same end.

1.3 How to Learn?

1.3.1 Analytical Model-Based Approaches

Note that even if a predictive function is tractable to compute, it does not imply that it is tractable or scalable to learn this function from a finite number of sampled segments. One classical approach to ensure tractability is to make explicit assumptions about the family of low-dimensional structures we are dealing with. Historically, due to limited computation and data, simple and idealistic analytical models were the first to be studied, as they often offer efficient closed-form or numerical solutions. In addition, they provide insights into more general problems and already yield useful solutions to important, though limited, cases. In the old days, when computational resources were scarce, analytical models that permitted efficient closed-form or numerical solutions were the only cases that could be implemented. *Linear structures* became the first class of models to be thoroughly studied.

For example, arguably the simplest case is to assume the data are distributed around a single low-dimensional subspace in a high-dimensional space. Somewhat equivalently, one may assume the data are distributed according to an almost degenerate low-dimensional Gaussian. Identifying such a subspace or Gaussian from a finite number of (noisy) samples is the classical problem of principal component analysis (PCA), and effective algorithms have been developed for this class of models [Jol02]. One can make the family of models increasingly more complex and expressive. For instance, one may assume the data are distributed around a mixture of low-dimensional components (subspaces or low-dimensional Gaussians), as in independent component analysis (ICA) [BJC85], dictionary learning (DL), generalized principal component analysis (GPCA) [VMS05], or the more general class of sparse low-dimensional models that have been studied extensively in recent years in fields such as compressive sensing [WM22].

Across all these analytical model families, the central problem is to identify the most compact model within each family that best fits the given data. Below, we give a brief account of these classical analytical models but leave a more systematic study to Chapter 2. In theory, these analytical models have provided tremendous insights into the geometric and statistical properties of low-dimensional structures. They often yield closed-form solutions or efficient and scalable algorithms, which are very useful for data whose distributions can be well approximated by such models. More importantly, for more general problems, they provide a sense of how easy or difficult the problem of identifying low-dimensional structures can be and what the basic ideas are to approach such a problem.

Linear Dynamical Systems

Wiener filter. As discussed in Section 1.2.1, a central task of intelligence is to learn what is predictable in sequences of observations. The simplest class of predictable sequences—or signals—are those generated by a *linear time-invariant* (LTI) process:

$$x[n] = h[n] * z[n] + \epsilon[n], \tag{1.3.1}$$

where $*$ is the convolution operation, z is the input and h is the impulse response function.[12] Here $\epsilon[n]$ denotes additive observation noise. Given the input process $\{z[n]\}$ and observations of the output process $\{x[n]\}$, the goal is to find the optimal $h[n]$ such that $\hat{x}[n] = h[n] * z[n]$ predicts $x[n]$ optimally. Prediction quality (i.e., goodness) is measured by the mean squared error (MSE):

$$\min_h \mathbb{E}[\epsilon[n]^2] = \mathbb{E}[\|x[n] - h[n] * z[n]\|_2^2]. \tag{1.3.2}$$

The optimal solution $h[n]$ is called a (denoising) filter. Norbert Wiener—who also initiated the Cybernetics movement—studied this problem in the 1940s and derived an elegant closed-form solution known as the *Wiener filter* [Wie42; Wie49]. This result, also known as least-variance estimation and filtering, became one of the cornerstones of signal processing. Interested readers may refer to [MKS+04, Appendix B] for a detailed derivation of this type of estimator.

Kalman filter. The idea of denoising or filtering a dynamical system was later extended by Rudolph Kalman in the 1960s to a linear time-invariant system described by a finite-dimensional state-space model:

$$\boldsymbol{z}[n] = \boldsymbol{A}\boldsymbol{z}[n-1] + \boldsymbol{B}\boldsymbol{u}[n] + \boldsymbol{\epsilon}[n]. \tag{1.3.3}$$

The problem is to estimate the system state $\boldsymbol{z}[n]$ from noisy observations of the form

$$\boldsymbol{x}[n] = \boldsymbol{C}\boldsymbol{z}[n] + \boldsymbol{w}[n], \tag{1.3.4}$$

[12] Typically, h is assumed to have certain desirable properties, such as finite length or a band-limited spectrum.

where $\boldsymbol{w}$ is (white) noise. The optimal causal[13] state estimator that minimizes the minimum-MSE (MMSE) prediction error

$$\min \mathbb{E}[\|\boldsymbol{x}[n] - \boldsymbol{C}\boldsymbol{z}[n]\|_2^2] \tag{1.3.5}$$

is given in closed form by the so-called *Kalman filter* [Kal60]. This result is a cornerstone of modern control theory because it enables estimation of a dynamical system's state from noisy observations. One can then introduce linear state feedback, for example $\boldsymbol{u}[n] = \boldsymbol{F}\hat{\boldsymbol{z}}[n]$, and render the closed-loop system fully autonomous, as shown in Equation (1.2.13). Interested readers may refer to [MKS+04, Appendix B] for a detailed derivation of the Kalman filter.

Identification of linear dynamical systems. To derive the Kalman filter, the system parameters $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ are assumed to be known. If they are not given in advance, the problem becomes more challenging and is known as *system identification*: how to *learn* the parameters $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ from many samples of the input sequence $\{\boldsymbol{u}[n]\}$ and observation sequence $\{\boldsymbol{x}[n]\}$. This is a classic problem in systems theory. If the system is linear, the input and output sequences $\{\boldsymbol{u}[n], \boldsymbol{x}[n]\}$ jointly lie on a certain low-dimensional subspace[14]. Hence, the identification problem is essentially equivalent to identifying this low-dimensional subspace [LV09; LV10; VM96].

Note that the above problems have two things in common: first, the noise-free sequences or signals are assumed to be generated by an explicit family of parametric models; second, these models are essentially linear. Conceptually, let $\boldsymbol{x}_o$ be a random variable whose "true" distribution is supported on a low-dimensional linear subspace, say S. To a large extent, the Wiener filter and Kalman filter both try to estimate such an $\boldsymbol{x}_o$ from its noisy observations:

$$\boldsymbol{x} = \boldsymbol{x}_o + \boldsymbol{\epsilon}, \quad \boldsymbol{x}_o \sim S, \tag{1.3.6}$$

where $\boldsymbol{\epsilon}$ is typically a random Gaussian noise (or process). Hence, their solutions rely on identifying a low-dimensional linear subspace that best fits the observed noisy data. By projecting the data onto this subspace, one obtains the optimal denoising operations, all in closed form.

Linear and Mixed Linear Models

Principal component analysis. From the above problems in classical signal processing and system identification, we see that the main task behind all these problems is to learn a *single* low-dimensional linear subspace from noisy observations. Mathematically, we may model such a structure as

$$\boldsymbol{x} = \boldsymbol{u}_1 z_1 + \boldsymbol{u}_2 z_2 + \cdots + \boldsymbol{u}_d z_d + \boldsymbol{\epsilon} = \boldsymbol{U}\boldsymbol{z} + \boldsymbol{\epsilon}, \quad \boldsymbol{U} \in \mathbb{R}^{D \times d}, \tag{1.3.7}$$

where $\boldsymbol{\epsilon} \in \mathbb{R}^D$ is small random noise. Figure 1.12 illustrates such a distribution with two principal components. The problem is to find the subspace basis $\boldsymbol{U}$

[13] This means the estimator can use only observations up to the current time step n. The Kalman filter is always causal, whereas the Wiener filter need not be.

[14] which has the same dimension as the order of the state-space model (1.3.3).

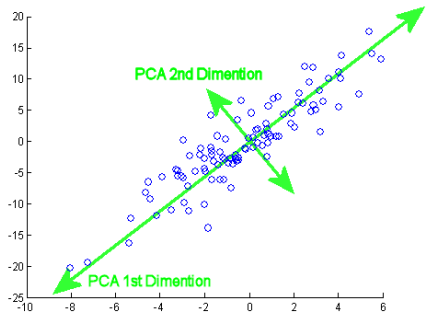


Figure 1.12: A distribution with two principal components.

from many samples of $\boldsymbol{x}$. A typical approach is to minimize the projection error onto the subspace:

$$\min_{\boldsymbol{U}} \mathbb{E}[\|\boldsymbol{x} - \boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{z}\|_2^2]. \tag{1.3.8}$$

This is essentially a denoising task: once the basis $\boldsymbol{U}$ is correctly found, we can denoise the noisy sample $\boldsymbol{x}$ by projecting it onto the low-dimensional subspace spanned by $\boldsymbol{U}$:

$$\boldsymbol{x} \to \hat{\boldsymbol{x}} = \boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{x}. \tag{1.3.9}$$

If the noise is small and the low-dimensional subspace $\boldsymbol{U}$ is correctly learned, we expect $\boldsymbol{x} \approx \hat{\boldsymbol{x}}$. Thus, PCA is a special case of a so-called "auto-encoding" scheme, which encodes the data by projecting it into a lower-dimensional space, and decodes a lower-dimensional code by it into the original space:

$$\boldsymbol{x} \xrightarrow{\boldsymbol{U}^\top} \boldsymbol{z} \xrightarrow{\boldsymbol{U}} \hat{\boldsymbol{x}}. \tag{1.3.10}$$

Because of the simple data structure, the encoder $\mathcal{E}$ and decoder $\mathcal{D}$ both become simple linear operators (projecting and lifting).

This classic problem in statistics is known as principal component analysis (PCA). It was first studied by Pearson in 1901 [Pea01] and later independently by Hotelling in 1933 [Hot33]. The topic is systematically summarized in the classic book [Jol02; Jol86]. One may also explicitly assume the data $\boldsymbol{x}$ is distributed according to a single low-dimensional Gaussian:

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{U}\boldsymbol{U}^\top + \sigma \boldsymbol{I}), \quad \boldsymbol{U} \in \mathbb{R}^{D \times d}, \tag{1.3.11}$$

which is equivalent to assuming that $\boldsymbol{z}$ in the PCA model (1.3.7) is a standard normal distribution. This problem is known as probabilistic PCA [TB99] and has the same computational solution as PCA.

In this book, we revisit PCA in Chapter 2 from the perspective of learning a low-dimensional distribution. Our goal is to use this simple and idealistic model to convey some of the most fundamental ideas for learning a compact representation of a low-dimensional distribution, including the important notions of compression via denoising and auto-encoding for a consistent representation.

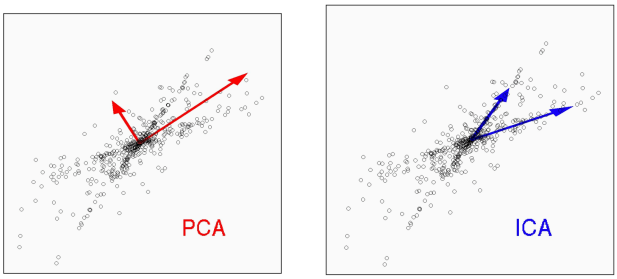


Figure 1.13: PCA (left) versus ICA (right). Note that PCA finds the *orthogonal* vectors that approximately span the data, while ICA finds (not necessarily orthogonal) vectors that *independently* combine to approximately form the data, as well as the coefficients of this combination.

