CLAUDE OPUS 4.5

INFORMATION THEORY

Contents

1
What Is Information?

You are on a ship in a storm. The rain is so heavy you can barely see the bow from the wheelhouse. The captain stands at the helm, knuckles white, navigating by instruments alone. Then a light flashes through the murk—another ship, signaling with a lamp. The first mate translates the Morse: three words. "Rocks. Bear. Starboard."

The captain spins the wheel. The crew scrambles to adjust the sails. An hour later, the storm breaks, and you see what you would have hit: a reef of black stone, invisible in the rain, exactly where you would have been.

Three words saved the ship. But here is a strange question: how much *information* did those three words contain? More than the thousands of words in the ship's log that day? More than the captain's thirty years of experience? Is there any sensible way to measure "how much" information something contains?

We use the word "information" constantly. We speak of information overload, of the information age, of misinformation. We say some messages are more informative than others. But what *is* information? Is it physical—like mass or energy? Is it abstract—like a number? Can it be created or destroyed? Can you have negative information?

In this book, we will see that these questions have precise answers. Information is as measurable as length or weight. And this measurement connects the workings of steam engines to the design of smartphones, the nature of black holes to the limits of artificial intelligence.

But before we can measure, we must understand what we are measuring.

1.1 Information Lives in Surprise

Let us start with a simple observation. Suppose someone tells you, "The sun rose this morning." Have they given you information? In one sense, yes—they have communicated a fact about the world. But in another sense, no—you already knew the sun would rise. You would have been astonished if it hadn't. The message told you nothing you didn't already expect.

Now suppose someone tells you, "There will be a total solar eclipse over your city tomorrow." This feels different. You didn't expect this. You couldn't have predicted it. The message has changed your understanding of tomorrow. This seems more *informative*—but why, exactly?

The difference has to do with expectation. When something expected happens, we learn little. When something unexpected happens, we learn much. Information, it seems, is connected to surprise.

The word "surprise" here is technical, not emotional. A coin landing heads isn't emotionally surprising, but it resolves genuine uncertainty.

Consider a doctor delivering test results. Doctor A tells Patient A: "You don't have the rare disease that affects one in a hundred thousand people." Doctor B tells Patient B: "Your test for the common condition that affects half the population came back negative." Both patients received medical results. But Patient B learned more—their uncertainty was reduced more dramatically. Before the test, Patient B genuinely didn't know whether they had the condition. Patient A, by contrast, already knew their odds were negligible.

This suggests a principle: information relates to *reduction of uncertainty*. The more uncertain you were before, the more information you gained from learning the answer.

But this principle has a problem. It makes information depend on the receiver. The eclipse message is highly informative to someone who didn't know about it, but worthless to someone who already had it marked on their calendar. Is information really so subjective?

You might say: "Perhaps what we mean by informative is *useful*. The eclipse message is more useful than the sunrise message." But usefulness depends on circumstances. The eclipse is useless information to someone who doesn't care about astronomy. We want to capture something more fundamental than practical value.

You might also say: "The eclipse message is more informative because it's rarer. Rare things are more surprising." This is closer to the truth. But how do we turn "rarer" and "more surprising" into something we can measure? That is the puzzle.

1.2 *The Guessing Game*

Let us approach the problem from a different angle. Consider the game of Twenty Questions. One player thinks of something—a person, a place, an object—and the other player asks yes-or-no questions to figure out what it is. The goal is to guess correctly in as few questions as possible.

Some players are bad at this game. They ask questions like "Is it a giraffe?" or "Is it the Eiffel Tower?" These are terrible questions because they're almost certainly wrong, and when wrong, they eliminate only a single possibility.

Good players ask different questions. "Is it alive?" "Is it larger than a breadbox?" "Did it exist before 1900?" These are powerful questions because, regardless of the answer, they eliminate roughly half the possibilities. If there are a million things in the world you might be thinking of, and I ask "Is it alive?", your answer—whether yes or no—cuts my search space roughly in half.

The strategy of halving is called "binary search" in computer science. It's one of the most important algorithms ever discovered, though people have been doing it intuitively for thousands of years.

This suggests something important. A good question is one that splits the remaining possibilities as evenly as possible. And the *answer* to such a question carries a certain amount of information: it eliminates half the uncertainty.

Let us put some numbers to this. Suppose there are exactly one million equally likely possibilities. How many yes-or-no questions do you need to identify which one is correct? Each question, if asked well, cuts the possibilities in half. After one question: 500,000. After two: 250,000. After three: 125,000. And so on.

After how many questions do we reach one possibility? We need to find how many times we must divide one million by two. The answer is about twenty, because 2^{20} is approximately one million. So Twenty Questions is well-named: with twenty good questions, you can identify any one of a million possibilities.

But notice what we've discovered. The number of questions needed is not proportional to the number of possibilities—it's proportional to something like the *logarithm* of the number of possibilities. A thousand possibilities require about ten questions ($2^{10} = 1024$). A million require about twenty. A billion require about thirty. This logarithmic relationship will turn out to be fundamental.

Now here is a complication. What if the possibilities aren't equally likely? Suppose you're thinking of a person, and there's a 99 percent chance it's the person standing right in front of us. A naive strategy would still ask "Is it alive?" But a clever player would ask, "Is it the person in front of us?"

Why is this clever? Not because it eliminates half the possibilities—it doesn't. If the answer is yes (which happens 99 percent of the time), we're done immediately. If the answer is no (which happens 1 percent of the time), we've eliminated only one possibility but learned something surprising. The question is good because it resolves most of the *uncertainty*, even though it doesn't split the possibilities evenly.

The difference between eliminating possibilities and eliminating uncertainty is subtle but crucial. We'll see in Chapter 2 that entropy measures uncertainty, not number of possibilities.

This tells us that information relates not just to counting possibilities, but to weighing them by their probabilities. A message that rules out likely possibilities is more informative than one that rules out unlikely ones.

Let us extend the metaphor. Every message you receive can be thought of as an answer in a game of Twenty Questions. Someone telling you "The train is late" is like answering "yes" to the question "Is the train late?" Someone handing you a photograph is like answering many questions at once about what things look like. Every piece of communication resolves uncertainty—narrows down the space of what might be true.

And the "information content" of a message, whatever that turns out to mean precisely, should relate to how much uncertainty it resolves. A message that could have been predicted—whose answer you already knew—tells you nothing. A message that could have gone either way tells you something. A message that completely upends your expectations tells you a great deal.

1.3 Telegrams and Economy

Let us turn from games to engineering. In the nineteenth century, the telegraph transformed communication. For the first time in history, messages could travel faster than people. But this miracle came with a cost: telegraph companies charged by the word.

If you needed to send a telegram, you thought carefully about every word. Victorian verbosity gave way to telegraphic brevity. "Dear Aunt Martha, I hope this letter finds you in good health and spirits, and I write to inform you that I shall be arriving on the afternoon train this Thursday" became "ARRIVING THURSDAY AFTERNOON." Four words instead of thirty-two.

But here is the interesting question: how much information was lost? The elaborate letter and the terse telegram convey the same essential fact—when the sender will arrive. The extra words in the letter were not carrying additional information; they were *redundant*. They could be predicted from context, from convention, from the rules of polite correspondence.

This observation—that most of what we communicate is predictable—turns out to be profound. Consider English text. If I write "The quick brown fox jumps over the lazy," you can guess the next word. If I write "TH_," you can guess the missing letter. English is highly redundant: each letter, each word, carries less information than it could because so much is predictable from what came before.

You can test redundancy yourself. Take a paragraph and remove some letters. If you can still read it, those letters were redundant. "Th_s s_nt_nc_ _s st_ll r__d_bl_."

Here is a thought experiment. Take a passage of English text and show it to someone letter by letter. After each letter, ask them to guess the next one. A skilled player, familiar with English, will guess correctly much of the time. When they guess wrong, you reveal the correct letter and continue. How many letters per word does the guesser need to be *told*, on average? Experiments suggest about one or two, even though

English words average about five letters. Most letters are predictable.

This has a striking implication. If most of what we write is redundant, then the "true information content" of a message might be much smaller than its length. We could, in principle, develop codes that transmit only the unpredictable parts. The incompressible core of a message—what's left after you've squeezed out all the redundancy—might be the actual information.

Samuel Morse understood something of this intuitively. When he designed his code, he made clever choices. The letter E, the most common in English, got the shortest code: a single dot. The letter T, second most common, got a single dash. Meanwhile, rare letters like Q and Z got long codes: dash-dash-dot-dash and dash-dash-dot-dot. Morse didn't know information theory—it wouldn't exist for another century—but his instincts were sound.

How did he know to do this? He visited a printer and counted letters in the type cases. The printers kept vast quantities of E's and T's, but only a handful of Z's and Q's. Frequency dictated the supply. Morse realized that common letters should have short codes so that common messages would be short. He was compressing before compression had a name.

Morse reportedly visited a printing press and counted the letters in the type cases to determine their frequencies. The printers kept more E's than Z's because they needed them more often.

You might say: "But compression depends on the code—different codes give different lengths." This is true. But here is the remarkable thing: there is a *limit* to how much any code can compress a message. You cannot compress English text below a certain point, no matter how clever your encoding. And that limit—the length of the incompressible core—tells us something fundamental about the information content of English.