Independent component analysis. Independent component analysis (ICA) was originally proposed by [BJC85] as a classic model for *learning a good representation*. In fact, it was originally proposed as a simple mathematical model for our memory. The ICA model takes a deceptively similar form to the above PCA model (1.3.7) by assuming that the observed random variable $\boldsymbol{x}$ is a linear superposition of multiple independent components z_i:[15]

$$\boldsymbol{x} = \boldsymbol{u}_1 z_1 + \boldsymbol{u}_2 z_2 + \cdots + \boldsymbol{u}_d z_d + \boldsymbol{\epsilon} = \boldsymbol{U}\boldsymbol{z} + \boldsymbol{\epsilon}. \tag{1.3.12}$$

However, here the components z_i are assumed to be independent *non-Gaussian* variables. For example, a popular choice is

$$z_i = \sigma_i \cdot w_i, \quad \sigma_i \sim \mathrm{Ber}(p), \tag{1.3.13}$$

where σ_i is a Bernoulli random variable and w_i could be a constant value or another random variable, say Gaussian.[16] The ICA problem aims to identify both $\boldsymbol{U}$ and $\boldsymbol{z}$ from observed samples of $\boldsymbol{x}$. Figure 1.13 illustrates the difference between ICA and PCA.

Although the (decoding) mapping from $\boldsymbol{z}$ to $\boldsymbol{x}$ seems linear and straightforward once $\boldsymbol{U}$ and $\boldsymbol{z}$ are learned, the (encoding) mapping from $\boldsymbol{x}$ to $\boldsymbol{z}$ can be complicated and may not be represented by a simple linear mapping. Hence, ICA generally gives an auto-encoding of the form:

$$\boldsymbol{x} \xrightarrow{\mathcal{E}} \boldsymbol{z} \xrightarrow{\boldsymbol{U}} \hat{\boldsymbol{x}}. \tag{1.3.14}$$

Thus, unlike PCA, ICA is somewhat more difficult to analyze and solve. In the 1990s, researchers such as Erkki Oja and Aapo Hyvärinen [HO97; HO00b] made significant theoretical and algorithmic contributions to ICA. In Chapter 2, we

[15] In PCA the "components" are the learned vectors, whereas in ICA they are scalars. This is just a difference in names and convention.

[16] Even if w is Gaussian, σw is no longer a Gaussian variable!

will study and provide a solution to ICA from which the encoding mapping $\mathcal{E}$ will become clear. In fact, Aapo Hyvärinen proposed the general score matching method in 2005 [Hyv05], in part to learn distributions such as a mixture of Gaussian like in ICA. We will discuss more about this later and again in Chapter 3 in great details.

Sparse structures and compressive sensing. If p in (1.3.13) is very small, the probability that any component is non-zero is small. In this case we say $\boldsymbol{x}$ is sparsely generated and concentrates on a union of linear subspaces whose dimension is $k = p \cdot d$. We may therefore extend the ICA model to a more general family of low-dimensional structures known as sparse models.

A k-sparse model is the set of all *k-sparse vectors*:

$$\mathcal{Z} = \{\boldsymbol{z} \in \mathbb{R}^n \mid \|\boldsymbol{z}\|_0 \leq k\}, \tag{1.3.15}$$

where $\|\cdot\|_0$ counts the number of non-zero entries. Thus $\mathcal{Z}$ is the union of all k-dimensional subspaces aligned with the coordinate axes, as illustrated in Figure 1.9 (left). A central problem in signal processing and statistics is to recover a sparse vector $\boldsymbol{z}$ from its linear observations

$$\boldsymbol{x} = \boldsymbol{A}\boldsymbol{z} + \boldsymbol{\epsilon}, \quad \boldsymbol{A} \in \mathbb{R}^{m \times n}, \tag{1.3.16}$$

where $\boldsymbol{A}$ is given, typically $m < n$, and $\boldsymbol{\epsilon}$ is small noise. This seemingly benign problem is NP-hard to solve and even hard to approximate (see [WM22] for details).

Despite a rich history dating back to the eighteenth century [Bos50], no provably efficient algorithm existed for this class of problems, although many heuristic algorithms were proposed between the 1960s and 1990s. Some were effective in practice but lacked rigorous justification. A major breakthrough came in the early 2000s when mathematicians including David Donoho, Emmanuel Candés, and Terence Tao [CT05a; CT05b; Don05] established a rigorous theoretical framework that characterizes precise conditions under which the sparse recovery problem can be solved effectively and efficiently via convex ℓ^1 minimization:

$$\min \|\boldsymbol{z}\|_1 \quad \text{subject to} \quad \|\boldsymbol{x} - \boldsymbol{A}\boldsymbol{z}\|_2 \leq \epsilon, \tag{1.3.17}$$

where $\|\cdot\|_1$ is the sparsity-promoting ℓ^1 norm of a vector and ϵ is a small constant. Any solution yields a sparse (auto) encoding

$$\boldsymbol{x} \xrightarrow{\mathcal{E}} \boldsymbol{z} \xrightarrow{\boldsymbol{A}} \hat{\boldsymbol{x}}. \tag{1.3.18}$$

We will describe such an algorithm (and thus mapping) in Chapter 2, revealing fundamental connections between sparse coding and deep learning.[17]

Conditions for ℓ^1 minimization to succeed are surprisingly general: the minimum number of measurements m required for a successful recovery is only

[17] Similarities between sparse-coding algorithms and deep networks were noted as early as 2010 [GL10].

proportional to the intrinsic dimension k. This is the *compressive sensing* phenomenon [Can06]. It extends to broad families of low-dimensional structures, including low-rank matrices. These results fundamentally changed our understanding of recovering low-dimensional structures in high-dimensional spaces. David Donoho, among others, celebrated this reversal as the "blessing of dimensionality" [D D00], in contrast to the usual pessimistic belief in the "curse of dimensionality." The complete theory and body of results is presented in [WM22].

The computational significance of this framework cannot be overstated. It transformed problems previously believed intractable into ones that are not only tractable but scalably solvable using extremely efficient algorithms:

$$\text{intractable} \implies \text{tractable} \implies \text{scalable}. \tag{1.3.19}$$

The algorithms come with rigorous theoretical guarantees of correctness given precise requirements in data and computation. The deductive nature of this approach contrasts sharply with the empirical, inductive practice of deep neural networks. Yet we now know that both approaches share a common goal—pursuing low-dimensional structures in high-dimensional spaces.

Dictionary learning. Conceptually, an even harder problem than the sparse-coding problem (1.3.16) arises when the observation matrix $\boldsymbol{A}$ is unknown and must itself be learned from a set of (possibly noisy) observations $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]$:

$$\boldsymbol{X} = \boldsymbol{A}\boldsymbol{Z} + \boldsymbol{E}, \quad \boldsymbol{A} \in \mathbb{R}^{m \times n}. \tag{1.3.20}$$

Here we are given only $\boldsymbol{X}$, not the corresponding $\boldsymbol{Z} = [\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_n]$ or the noise term $\boldsymbol{E} = [\boldsymbol{\epsilon}_1, \boldsymbol{\epsilon}_2, \ldots, \boldsymbol{\epsilon}_n]$, except that the $\boldsymbol{z}_i$ are known to be sparse. This is the *dictionary learning* problem, which generalizes the ICA problem (1.3.12) discussed earlier. In other words, given that the distribution of the data $\boldsymbol{X}$ is the image of a standard sparse distribution $\boldsymbol{Z}$ under a linear transform $\boldsymbol{A}$, we wish to learn both $\boldsymbol{A}$ and its "inverse" mapping $\mathcal{E}$ so as to obtain an autoencoder:

$$\boldsymbol{X} \xrightarrow{\ \mathcal{E}\ } \boldsymbol{Z} \xrightarrow{\ \boldsymbol{A}\ } \hat{\boldsymbol{X}}. \tag{1.3.21}$$

PCA, ICA, and dictionary learning all assume that the data distribution is supported on or near low-dimensional linear or piecewise-linear structures. Each method requires learning a (global or local) basis for these structures from noisy samples. In Chapter 2 we study how to identify such low-dimensional structures through these classical models. In particular, we will see that all of these low-dimensional (piecewise) linear models can be learned efficiently by the same family of fast algorithms known as *power iteration* [ZMZ+20]. Although these linear or piecewise-linear models are somewhat idealized for most real-world data, understanding them is an essential first step toward understanding more general low-dimensional distributions.

General Distributions

The distributions of real-world data such as images, videos, and audio are too complex to be modeled by the above, somewhat idealistic, linear models or Gaussian processes. We normally do not know *a priori* from which family of parametric models they are generated. Historically, many attempts have been made to develop analytical models for these data. In particular, Fields Medalist David Mumford spent considerable effort in the 1990s trying to understand and model the statistics of natural images [Mum96]. He and his students, including Song-Chun Zhu, drew inspiration and techniques from statistical physics and proposed many statistical and stochastic models for the distribution of natural images, such as random fields or stochastic processes [HM99; LPM03; MG99; ZM97a; ZWM97; ZM97b]. However, these analytical models achieved only limited success in producing samples that closely resemble natural images. Clearly, for real-world data like images, we need to develop more powerful and unifying methods to pursue their more general low-dimensional structures and obtain efficient representations, e.g., for encoding and decoding purposes [ACM12; LMR17].

In practice, for a general distribution of real-world data, we typically only have many samples from the distribution—the so-called *empirical distribution*. In such cases, we normally cannot expect to have a clear analytical form for their low-dimensional structures, nor for the resulting denoising operators.[18] We therefore need to develop a more general solution to these empirical distributions, not necessarily in closed form but at least efficiently computable. If we do this correctly, solutions to the aforementioned linear models should become their special cases.

Denoising. In the 1950s, statisticians became interested in the problem of denoising data drawn from an arbitrary distribution. Let $\boldsymbol{x}_o$ be a random variable with probability density function $p_o(\cdot)$. Suppose we observe a noisy version of $\boldsymbol{x}_o$:

$$\boldsymbol{x} = \boldsymbol{x}_o + \sigma \boldsymbol{g}, \tag{1.3.22}$$

where $\boldsymbol{g} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ is standard Gaussian noise and σ is the noise level of the observation. Let $p(\cdot)$ be the probability density function of $\boldsymbol{x}$.[19] Remarkably, the posterior expectation of $\boldsymbol{x}_o$ given $\boldsymbol{x}$ can be calculated by an elegant formula, known as Tweedie's formula [Rob56]:[20]

$$\hat{\boldsymbol{x}}_o = \mathbb{E}[\boldsymbol{x}_o \mid \boldsymbol{x}] = \boldsymbol{x} + \sigma^2 \nabla \log p(\boldsymbol{x}). \tag{1.3.23}$$

As can be seen from the formula, the function $\nabla \log p(\boldsymbol{x})$ plays a very special role in denoising the observation $\boldsymbol{x}$ here. The noise $\boldsymbol{g}$ can be explicitly estimated

[18] This is unlike the cases of PCA, Wiener filter, and Kalman filter.

[19] That is, $p(\boldsymbol{x}) = \int_{-\infty}^{\infty} \varphi_\sigma(\boldsymbol{x} - \boldsymbol{z}) p_o(\boldsymbol{z}) \mathrm{d}\boldsymbol{z}$, where φ_σ is the density function of the Gaussian distribution $\mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$.

[20] Herbert Robbins gave the credit for this formula to Maurice Kenneth Tweedie from their personal correspondence.

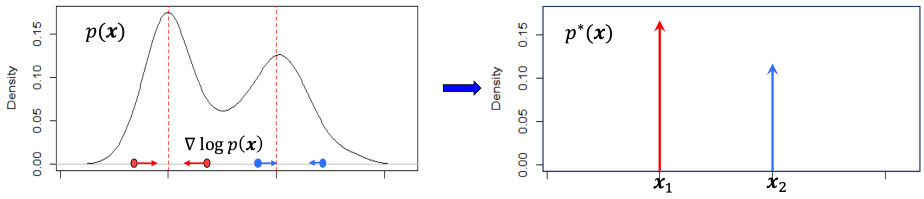


Figure 1.14: Geometric interpretation of a score function $\nabla \log p(\boldsymbol{x})$ for a distribution with density $p(\boldsymbol{x})$ on the left. The operation generated by the score function pushes the distribution towards areas of higher density. The goal is that, by a certain measure of compactness (e.g. entropy or coding length), the resulting distribution is more "compressed". Eventually, the distribution converges to one that has a lower-dimensional support, as $p^*(\boldsymbol{x})$ shown on the right.

as

$$\hat{\boldsymbol{g}} = \frac{\boldsymbol{x} - \hat{\boldsymbol{x}}_o}{\sigma} = -\sigma \nabla \log p(\boldsymbol{x}), \tag{1.3.24}$$

for which we only need to know the distribution $p(\cdot)$ of $\boldsymbol{x}$, not the ground truth $p_o(\cdot)$ for $\boldsymbol{x}_o$. An important implication of this result is that if we add Gaussian noise to any distribution, the denoising process can be done easily if we can somehow obtain the function $\nabla \log p(\boldsymbol{x})$.

Because this is such an important and useful result, it has been rediscovered and used in many different contexts and areas. For example, after Tweedie's formula [Rob56], it was rediscovered a few years later by [Miy61] where it was termed "empirical Bayesian denoising." In the early 2000s, the function $\nabla \log p(\boldsymbol{x})$ was rediscovered again in the context of learning a general distribution and was named the "score function" by Aapo Hyvärinen [Hyv05]. Its connection to (empirical Bayesian) denoising was soon recognized by [Vin11]. Generalizations to other measurement distributions (beyond Gaussian noise) have been made by Eero Simoncelli's group [RS11]. Today, the most direct application of Tweedie's formula and denoising is image generation via iterative denoising [HJA20; KS21].

Entropy minimization. The score has an intuitive information-theoretic and geometric interpretation. In information theory, $-\log p(\boldsymbol{x})$ corresponds to the number of bits needed to encode $\boldsymbol{x}$[21]. The gradient $\nabla \log p(\boldsymbol{x})$ points toward higher-probability-density regions, as shown in Figure 1.14 (left). Moving in this direction reduces the number of bits required to encode $\boldsymbol{x}$. Thus, the operator $\nabla \log p(\boldsymbol{x})$ pushes the distribution to "shrink" toward high-density regions. Formally, one can show that the (differential) entropy

$$H(\boldsymbol{x}) = -\int p(\boldsymbol{w}) \log p(\boldsymbol{w}) \mathrm{d}\boldsymbol{w} \tag{1.3.25}$$

[21] at least for discrete variables, as detailed in Chapter 3.

decreases under this operation (see Chapter 3 and Chapter B). With an optimal codebook, the resulting distribution achieves a lower coding rate and is thus more compressed. Repeating this denoising process indefinitely produces a distribution whose probability mass is concentrated on a support of lower dimension. For instance, the distribution $p(\boldsymbol{x})$ in Figure 1.14 (left) converges to $p^*(\boldsymbol{x})$ (right):[22][23]

$$H(\boldsymbol{x}) = -\int p(\boldsymbol{w}) \log p(\boldsymbol{w}) \mathrm{d}\boldsymbol{w} \quad \xrightarrow{\text{decreasing}} \quad H^*(\boldsymbol{x}) = -\int p^*(\boldsymbol{w}) \log p^*(\boldsymbol{w}) \mathrm{d}\boldsymbol{w}. \tag{1.3.26}$$

As the distribution converges to $p^*(\boldsymbol{x})$, its differential entropy approaches negative infinity due to a technical difference between continuous and discrete entropy definitions. Chapter 3 resolves this using a unified *rate-distortion* measure.

Later in this chapter and in Chapters 3 and 4, we explore how this simple denoising-compression framework unifies powerful methods for learning low-dimensional distributions in high-dimensional spaces, including natural image distributions.

1.3.2 Empirical Data-Driven Approaches

In practice, it is difficult to model important real-world data—such as images, sounds, and text—with the idealized linear, piecewise-linear, or other analytical models discussed in the previous section. Historically, many non-parametric models and methods have therefore been proposed to model or capture the empirical distribution of the data and then use such "models" for inference. These models, especially deep neural networks, often drew inspiration from the biological nervous system, because the brain of an animal or human processes such data with remarkable efficiency and effectiveness. Compared to the analytical models introduced in the previous section, these data-driven models are the least understood despite the fact that they have very much dominated the practice of AI technology in the past decade or so. Hence, one of the main goals of this book is to provide a principled and unified framework (very much through Chapters 5 to 8) to understand both these "modern" empirical models and the classic analytical models and reveal fundamental connections between them.

Classic Artificial Neural Networks

Artificial neuron. Inspired by the nervous system in the brain, the first mathematical model of an artificial neuron[24] was proposed by Warren McCulloch[25] and Walter Pitts in 1943 [MP43]. It describes the relationship between

[22] Strictly speaking, $p^*(\boldsymbol{x})$ is a generalized function: $p^*(\boldsymbol{x}) = p^*(\boldsymbol{x}_1)\delta(\boldsymbol{x}-\boldsymbol{x}_1)+p^*(\boldsymbol{x}_2)\delta(\boldsymbol{x}-\boldsymbol{x}_2)$ with $p^*(\boldsymbol{x}_1)+p^*(\boldsymbol{x}_2)=1$.

[23] Notice that in this section we discuss the process of iteratively denoising and compressing a high-entropy noise distribution until it converges to the low-entropy data distribution. As we will see in Chapter 3, this problem is dual to the problem of learning an optimal encoding for the data distribution, which is also a useful application [RAG+24].

[24] known as the Linear Threshold Unit, or perceptron.

[25] a professor of psychiatry at the University of Chicago at the time

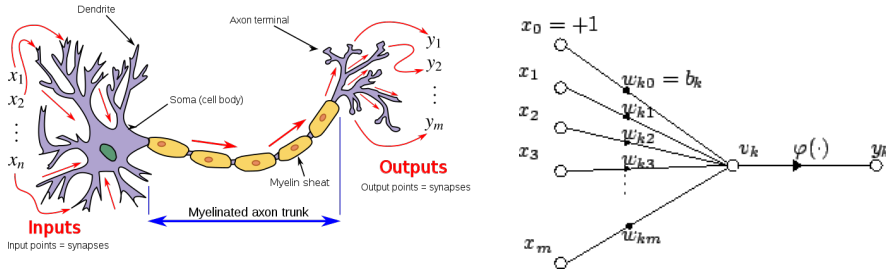


Figure 1.15: The first mathematical model of an artificial neuron (right) that emulates how a neuron (left) processes signals.

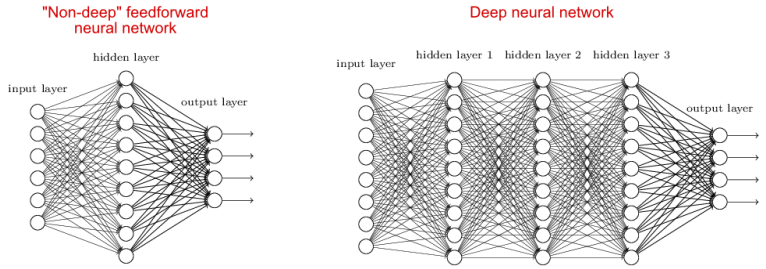


Figure 1.16: A network with one hidden layer (left) versus a deep network (right).

the input x_i and output o_j as:

$$o_j = \varphi\Big(\sum_i w_{ji} x_i\Big), \tag{1.3.27}$$

where $\varphi(\cdot)$ is some nonlinear activation, typically modeled by a threshold function. This model is illustrated in Figure 1.15. As we can see, this form already shares the main characteristics of a basic unit in modern deep neural networks. The model is derived from observations of how a single neuron works in our nervous system. However, researchers did not know exactly what functions a collection of such neurons could realize and perform. On a more technical level, they were also unsure which nonlinear activation function $\varphi(\cdot)$ should be used. Hence, historically many variants have been proposed.[26]

Artificial neural network. In the 1950s, Frank Rosenblatt was the first to build a machine with a *network* of such artificial neurons, shown in Figure 1.17. The machine, called Mark I Perceptron, consists of an input layer, an output

[26] Step function, hard or soft thresholding, rectified linear unit (ReLU), sigmoid, etc. [DSC22].

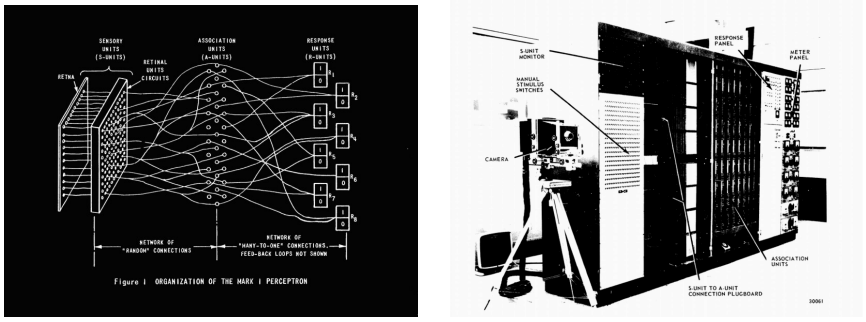


Figure 1.17: The Mark I Perceptron machine developed by Frank Rosenblatt in the late 1950s.

layer, and a single hidden layer of 512 artificial neurons, as shown in Figure 1.17 left, which is similar to what is illustrated in Figure 1.16 (left). It was designed to classify optical images of letters. However, the capacity of a single-layer network is limited and can only learn linearly separable patterns. In the 1969 book *Perceptrons: An Introduction to Computational Geometry* by Marvin Minsky and Seymour Papert [MP69], it was shown that the single-layer architecture of Mark I Perceptron cannot learn an XOR function. This result significantly dampened interest in artificial neural networks, even though it was later proven that a multi-layer network can learn an XOR function [RHW86a]. In fact, a sufficiently large multi-layer network, as shown in Figure 1.16 (right), consisting of such simple neurons can simulate any finite-state machine, even the universal Turing machine.[27] Nevertheless, the study of artificial neural networks subsequently entered its first winter in the 1970s.