We are circling around a definition. Information seems to be what's left after you remove the predictable parts. It's the surprise, the uncertainty resolved, the questions answered. But we still can't say *how much* information in precise terms. We need one more perspective.

1.4 *The Gambler's Edge*

Imagine you're at a racetrack. There are ten horses running, and the bookmakers have posted odds. According to the public betting, Horse A is the favorite at 2:1, while Horse J is a long shot at 50:1.

Now suppose you have inside information. You know that Horse J has been secretly training at altitude, that the jockey has a particular strategy for this track, that the conditions today favor speed over stamina. You believe Horse J actually has a one-in-ten chance of winning, not one-in-fifty.

What is this knowledge worth? You can calculate it precisely. If you bet optimally over many races, your inside information translates into expected profit. The bookmaker's odds reflect what the public knows; your private knowledge is the gap between public and private, and that gap is money.

This isn't an endorsement of gambling. But the mathematics of betting has illuminated many deep questions, from probability theory to financial economics.

This gives us another lens on information. Information is what gives you an edge. Perfect information means you can never be surprised; you always know what will happen. No information means you're always guessing; you're at the mercy of chance. The more information you have, the better you can predict, the more you can profit.

There is even a precise theory of this, called the Kelly criterion. It tells you exactly how much to bet when you know more than the market. And the optimal growth rate of your wealth depends on a quantity that, as we shall see, is closely related to information.

Consider a simpler scenario. You have access to an oracle who will truthfully answer one yes-or-no question about tomorrow's stock market. What question do you ask? "Will stocks go up tomorrow?" might be good—but only if it's currently uncertain. If everyone already knows stocks will rise, the question is worthless. The value of your question depends on your current uncertainty.

Or consider a different oracle: one who answers questions about a coin that's about to be flipped. If the coin is fair, your question "Will it land heads?" is maximally valuable—you're completely uncertain, and the answer completely resolves that uncertainty. But if the coin is biased 99-to-1 toward heads, your question is nearly worthless. You already knew it would probably land heads; the oracle confirms what you expected.

There's a deep connection here to physics. In thermodynamics, "free energy" measures how much work you can extract from a system. Information turns out to play a similar role: it measures how much you can *do* with your knowledge.

We keep finding the same pattern. Information measures the gap between what you knew and what you learned. The more uncertain you were, the more information you gained. The more predictable the outcome, the less information it contains.

But "gap between what you knew and what you learned" is still poetry, not mathematics. Can we do better?

1.5 What Information Is Not

Before we go further, let us clear away some common confusions. The word "information" in everyday English carries many meanings, and not all of them are what we want to measure.

Information is not meaning. "The cat sat on the mat" and "Le chat s'est assis sur le tapis" express the same meaning, but they require different amounts of space to transmit. The English sentence has 22 letters; the French has 30. In our sense, the French sentence contains more "information"—it's longer and requires more yes-or-no questions to specify—even though it means the same thing. We are measuring *representation*, not *interpretation*.

Information is not usefulness. A random string of a thousand digits—say, 7293847291038475120...—contains a lot of information in our sense. It's completely unpredictable; you can't compress it; every digit is a surprise. But it might be completely useless. Meanwhile, a single word at the right moment—"Fire!"—contains very little information (it's short, it could almost have been predicted from the smoke) but might save your life. We are measuring something technical, not something practical.

Information is not knowledge. You can receive information without understanding it. If I send you an encrypted message, you've received just as much information as if the message were in plain text—the same number of bits arrived at your computer. But you might have no idea what it means. Information theory concerns what is transmitted, not what is comprehended.

Information is not truth. "The moon is made of cheese" contains information just like "The moon is made of rock." Both are sentences of similar length, both resolve uncertainty about what the speaker believes, both require similar resources to transmit. A false message carries information just as a true one does.

You might say: "If information isn't meaning, or usefulness, or knowledge, or truth, then what good is measuring it?" The answer: because transmission, storage, and processing don't care about meaning. A telephone cable doesn't know whether the words passing through it are profound or trivial. A hard drive doesn't know whether the files on it are true or false. The *physical* constraints on communication depend only on the *amount* of information, not its content.

This is liberating. It means we can study communication in general, without getting tangled in the specifics of particular messages.

And understanding those physical constraints—how many bits can flow through a wire, how much can be stored on a disk, how reliably messages can survive noise—turns out to illuminate much more than engineering. We will see that the same concepts apply to thermodynamics, to statistics, to the fundamental limits of computation.