Convolutional neural networks. Early experiments with artificial neural networks such as the Mark I Perceptron in the 1950s and 1960s were somewhat disappointing. They suggested that simply connecting neurons in a general fashion, as in multi-layer perceptrons (MLPs), might not suffice. To build effective and efficient networks, it is extremely helpful to understand the collective purpose or function neurons in the network must achieve so that they can be organized and learned in a specialized way. Thus, at this juncture, once again the study of machine intelligence turned to the animal nervous system for inspiration.

It is known that most of our brain is dedicated to processing visual information [Pla99]. In the 1950s and 1960s, David Hubel and Torsten Wiesel systematically studied the visual cortices of cats. They discovered that the visual cortex contains different types of cells—simple cells and complex cells—which are sensitive to visual stimuli of different orientations and locations [HW59]. Hubel and Wiesel won the 1981 Nobel Prize in Physiology or Medicine for this

[27] Do not confuse what neural networks are capable of doing in principle with whether it is tractable or easy to learn a neural network that realizes certain desired functions.

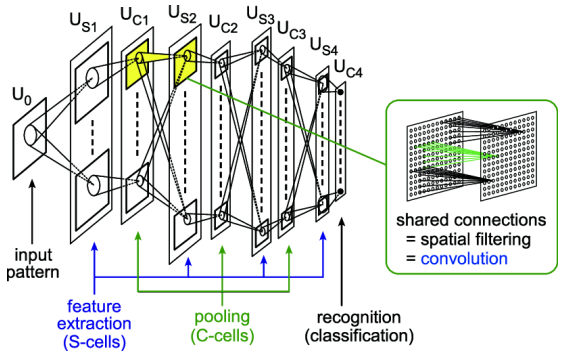


Figure 1.18: Origin of convolutional neural networks: the Neocognitron by Kunihiko Fukushima in 1980. Notice that the interleaving layers of convolutions and pooling emulate the functions of simple cells and complex cells discovered in the visual cortices of cats.

groundbreaking discovery.

On the artificial neural network side, Hubel and Wiesel's work inspired Kunihiko Fukushima to design the "neocognitron" network in 1980, which consists of artificial neurons that emulate biological neurons in the visual cortices [Fuk80]. This is known as the first *convolutional neural network* (CNN), and its architecture is illustrated in Figure 1.18. Unlike the perceptron, the neocognitron had more than one hidden layer and could be viewed as a deep network, as shown in Figure 1.16 (right).

Also inspired by how neurons work in the cat's visual cortex, Fukushima was the first to introduce the *rectified linear unit* (ReLU):

$$\varphi(x) = \max\{0, x\} = \begin{cases} x, & \text{if } x > 0, \\ 0, & \text{if } x \le 0, \end{cases} \tag{1.3.28}$$

as the activation function $\varphi(\cdot)$ in 1969 [Fuk69]. However, it was not until recent years that ReLU became widely used in modern deep (convolutional) neural networks. This book will explain why ReLU is a good choice once we discuss the main operations deep networks implement: compression.

CNN-type networks continued to evolve in the 1980s, and many different variants were introduced and studied. However, despite the remarkable capacities of deep networks and the improved architectures inspired by neuroscience, it remained extremely difficult to train such deep networks for real tasks such as image classification. Getting a network to work depended on many unexplainable heuristics and tricks, which limited the appeal and applicability of neural networks. A major breakthrough came around 1989 when Yann LeCun successfully used *back propagation* (BP) to learn a deep convolutional neural network for recognizing handwritten digits [LBD+89], later known as LeNet (see Figure 1.19). After several years of persistent development, his perseverance paid off: LeNet's performance eventually became good enough for practical use in

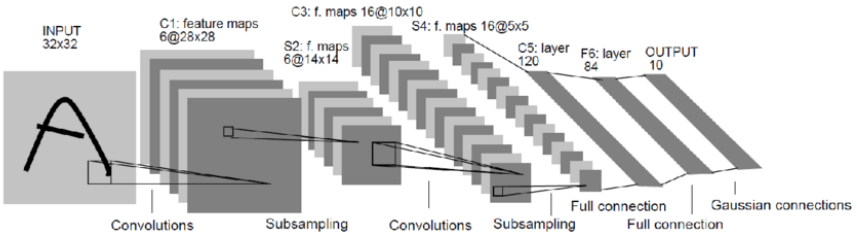


Figure 1.19: The LeNet-5 convolutional neural network designed by Yann LeCun in 1989.

the late 1990s [LBB+98a]. It was used by the US Post Office for recognizing handwritten digits (for zip codes). LeNet was considered the "prototype" network for all modern deep neural networks, such as AlexNet [KSH12] and ResNet [HZR+16b], which we will discuss later. For this work, Yann LeCun was awarded the 2018 Turing Award.[28]

Backpropagation. Throughout history, the fate of deep neural networks has been tied to how easily and efficiently they can be trained. Backpropagation (BP) was introduced for this purpose. A multilayer perceptron can be expressed as a composition of linear mappings and nonlinear activations:

$$h(\boldsymbol{W}_1, \ldots, \boldsymbol{W}_L) = f^L(\boldsymbol{W}_L f^{L-1}(\boldsymbol{W}_{L-1} \cdots f^2(\boldsymbol{W}_2 f^1(\boldsymbol{W}_1 \boldsymbol{x})))). \tag{1.3.29}$$

To train the network weights $\{\boldsymbol{W}_l\}_{l=1}^L$ via gradient descent, we must evaluate the gradient $\partial h / \partial \boldsymbol{W}_l$. The chain rule in calculus shows that gradients can be computed efficiently for such functions—a technique later termed backpropagation or BP; see Chapter A for details. BP was already known in optimal control and dynamic programming during the 1960s and 1970s, appearing in Paul Werbos's 1974 PhD thesis [Wer74; Wer94]. In 1986, David Rumelhart et al. were the first to apply it to train a multilayer perceptron (MLP) [RHW86a]. Since then, BP has become the dominant technique for training deep networks, as it is scalable and can be efficiently implemented on parallel and distributed computing platforms. However, nature likely does not use BP,[29] as the mechanism is too expensive for physical implementation.[30] This leaves ample room for future improvement.

Despite the aforementioned algorithmic advances, training deep networks remained finicky and computationally expensive in the 1980s and 1990s. By the

[28] Together with two other pioneers of deep networks, Yoshua Bengio and Geoffrey Hinton.

[29] As we have discussed earlier, nature almost ubiquitously learns to correct errors via closed-loop feedback.

[30] End-to-end BP is computationally intractable for neural structures as complex as the brain, especially if the updates need to happen in real time. Instead, localized updates to small sections of the neural circuitry are much more biologically feasible. There is a relatively small amount of work on transplanting such "local learning" rules to training deep networks, circumventing BP [BS16; LTP25; MSS+22]. We believe this is an exciting opportunity to improve the scalability of training deep networks.

late 1990s, support vector machines (SVMs) [CV95] had gained popularity as a superior alternative for classification tasks.[31] SVMs offered two advantages: a rigorous statistical learning framework known as the Vapnik–Chervonenkis (VC) theory and efficient convex optimization algorithms [BV04]. The rise of SVMs ushered in a second winter for neural networks in the early 2000s.

Compressive auto-encoding. In the late 1980s and 1990s, artificial neural networks were already being used to learn low-dimensional representations of high-dimensional data such as images. It had been shown that neural networks could learn PCA directly from data [BH89; Oja82], rather than using the classic methods discussed in Section 1.3.1. It was also argued during this period that, due to their ability to model nonlinear transformations, neural networks could learn low-dimensional representations for data with nonlinear distributions. Similar to the linear PCA case, one can simultaneously learn a nonlinear dimension-reduction encoder f and a decoder g, each modeled by a deep neural network [Kra91; RHW86a]:

$$\boldsymbol{X} \xrightarrow{\ f\ } \boldsymbol{Z} \xrightarrow{\ g\ } \hat{\boldsymbol{X}}. \tag{1.3.30}$$

By enforcing consistency between the decoded data $\hat{\boldsymbol{X}}$ and the original $\boldsymbol{X}$—for example, by minimizing a MSE-type reconstruction error[32]:

$$\min_{f,g} \|\boldsymbol{X} - \hat{\boldsymbol{X}}\|_2^2, \qquad \text{where} \quad \hat{\boldsymbol{X}} = g(f(\boldsymbol{X})), \tag{1.3.31}$$

an autoencoder can be learned directly from the data $\boldsymbol{X}$.

But how can we guarantee that such an auto-encoding indeed captures the true low-dimensional structures in $\boldsymbol{X}$ rather than yielding a trivial redundant representation? For instance, one could simply choose f and g to be identity maps and set $\boldsymbol{Z} = \boldsymbol{X}$. To ensure the auto-encoding is worthwhile, the resulting representation should be compressive according to some computable measure of complexity. In 1993, Geoffrey Hinton and colleagues proposed using coding length as such a measure, transforming the auto-encoding objective into finding a representation that minimizes coding length [HZ93]. This work established a fundamental connection between the principle of minimum description length [Ris78] and free (Helmholtz) energy minimization. Later work from Hinton's group [HS06] empirically demonstrated that such auto-encoding can learn meaningful low-dimensional representations for real-world images. Pierre Baldi provided a comprehensive survey of autoencoders in 2011 [Bal11], just before deep networks gained widespread popularity. We will discuss measures of complexity and auto-encoding further in Section 1.4, and present a systematic study of compressive auto-encoding in Chapter 6 from a more unified perspective.

[31] Similar ideas for classification problems trace back to Thomas Cover's 1964 PhD dissertation, which was condensed and published in a paper in 1964 [Cov64].

[32] MSE-type errors are known to be problematic for imagery data with complex nonlinear structures. As we will soon discuss, much recent work in generative methods, including GANs, has focused on finding better surrogate distance functions between the original data $\boldsymbol{X}$ and the regenerated $\hat{\boldsymbol{X}}$.

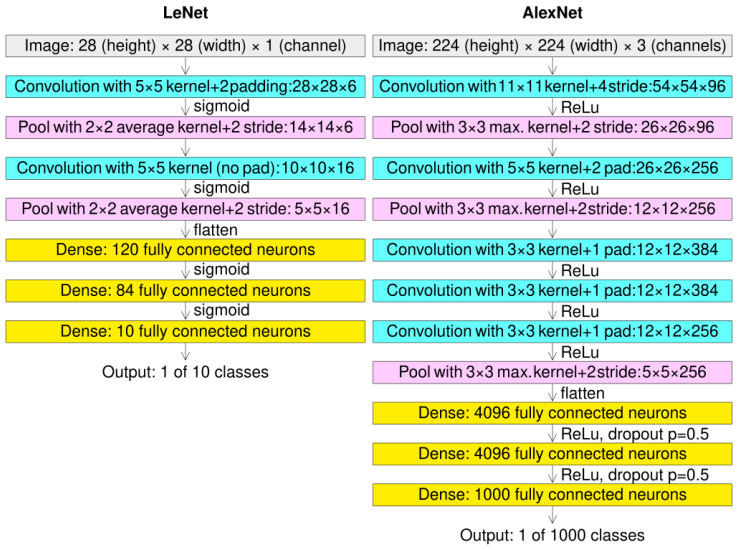


Figure 1.20: Architecture of LeNet [LBD+89] versus AlexNet [KSH12].

Modern Deep Neural Networks

For nearly 30 years—from the 1980s to the 2010s—neural networks were not taken seriously by the mainstream machine learning community. Early deep networks such as LeNet showed promising performance on small-scale classification problems like digit recognition, yet their design was largely empirical, the available datasets were tiny, and back-propagation was computationally prohibitive for the hardware of the era. These factors led to waning interest and stagnant progress, with only a handful of researchers persisting.

Classification and recognition. The tremendous potential of deep networks could be unleashed only once sufficient data and computational power became available. By the 2010s, large datasets such as ImageNet had emerged, and GPUs had become powerful enough to make back-propagation affordable even for networks far larger than LeNet. In 2012, the deep convolutional network AlexNet — named for Alex Krizhevsky, one of the authors [KSH12] — attracted widespread attention by surpassing existing classification methods on ImageNet by a significant margin.[33] Figure 1.20 compares AlexNet with LeNet. AlexNet retains many characteristics of LeNet—it is simply larger and replaces LeNet's sigmoid activations with ReLUs. This work contributed to Geoffrey Hinton's 2018 Turing Award.

This early success inspired the machine intelligence community to explore new neural network architectures, variations, and improvements. Empirically,

[33] Deep networks had already achieved state-of-the-art results on speech-recognition tasks, but these successes received far less attention than the breakthrough on image classification.

it was discovered that larger and deeper networks yield better performance on tasks such as image classification. Notable architectures that emerged include VGG [SZ15], GoogLeNet [SLJ+14], ResNet [HZR+16a], and, more recently, Transformers [VSP+17b]. Despite rapid empirically-driven performance improvements, theoretical explanations for these architectures—and the relationships among them, if any —remained scarce. One goal of this book is to uncover the common objective these networks optimize and to explain their shared characteristics, such as multiple layers of linear operators interleaved with nonlinear activations (see Chapter 5).

Reinforcement learning. Early deep network successes were mainly in supervised classification tasks such as speech and image recognition. Later, deep networks were adopted to learn decision-making or control policies for game playing. In this context, deep networks model the optimal decision/control policy (i.e., a probability distribution over actions to take in order to maximize the expected reward) or the associated optimal value function (an estimate of the expected reward from the given state), as shown in Figure 1.21. Network parameters are incrementally optimized[34] based on rewards returned from the success or failure of playing the game with the current policy. This learning paradigm is called *reinforcement learning* [SB18]; it originated in control-systems practice of the late 1960s [MM70; WF65] and traces back through a long and rich history to Richard Bellman's *dynamic programming* [Bel57] and Marvin Minsky's *trial-and-error learning* [Min54] in the 1950s.

From an implementation standpoint, the marriage of deep networks and reinforcement learning proved powerful: deep networks can approximate control policies and value functions for real-world environments that are difficult to model analytically. This culminated in DeepMind's AlphaGo system, which stunned the world in 2016 by defeating top Go player Lee Sedol and, in 2017, world champion Jie Ke.[35]

AlphaGo's success surprised the computing community, which had long regarded the game's state space as too vast for any efficient solution in terms of computation and sample size. The only plausible explanation is that the optimal value and policy function of Go possess significant favorable structure: qualitatively speaking, their intrinsic dimensions are low enough so that they can be well approximated by neural networks learnable from a manageable number of samples.

Generation and prediction. From a certain perspective, the early practice of deep networks in the 2010s was focused on extracting relevant information from the data $\boldsymbol{X}$ and encoding it into a task-specific representation $\boldsymbol{Z}$ (say $\boldsymbol{Z}$

[34] Typically via back-propagation (BP).

[35] In 1996, IBM's Deep Blue made history by defeating Russian grandmaster Garry Kasparov in chess using traditional machine learning techniques such as tree search and pruning, methods that have proven less scalable and unsuccessful for more complex games like Go.

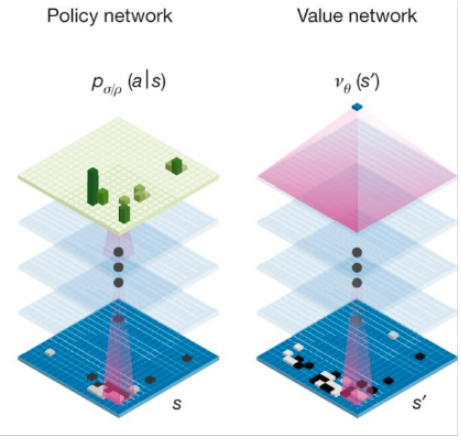


Figure 1.21: AlphaGo: using deep neural networks to model the optimal policy or value function for the game Go.

denotes class labels in classification tasks[36]):

$$\boldsymbol{X} \xrightarrow{f} \boldsymbol{Z}. \tag{1.3.32}$$

Here the learned mapping f needs not preserve most distributional information about $\boldsymbol{X}$; it suffices to retain only the sufficient statistics for the task. For example, a sample $\boldsymbol{x} \in \boldsymbol{X}$ might be an image of an apple, mapped by f to the class label $\boldsymbol{z} =$ "apple".

In many modern settings—such as the so-called large general-purpose ("foundation") models—we may need to also decode $\boldsymbol{Z}$ to recover the corresponding $\boldsymbol{X}$ to a prescribed precision:

$$\boldsymbol{Z} \xrightarrow{g} \hat{\boldsymbol{X}}. \tag{1.3.33}$$

Because $\boldsymbol{X}$ typically represents data observed from the external world, a good decoder allows us to simulate or predict what happens in the world. In a "text-to-image" or "text-to-video" task, for instance, $\boldsymbol{z}$ is the text describing the desired image $\boldsymbol{x}$, and the decoder should generate an $\hat{\boldsymbol{x}}$ whose content matches $\boldsymbol{x}$. Given an object class $\boldsymbol{z} =$ "apple", the decoder g should produce an image $\hat{\boldsymbol{x}}$ that looks like an apple, though not necessarily identical to the original $\boldsymbol{x}$.

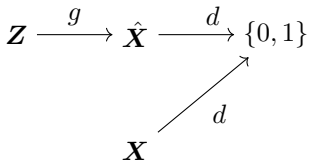
Generation via discriminative approaches. For the generated images $\hat{\boldsymbol{X}}$ to resemble true natural images $\boldsymbol{X}$, we must be able to evaluate and minimize some distance:

$$\min \operatorname{dist}(\boldsymbol{X}, \hat{\boldsymbol{X}}). \tag{1.3.34}$$

[36] This is a highly atypical notation for labels — the usual notation is $\boldsymbol{y}$ — but it is useful for our purposes to consider labels as highly compressed and sparse representations of the data. See Example 1.2 for more details.

As it turns out, most theoretically motivated distances are extremely difficult—if not impossible—to compute and optimize for distributions in high-dimensional space with low intrinsic dimension.[37]

In 2007, Zhuowen Tu, disappointed by early analytical attempts to model and generate natural images, decided to try a drastically different approach. In a paper published at CVPR 2007 [Tu07], he first suggested learning a generative model for images via a discriminative approach. The idea is simple: if evaluating the distance $\mathrm{dist}(\boldsymbol{X}, \hat{\boldsymbol{X}})$ proves difficult, one can instead learn a discriminator d to separate $\hat{\boldsymbol{X}}$ from $\boldsymbol{X}$:



where labels 0 and 1 indicate whether an image is generated or real. Intuitively, the harder it becomes to separate $\hat{\boldsymbol{X}}$ and $\boldsymbol{X}$, the closer they likely are.

Tu's work [Tu07] first demonstrated the feasibility of learning a generative model from a discriminative approach. However, the work employed traditional methods for image generation and distribution classification (such as boosting), which proved slow and difficult to implement. After 2012, deep neural networks gained popularity for image classification. In 2014, Ian Goodfellow and colleagues again proposed generating natural images with a discriminative approach [GPM+14a]. They suggested modeling both the generator g and discriminator d with deep neural networks. Moreover, they proposed learning g and d via a minimax game:

$$\min_{g} \max_{d} \ell(\boldsymbol{X}, \hat{\boldsymbol{X}}), \tag{1.3.35}$$

where $\ell(\cdot)$ is some natural loss function associated with the classification task. In this formulation, the discriminator d maximizes its success in separating $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$, while the generator g does the opposite. This approach is named *generative adversarial networks* (GANs). It was shown that once trained on a large dataset, GANs can indeed generate photo-realistic images. Partially due to this work's influence, Yoshua Bengio received the 2018 Turing Award.

The discriminative approach appears to cleverly bypass a fundamental difficulty in distribution learning. However, rigorously speaking, this approach does not fully resolve the fundamental difficulty. It is shown in [GPM+14a] that with a properly chosen loss, the minimax formulation becomes mathematically equivalent to minimizing the Jensen-Shannon distance (see [CT91]) between $\boldsymbol{X}$

[37] This remains true even when a parametric family of distributions for $\boldsymbol{X}$ is specified. The distance often becomes ill-conditioned or ill-defined for distributions with low-dimensional supports. Worse still, the chosen family might poorly approximate the true distribution of interest.

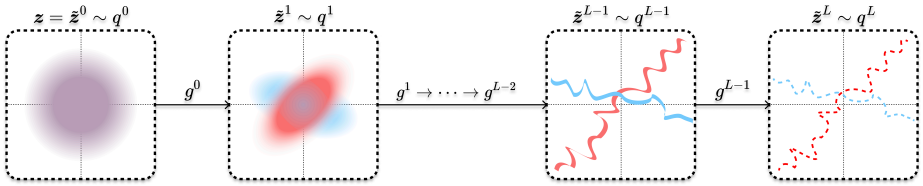


Figure 1.22: Illustration of an iterative denoising and compressing process that, starting from a Gaussian distribution $q^0 = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, converges to an arbitrary low-dimensional data distribution $q^L = p$.

and $\hat{\boldsymbol{X}}$. This remains a hard problem for two low-dimensional distributions in high-dimensional space. Consequently, GANs typically rely on many heuristics and engineering tricks and often suffer from instability issues such as mode collapsing.[38]

Generation via denoising and diffusion. In 2015, shortly after GANs were introduced and gained popularity, Surya Ganguli and his students proposed that an iterative denoising process modeled by a deep network could be used to learn a general distribution, such as that of natural images [SWM+15]. Their method was inspired by properties of special Gaussian and binomial processes studied by William Feller in 1949 [Fel49].[39]

Soon, denoising operators based on the score function [Hyv05], briefly introduced in Section 1.3.1, were shown to be more general and unified the denoising and diffusion processes and algorithms [HJA20; SE19; SSK+21]. Figure 1.22 illustrates the process that transforms a generic Gaussian distribution $q^0 = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ to an (arbitrary) empirical distribution $p(\boldsymbol{x})$ by performing a sequence of iterative denoising (or compressing) operations:

$$\boldsymbol{z}^0 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \xrightarrow{g^0} \boldsymbol{z}^1 \xrightarrow{g^1} \cdots \xrightarrow{g^{L-1}} \boldsymbol{z}^L \sim p(\boldsymbol{x}). \tag{1.3.36}$$

By now, denoising (and diffusion) has replaced GANs and become the mainstream method for learning distributions of images and videos, leading to popular commercial image generation engines such as Midjourney and Stability.ai. In Chapter 3 we will systematically introduce and study the denoising and diffusion method for learning a general low-dimensional distribution.

1.4 A Unifying Approach

So far, we have given a brief account of the main objective and history of machine intelligence, along with many important ideas and approaches associated with

[38] Nevertheless, such a minimax formulation provides a practical approximation of the distance. It simplifies implementation and avoids certain caveats in directly computing the distance.

[39] Again, in the magical era of the 1940s!

it. In recent years, following the empirical success of deep neural networks, tremendous efforts have been made to develop theoretical frameworks that help us understand empirically designed deep networks—whether specific seemingly necessary components (e.g., dropout [SHK+14], normalization [BKH16; IS15], attention [VSP+17b]) or their overall behaviors (e.g., double descent [BHM+19], neural collapse [PHD20]).

Motivated in part by this trend, this book pursues several important and challenging goals:

- Develop a theoretical framework that yields rigorous mathematical interpretations of deep neural networks.
- Ensure correctness of the learned data distribution and consistency with the learned representation.
- Demonstrate that the framework leads to performant architectures and guides further practical improvements.

In recent years, mounting evidence suggests these goals can indeed be achieved by leveraging the theory and solutions of the classical analytical low-dimensional models discussed earlier (treated more thoroughly in Chapter 2) and by integrating fundamental ideas from related fields—namely information/coding theory, control/game theory, and optimization. This book aims to provide a systematic introduction to this new approach.

1.4.1 Learning Parsimonious Representations

One necessary condition for any learning task to be possible is that the sequences of interest must be *computable*, at least in the sense of Alan Turing [Tur36]. That is, a sequence can be computed via a program on a typical computer.[40] In addition to being computable, we require computation to be *tractable*.[41] That is, the computational cost (space and time) for learning and computing the sequence should not grow exponentially in length. Furthermore, as we see in nature (and in the modern practice of machine intelligence), for most practical tasks an intelligent system needs to learn what is predictable from massive data in a very high-dimensional space, such as from vision, sound, and touch. Hence, for intelligence we do not need to consider all computable and tractable sequences or structures; we should focus only on predictable sequences and structures that admit *scalable* realizations of their learning and computing algorithms:

$$\textbf{computable} \implies \textbf{tractable} \implies \textbf{scalable}. \tag{1.4.1}$$

This is because whatever algorithms intelligent beings use to learn useful information must be *scalable*. More specifically, the computational complexity of the algorithms should scale gracefully—typically linear or even sublinear—in

[40] There are indeed well-defined sequences that are not computable.

[41] We do not need to consider predicting things whose computational complexity is intractable—say, grows exponentially in the length or dimension of the sequence.

the size and dimension of the data. On the technical level, this requires that the operations the algorithms rely on to learn can only utilize oracle information that can be efficiently computed from the data. More specifically, when the dimension is high and the scale is large, the only oracle one can afford to compute is either the first-order geometric information about the data[42] or the second-order statistical information[43]. The main goal of this book is to develop a theoretical and computational framework within which we can systematically develop efficient and effective solutions or algorithms with such scalable oracles and operations to *learn* low-dimensional structures from the sampled data and subsequently the predictive function.

Pursuing low-dimensionality via compression. From the examples of sequences we gave in Section 1.2.1, it is clear that some sequences are easy to model and compute, while others are more difficult. The computational cost of a sequence depends on the complexity of the predicting function f. The higher the degree of regression d, the more costly it is to compute. The function f could be a simple linear function, or it could be a nonlinear function that is arbitrarily difficult to specify and compute.

It is reasonable to believe that if a sequence is harder—by whatever measure we choose—to specify and compute, then it will also be more difficult to learn from its sampled segments. Nevertheless, for any given predictable sequence, there are in fact many, often infinitely many, ways to specify it. For example, for the simple sequence $x_{n+1} = ax_n$, we could also define the same sequence with $x_{n+1} = ax_n + bx_{n-1} - bx_{n-1}$. Hence it would be very useful to develop an objective and rigorous notion of "complexity" for any given computable sequence.

Andrey Kolmogorov, a Russian mathematician, was one of the first to define complexity for any computable sequence.[44] He proposed that among all programs computing the same sequence, the length of the shortest program measures its complexity. This aligns with Occam's Razor—*choose the simplest theory explaining the same observation*. Formally, let p be a program generating sequence S on universal computer $\mathcal{U}$. The Kolmogorov complexity of S is:

$$K(S) = \min_{p\,:\,\mathcal{U}(p)=S} L(p). \tag{1.4.2}$$

Thus, complexity measures how parsimoniously we can specify or compute the sequence. This definition is conceptually important and historically inspired profound studies in computational complexity and theoretical computer science.

The shortest program length represents the ultimate compression of the sequence, quantifying our gain from learning its generative mechanism. However,

[42] such as approximating a nonlinear structure locally with linear subspaces and computing the gradient of an objective function.

[43] such as covariance or correlation of the data or their features. As we will see, popular deep architectures such as Transformers rely on this type of information.

[44] Many contributed to this notion of sequence complexity, most notably Ray Solomonoff and Greg Chaitin. All three developed algorithmic information theory independently—Solomonoff in 1960, Kolmogorov in 1965 [Kol98], and Chaitin around 1966 [Cha66].

Kolmogorov complexity is generally *uncomputable* [CT91] and intractable to approximate. Consequently, it has little practical use—it cannot predict learning difficulty or assess how well we have learned.

Computable measure of parsimony. For practical purposes, we need an efficiently computable measure of complexity for sequences generated by the same predicting function.[45] Note that part of the reason Kolmogorov complexity is not computable is that its definition is non-constructive.

To introduce a computable measure of complexity, we may take a more constructive approach, as advocated by Claude Shannon through the framework of information theory [CT91; Sha48].[46] By assuming the sequence S is drawn from a probabilistic distribution $p(S)$, the entropy of the distribution:[47]

$$h(S) \doteq -\int p(s)\log p(s)\mathrm{d}s \tag{1.4.3}$$

provides a natural measure of its complexity. This measure also has a natural interpretation as the average number of binary bits needed to encode such a sequence, as we will see in Chapter 3.

To illustrate the main ideas of this view, let us take a large number of long sequence segments:

$$\{S_1, S_2, \ldots, S_i, \ldots, S_N\} \subset \mathbb{R}^D, \tag{1.4.4}$$

generated by a predicting function f. Without loss of generality, we assume all sequences have the same length D, so each sequence can be viewed as a vector in $\mathbb{R}^D$. We introduce a coding scheme (with a codebook), denoted $\mathcal{E}$, that maps every segment S_i to a unique stream of binary bits $\mathcal{E}(S_i)$, and a corresponding decoding scheme $\mathcal{D}$ that maps binary bit strings back to sequence segments (say subject to error $\epsilon \geq 0$ as such an encoding scheme can be lossy). The simplest scheme fills the space spanned by all segments with ϵ-balls, as shown in Figure 1.23. We then number the balls, and each sequence is encoded as the binary index of its closest ball, while each binary index is decoded back to the center of the corresponding ball. Thus, each segment can be recovered up to precision ϵ from its bit stream.

Then, for an encoding $\mathcal{E}$, the complexity of the predicting function f can be evaluated as the average coding length $L(\mathcal{E}(S))$ of all sequences S that it generates, known as the coding rate:[48]

$$R(S \mid \mathcal{E}) = \mathbb{E}[L(\mathcal{E}(S))] \approx \frac{1}{N}\sum_{i=1}^{N} L(\mathcal{E}(S_i)). \tag{1.4.5}$$

[45] In practice, we typically care about learning the predicting function f, rather than any particular sequence generated by f.

[46] This framework has successfully guided engineering practice in the communication industry for over 80 years.

[47] Here we consider differential entropy as we assume the sequence consists of continuous variables. For discrete variables, we would use $H(S) = -\sum_i p(s_i)\log p(s_i)$.

[48] One may make this more precise by taking $R(S \mid \mathcal{E})$ to be the expected coding length for all segments S of length D.

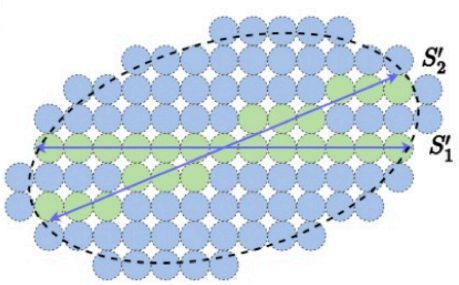


Figure 1.23: Comparison of two coding schemes. Imagine the true data distribution lies along the two arrowed lines. Samples from these lines can be encoded with a codebook of all blue balls, or with a codebook of only the green balls. The second scheme yields a much lower coding length/rate subject to the same precision.

The coding rate measure will change if we use a different coding scheme (or codebook). In practice, the better we know the low-dimensional structure around which the segments are distributed, the more efficient a codebook we can design, as illustrated in Figure 1.23. Astute readers may recognize that conceptually the denoising process illustrated in Figure 1.22 closely resembles going from the coding scheme with the blue balls to that with the green ones.

Given two coding schemes $\mathcal{E}_1$ and $\mathcal{E}_2$ for the segments, if the difference in the coding rates is positive:

$$R(S \mid \mathcal{E}_1) - R(S \mid \mathcal{E}_2) > 0, \tag{1.4.6}$$

we may say the coding scheme $\mathcal{E}_2$ is better. This difference can be viewed as a measure of how much more information $\mathcal{E}_2$ has over $\mathcal{E}_1$ about the distribution of the data. To a large extent, the goal of learning the data distribution is equivalent to finding the most efficient encoding and decoding scheme that minimizes the coding rate subject to a desired precision:

$$\min_{\mathcal{E},\mathcal{D}} R(S \mid \mathcal{E}) \quad \text{subject to} \quad \operatorname{dist}(S, \mathcal{D}(\mathcal{E}(S))) \leq \epsilon. \tag{1.4.7}$$

As we will see in Chapter 4, the achievable minimal rate is closely related to the notion of entropy $H(S)$ (1.4.3).

Remark 1.2. The perspective of measuring data complexity with explicit encoding schemes has motivated several learning objectives proposed to revise Kolmogorov complexity for better computability [WD99], including the minimum message length (MML) proposed in 1968 [WB68] and the minimum description length (MDL) in 1978 [HY01; Ris78]. These objectives normally count the coding length for the coding scheme $\mathcal{E}$ itself (including its codebook) in addition to the data S of interest: $L(\mathcal{E}(S)) + L(\mathcal{E})$. However, if the goal is to learn a finite-sized codebook and apply it to a large number of sequences, the amortized

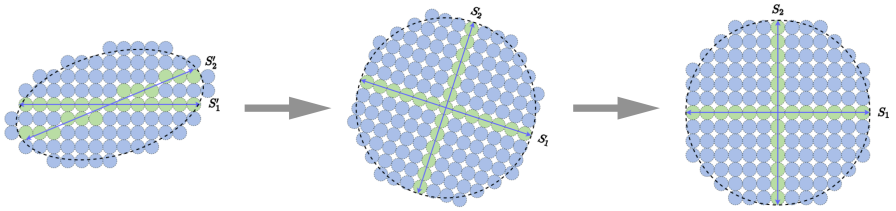


Figure 1.24: Transforming the identified low-dimensional data distribution into a more informative and structured representation.

cost of the codebook can be ignored since

$$\frac{1}{N}\left(L(\mathcal{E}) + \sum_{i=1}^{N} L(\mathcal{E}(S_i))\right) \approx \frac{1}{N}\sum_{i=1}^{N} L(\mathcal{E}(S_i)) \tag{1.4.8}$$

as N becomes large.

Again, one may view the resulting optimal coding scheme as the one that achieves the best compression of the observed data. In general, compared to the Kolmogorov complexity, the coding length given by any encoding scheme will always be larger:

$$K(S) < L(\mathcal{E}(S)). \tag{1.4.9}$$

Therefore, minimizing the coding rate/length essentially minimizes an upper bound of the otherwise uncomputable Kolmogorov complexity.

1.4.2 Learning Informative Representations

If the goal were simply to compress the given data for its own sake, the optimal codes approaching Kolmogorov complexity would in theory become nearly random or structureless [Cha66].[49] However, our true purpose in learning the data distribution of sequences S is to use it repeatedly with ease in future predictions. Hence, while compression allows us to identify the low-dimensional distribution in the data, we would like to encode this distribution in a *structured and organized* way so that the resulting representation is informative and efficient to use[50], as a "memory". Figure 1.24 illustrates intuitively why such a transformation is desirable.

As we will show in Chapter 4, these desired structures in the final representation can be precisely promoted by choosing a natural measure of *information gain* based on the coding rates of the chosen coding schemes.[51] Throughout this

[49] Any codes with structure can be further compressed.

[50] For example, to sample the distribution under different conditions.

[51] The information gain is a pervasive idea in algorithmic information theory and has been incorporated in several approaches to this topic, most recently $\mathcal{V}$-information [XZS+20]. This particular instantiation, similarly to many other measures of complexity we discussed, suffers from being increasingly poorly-conditioned as the data support becomes more compact and low-dimensional, and so is not suitable for our needs.

book, such an explicit and constructive coding approach provides a powerful computational framework for learning good representations of low-dimensional structures in real-world data. In many cases of practical importance, the coding length function can be efficiently computed or accurately approximated. In some benign cases, we can even obtain closed-form formulae—for example, subspace and Gaussian models.

Moreover, this computational framework leads to a principled approach that naturally reveals the role deep networks play in this learning process. As we will derive systematically in Chapter 5, the layers of a deep network perform operations that incrementally optimize the objective function of interest. From this perspective, the role of deep networks can be precisely interpreted as emulating a certain iterative optimization algorithm—say, gradient descent—to optimize the objective of information gain. Layers of the resulting deep architectures can be endowed with precise statistical and geometric interpretations, namely performing incremental compressive encoding and decoding operations. Consequently, the derived deep networks become transparent "white boxes" that are mathematically fully explainable.

1.4.3 Learning Consistent Representations

To summarize our discussions so far, let us denote the data as

$$\boldsymbol{X} = \{S_1, S_2, \ldots, S_i, \ldots, S_N\} \subset \mathbb{R}^D, \tag{1.4.10}$$

and let $\boldsymbol{Z} = \mathcal{E}(\boldsymbol{X})$ be the codes of $\boldsymbol{X}$ via some encoder $\mathcal{E}$:

$$\boldsymbol{X} \xrightarrow{\mathcal{E}} \boldsymbol{Z}. \tag{1.4.11}$$

In the machine learning context, $\boldsymbol{Z}$ is often called the "features" of $\boldsymbol{X}$. Note that without knowing the underlying distribution of $\boldsymbol{X}$, we do not know which encoder $\mathcal{E}$ should be used to retain the most useful information about the distribution of $\boldsymbol{X}$. In practice, people often start by trying a compact encoding scheme that serves a specific task well. In particular, they try to learn an encoder that optimizes a certain (empirical) measure of parsimony for the learned representation:

$$\min \rho(\boldsymbol{Z}). \tag{1.4.12}$$

Example 1.2. For example, image classification is such a case: we assign all images in the same class to a single code and images in different classes to different codes, say "one-hot" vectors:

$$\boldsymbol{x} \mapsto \boldsymbol{z} \in \{[1, 0, 0, \ldots, 0, 0],\ [0, 1, 0 \ldots, 0, 0],\ \ldots,\ [0, 0, 0, \ldots, 0, 1].\} \tag{1.4.13}$$

Now, a classifier $f(\cdot)$ can be modeled as a function that predicts the probability of a given $\boldsymbol{x}$ belonging to each of the k classes: $\hat{\boldsymbol{z}} = f(\boldsymbol{x}) \in \mathbb{R}^K$. Then the

"goodness" of a classifier can be measured by the so-called *cross entropy*:[52]

$$\ell(\hat{\boldsymbol{z}}, \boldsymbol{z}) = \sum_{k=1}^{K} -z_k \log \hat{z}_k, \tag{1.4.14}$$

where z_k indicates the k-th entry of the vector $\boldsymbol{z}$. As the early practice of deep networks indicated [KSH12], if enough data are given, such an encoding scheme can often be represented by a deep network and learned in an end-to-end fashion by optimizing the cross-entropy. ■

The cross-entropy loss $\ell(\hat{\boldsymbol{z}}, \boldsymbol{z})$ can be viewed as a special measure of parsimony $\rho(\boldsymbol{z})$ associated with a particular family of encoding schemes that are suitable for classification. However, such an encoding is obviously *very lossy*. The learned $\boldsymbol{z}$ does not contain any other information about $\boldsymbol{x}$ except for its class type. For example, by assigning an image with (a code representing) the class label "apple", we no longer know which specific type of apple is in the original image from the label alone.

Of course, the other extreme is to require the coding scheme to be *lossless*. That is, there is a one-to-one mapping between $\boldsymbol{x}$ and its code $\boldsymbol{z}$. However, as we will see in Chapter 4, lossless coding (or compression) is impractical unless $\boldsymbol{x}$ is discrete. For a continuous random variable, we may only consider lossy coding schemes so that the coding length for the data can be finite. That is, we only encode the data up to a certain prescribed precision. As we will elaborate more in Chapter 4, lossy coding is not merely a practical choice; it plays a fundamental role in making learning of the underlying continuous distribution possible from finite samples of the distribution.

For many learning purposes, we want the feature $\boldsymbol{z}$, although *lossy*, to retain more information about $\boldsymbol{x}$ than just its class type. In this book, we will introduce a more general measure of parsimony based on coding length/rate associated with a more general family of coding schemes—coding with a mixture of subspaces or Gaussians. This family has the capability to closely approximate arbitrary real-world distributions up to a certain precision. As we will see in Chapter 4 and Chapter 5, such a measure will not only preserve most information about the distribution of $\boldsymbol{X}$ but will also promote certain desirable geometric and statistical structures for the learned representation $\boldsymbol{Z}$.

Bidirectional auto-encoding for consistency. In a broader learning context, the main goal of a compressive coding scheme $\mathcal{E}$ is to identify the low-dimensional structures in the data $\boldsymbol{X}$ so that they can be used to predict things in the original data space. This requires that the learned encoding scheme $\mathcal{E}$ admits an efficient decoding scheme, denoted $\mathcal{D}$, which maps the feature or

[52] The cross entropy can be viewed as a distance measure between the ground truth distribution of $\boldsymbol{z}$ and that of the prediction $\hat{\boldsymbol{z}}$. It can also be viewed as the expected coding length of $\boldsymbol{z}$ if we use the optimal code book for $\hat{\boldsymbol{z}}$ to encode $\boldsymbol{z}$. The cross entropy reaches its minimum when $\boldsymbol{z}$ and $\hat{\boldsymbol{z}}$ have the same distribution.

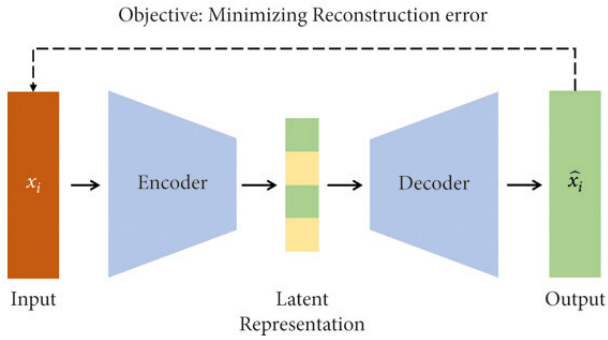


Figure 1.25: Illustration of the architecture of an autoencoder.

encoded data $\boldsymbol{Z}$ back to the data space:

$$\boldsymbol{X} \xrightarrow{\mathcal{E}} \boldsymbol{Z} \xrightarrow{\mathcal{D}} \hat{\boldsymbol{X}}. \tag{1.4.15}$$

We call such an encoding–decoding pair $(\mathcal{E}, \mathcal{D})$ an *auto-encoding*. Figure 1.25 illustrates this process.

Ideally, the decoding is approximately an "inverse" of the encoding, so that the data (distribution) $\hat{\boldsymbol{X}}$ decoded from $\boldsymbol{Z}$ is similar to the original data (distribution) $\boldsymbol{X}$ to some extent.[53] If this holds, we can recover or predict from $\boldsymbol{Z}$ what occurs in the original data space. In this case, we say the pair $(\mathcal{E}, \mathcal{D})$ provides a *consistent* auto-encoding. For most practical purposes, we not only need such a decoding to exist, but also need it to be efficiently realizable and physically implementable. For example, if $\boldsymbol{x}$ is real-valued, quantization is required for any decoding scheme to be realizable on a finite-state machine, as we will explain in Chapter 4. Thus, in general, realizable encoding and decoding schemes are necessarily lossy. A central question is how to learn a compact (lossy) representation $\boldsymbol{Z}$ that can still predict $\boldsymbol{X}$ well.

Generally speaking, both the encoder and decoder can be modeled and realized by deep networks and learned by solving an optimization problem of the following form:

$$\min_{f,g}[\ell(\boldsymbol{X}, \hat{\boldsymbol{X}}) + \rho(\boldsymbol{Z})], \qquad \text{where} \quad \boldsymbol{Z} = f(\boldsymbol{X}), \quad \hat{\boldsymbol{X}} = g(\boldsymbol{Z}) \tag{1.4.16}$$

where $\ell(\cdot, \cdot)$ is a distance function that promotes similarity between $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$[54] and $\rho(\boldsymbol{Z})$ is a measure that promotes parsimony and information richness of $\boldsymbol{Z}$. The classic principal component analysis (PCA) [Jol02] is a typical example of a consistent auto-encoding, which we will study in great detail in Chapter 2, as a precursor to more general low-dimensional structures. In Chapter 6, we will study how to learn consistent auto-encoding for general (say nonlinear) low-dimensional distributions.

[53] We will make precise what we mean by "similar" later.

[54] Either sample-wise or distribution-wise, depending on the choice of ℓ.

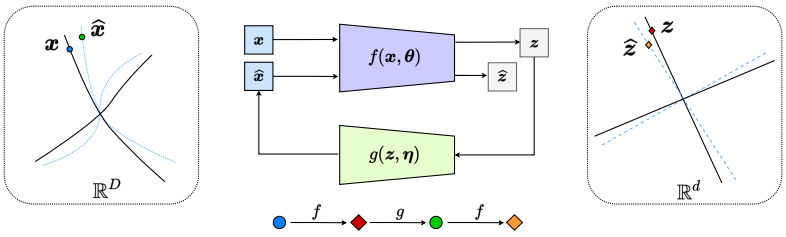


Figure 1.26: Illustration of a closed-loop transcription. Here we use a mapping f to represent the encoder $\mathcal{E}$ and g to represent the decoder $\mathcal{D}$.

1.4.4 Learning Self-Consistent Representations

In the auto-encoding objective above, one must evaluate how close the decoded data $\hat{\boldsymbol{X}}$ is to the original $\boldsymbol{X}$. This typically requires external supervision or knowledge of an appropriate similarity measure. Computing similarity between $\hat{\boldsymbol{X}}$ and $\boldsymbol{X}$ can be prohibitively expensive, or even impossible.[55] In nature, animals learn autonomously without comparing their estimate $\hat{\boldsymbol{X}}$ with the ground truth $\boldsymbol{X}$ in the data space; they typically lack that option.[56]

How, then, can a system learn without external supervision or comparison? How can it know that $\hat{\boldsymbol{X}}$ is consistent with $\boldsymbol{X}$ without direct comparison? This leads to the idea of "closing the loop." Under mild conditions (made precise in Chapter 6), ensuring consistency between $\boldsymbol{X}$ and $\hat{\boldsymbol{X}}$ reduces to encoding $\hat{\boldsymbol{X}}$ as $\hat{\boldsymbol{Z}}$ and checking consistency between $\boldsymbol{Z}$ and $\hat{\boldsymbol{Z}}$. We call this notion *self-consistency*, illustrated by:

$$\boldsymbol{X} \xrightarrow{\ \mathcal{E}\ } \boldsymbol{Z} \xrightarrow{\ \mathcal{D}\ } \hat{\boldsymbol{X}} \xrightarrow{\ \mathcal{E}\ } \hat{\boldsymbol{Z}}, \tag{1.4.17}$$

We call this process *closed-loop transcription*,[57] depicted in Figure 1.26.

It is arguably true that any autonomous intelligent being only needs to learn a self-consistent representation $\boldsymbol{Z}$ of the observed data $\boldsymbol{X}$, because checking consistency in the original data space—often meaning in the external world—is either too expensive or even physically infeasible. The closed-loop formulation allows one to learn an optimal encoding $f(\cdot,\boldsymbol{\theta})$ and decoding $g(\cdot,\boldsymbol{\eta})$ via a min-max game that depends only on the internal (learned) feature $\boldsymbol{Z}$:

$$\max_{\boldsymbol{\theta}} \min_{\boldsymbol{\eta}} [\ell(\boldsymbol{Z},\hat{\boldsymbol{Z}}) + \rho(\boldsymbol{Z})], \qquad \text{where} \quad \boldsymbol{Z} = f(\boldsymbol{X}), \quad \hat{\boldsymbol{X}} = g(\boldsymbol{Z}), \quad \hat{\boldsymbol{Z}} = f(\hat{\boldsymbol{X}}) \tag{1.4.18}$$

where $\ell(\boldsymbol{Z},\hat{\boldsymbol{Z}})$ is a loss function based on coding rates of the features $\boldsymbol{Z}$ and $\hat{\boldsymbol{Z}}$, which, as we will see, can be much easier to compute. Here again, $\rho(\boldsymbol{Z})$ is some measure that promotes parsimony and information richness of $\boldsymbol{Z}$. Somewhat

[55] For instance, minimizing a distributional distance between the two random variables is statistically and computationally difficult.

[56] In animals and humans, the eyes process visual data long before it arrives at the brain; the brain does not have a mechanism to process raw visual data and must rely on the eyes' features to do any learning.

[57] Inspired by transcription between DNA and RNA or other proteins.

surprisingly, as we will see in Chapter 6, under rather mild conditions such as $\boldsymbol{X}$ being sufficiently low-dimensional, self-consistency between $(\boldsymbol{Z}, \hat{\boldsymbol{Z}})$ implies consistency in $(\boldsymbol{X}, \hat{\boldsymbol{X}})$! In addition, we will also see that a closed-loop system will allow us to learn the distribution in a *continuous and incremental* manner,[58] without suffering from problems such as catastrophic forgetting associated with open-loop models.

1.5 Bridging Theory and Practice for Machine Intelligence

So far, we have introduced several related frameworks for learning a compact and structured representation $\boldsymbol{Z}$ for a given data distribution $\boldsymbol{X}$:

- the open-ended encoding (1.4.11);
- the bi-directional auto-encoding (1.4.15);
- the closed-loop transcription (1.4.17).

In this book, we will systematically study all three frameworks, one after another:

$$\textbf{open-ended} \implies \textbf{bi-directional} \implies \textbf{closed-loop}, \tag{1.5.1}$$

in Chapter 5, Section 6.1, and Section 6.2, respectively.

As we will see in this book, these frameworks, including many discussed earlier in the chapter, can now be viewed as attempts to address *partially* the problem of learning a *complete and computable* representation of the distribution of the data $\boldsymbol{X}$:

$$\boldsymbol{X} \longleftrightarrow \boldsymbol{Z} \longleftrightarrow \boldsymbol{G}, \tag{1.5.2}$$

which includes two sets of mappings:

1. an autoencoding between the data $\boldsymbol{X}$ and its informative feature representation $\boldsymbol{Z}$,
2. a diffusion and denoising process between the learned representation $\boldsymbol{Z}$ and a simple distribution $\boldsymbol{G}$, say Gaussian.

The later process allows us to access and sample the learned structured low-dimensional features $\boldsymbol{Z}$ for tasks such as regenerating the original data $\hat{\boldsymbol{X}}$.[59] The overall geometric picture associated with these two (transforming) processes is illustrated in Figure 1.27.

[58] That is, to learn with one class at a time or even one sample at a time.

[59] Notice that even if the learned features lie on a single low-dimensional linear subspace, the most efficient ways to identify the subspace is essentially through a denoising process: the power iteration that we will elaborate in Chapter 2.

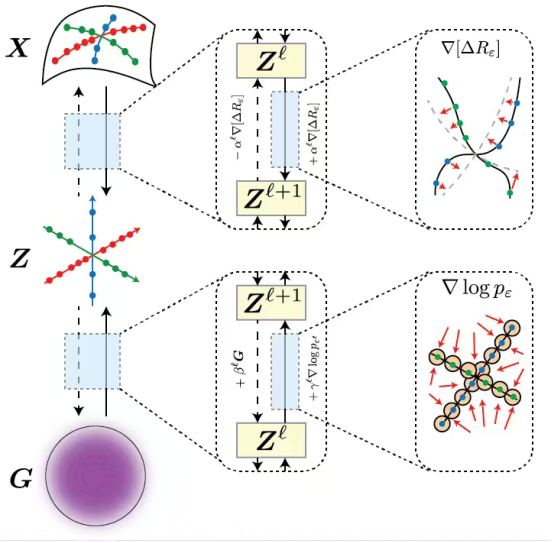


Figure 1.27: Illustration of a learned informative and structured feature representation $\boldsymbol{Z}$ for the data $\boldsymbol{X}$ that can be efficiently accessed through a denoising process from $\boldsymbol{G}$.

As we will see in Chapter 5, the role of modern deep networks can be precisely interpreted as to realize the operators[60] needed to conduct the above transformations among distributions. Popular architectures of deep neural networks (including CNNs, ResNet or Transformers) that were developed empirically will be mathematically interpreted, justified and improved through this approach.

In recent years, many theoretical frameworks have been proposed to help understand deep networks. However, most have failed to provide scalable solutions that match the performance of empirical methods on real-world data and tasks. Many theories offer little practical guidance for further empirical improvement. Chapter 7 and Chapter 8 will demonstrate how the framework presented in this book can bridge the gap between theory and practice. Chapter 7 will show how to use the learned distribution and its representation to conduct (Bayesian) inference for practical tasks involving (conditional) generation, estimation, and prediction. Chapter 8 will provide compelling experimental evidence that networks and systems designed from first principles can achieve competitive—and potentially superior—performance on tasks such as visual representation learning, image classification, image completion, segmentation, and text generation.

Back to Intelligence. As mentioned at the beginning, a fundamental task of any intelligent being is to learn predictable information from sensed data. We now understand the computational nature of this task and recognize that it is

[60] via unrolled optimization, as we will see

a never-ending process for two reasons:

- Memory or knowledge learned from data, say via encoding and decoding schemes, is unlikely to be correct or optimal. Intelligence must be able to improve when predictions of new observations contain errors.
- Observed data do not yet cover all predictable information. Intelligence must recognize when current memory or knowledge is inadequate and acquire new information whenever it becomes available.

Thus, intelligence is *not* about collecting all informative data in advance and training a model to memorize their (low-dimensional) distribution. Instead, it requires computational mechanisms that can continuously improve current knowledge of the distribution and acquire new information when needed. A fundamental characteristic of any intelligent being or system—whether an animal, a human, a civilization, or the entire scientific community—is *the ability to continuously improve or gain new information or knowledge on its own* through a closed-loop feedback about how good its current memory or knowledge is at explaining or predicting its observations. Hence, to some extent, we can symbolically express the relationship between intelligence and knowledge[61] as:

$$\begin{aligned} \text{Intelligence}(t) &= \frac{\mathrm{d}}{\mathrm{d}t}\,\text{Knowledge}(t), \\ \text{Knowledge}(t) &= \int_0^t \text{Intelligence}(s)\mathrm{d}s. \end{aligned}$$

A closed-loop feedback framework is a universal mechanism for any system capable of self-improvement and self-learning—reinforcement being a primitive form, as in natural selection—or gaming, with scientific inquiry representing its most advanced form through hypothesizing or conjecturing and falsification or verification. All intelligent beings and systems in nature employ such closed-loop mechanisms for learning at every level and scale. This ubiquity inspired early attempts to model intelligence with autonomous machines or semi-autonomous computers, particularly the Cybernetics movement pioneered and advocated by Norbert Wiener in the 1940s.

We hope this book helps readers better understand the scientific objectives, mathematical principles, and computational mechanisms underlying intelligence, at least the part for forming useful memory or empirical knowledge. We believe that this work not only bridges the theory and practice, but also connects the current to the future: It clarifies what we have done so far and at the same time reveals what we have not. Hence it lays the groundwork for future study of possibly new mechanisms behind higher-level intelligence that is unique to human—true "Artificial Intelligence" meant by the 1956 Dartmouth program. Such mechanisms can create abstract concepts and conduct deductive inference that, we believe, go beyond compression for memory from empirical data and

[61] Here we mean "knowledge" in the broadest possible sense, including both empirical memory and scientific knowledge.

conduct inductive inference that are studied and explained in this book and are being exploited in current "AI" technology. We will discuss open problems towards understanding and potentially realizing more advanced intelligence in the final Chapter 9.