There is something philosophically strange here. We are defining a technical concept called "information" that doesn't quite match the everyday word. But this is normal in science. The physicist's "energy" doesn't mean enthusiasm. The physicist's "work" doesn't mean employment. Technical terms earn their meaning from their usefulness in building a coherent framework.

1.6 The Unity Emerges

Let us gather what we've found. We have approached information from several directions:

Surprise. Information relates to the unexpected. When something expected happens, we learn little; when something unexpected happens, we learn much.

Questions. Information relates to yes-or-no questions answered. A message that identifies one possibility out of a million is worth about twenty questions; one possibility out of two is worth one question.

Compression. Information relates to the incompressible core of a message. Redundancy can be squeezed out; what remains is the true content.

Advantage. Information relates to betting edges. The more you know beyond what others know, the more you can profit.

All of these point toward the same underlying concept: *uncertainty reduced*. Information is what closes the gap between ignorance and knowledge. The more uncertain you were before receiving a message, the more information it contained.

But "uncertainty reduced" is still vague. We need to make it mathematical. How do we measure uncertainty? If I learn two things, is my total information the sum of what each taught me? Can uncertainties be added and subtracted like ordinary numbers?

We need a *function*—something that takes a probability distribution as input and outputs a number measuring uncertainty. And this function should satisfy certain sensible requirements:

First, more outcomes should mean more uncertainty. If a coin could land heads or tails, there's less uncertainty than if a die could show any of six faces.

Second, certain outcomes should contribute nothing. If I already know something will happen (probability 100%), learning that it happened tells me nothing.

Third, independent information should add. If I flip a coin and roll a die, and these are unrelated, my total uncertainty should be the sum of my uncertainty about the coin plus my uncertainty about the die.

Is there a function satisfying all these requirements? There is. It's called *entropy*, and it will be the foundation of everything that follows.

These requirements might seem obvious, but making them precise turns out to determine the answer almost uniquely. This is one of the remarkable features of information theory.

1.7 The Name That Connects Two Worlds

There is a story—perhaps apocryphal, but illuminating—about how entropy got its name in information theory.

In the 1940s, Claude Shannon was developing his mathematical theory of communication at Bell Telephone Laboratories. He had a quantity that measured uncertainty, and he wasn't sure what to call it. He consulted John von Neumann, one of the great mathematicians of the twentieth century.

Von Neumann's reply: "You should call it entropy, for two reasons. First, your uncertainty function has been used in statistical mechanics under that name, so it already has a name. Second, and more importantly, nobody really knows what entropy is, so in a debate you will always have the advantage."

Von Neumann was being somewhat ironic. Physicists had been arguing about what entropy "really meant" since Clausius coined the term in 1865.

Whether or not this conversation happened exactly as described, the name stuck. Shannon's entropy. But the choice was not arbitrary. There is a deep connection between Shannon's concept and the entropy that physicists had been studying for almost a century.

The story begins in the 1860s with Rudolf Clausius, who was trying to understand heat engines. He noticed that certain processes are irreversible—heat flows from hot to cold, never the reverse—and he introduced a quantity to track this directionality. He called it *entropy*, from the Greek word for "transformation."

A few decades later, Ludwig Boltzmann gave entropy a statistical interpretation. He showed that the entropy of a gas was related to counting: specifically, to the number of microscopic arrangements of molecules consistent with what you could observe macroscopically. A gas has high entropy when there are many such arrangements, low entropy when there are few. Entropy measured something like the "hiddenness" of the microscopic world—how much you *don't* know about the exact state even when you know the bulk properties.

Boltzmann's formula, $S = k \log W$, is engraved on his tombstone. Here W is the number of microscopic arrangements and k is Boltzmann's constant.

Is this the same as Shannon's entropy? On the surface, they seem completely different. Boltzmann was talking about molecules bouncing around in a box; Shannon was talking about messages traveling down a wire. One is physics; the other is communication.

But the mathematical form is almost identical. Both involve logarithms. Both increase when there are more equally likely possibilities. Both have something to do with "not knowing"—in Boltzmann's case, not knowing the microscopic details; in Shannon's case, not knowing what message will arrive.

This is not a coincidence. As we will see in later chapters, the connection is profound and physical. Information theory and thermodynamics are two perspectives on the same underlying reality. The entropy of a physical system *is* information—specifically, the information you're missing about the system's exact state. And this connection has practical consequences, from the efficiency of heat engines to the fundamental limits on computing.

For now, it is enough to note that the name "entropy" carries weight. It suggests that information is not merely an abstract concept, but something connected to the physical world, to energy, to the fundamental laws governing the universe.

1.8 The Road Ahead

We have circled around a definition without quite stating it. We have seen that information relates to surprise, to questions answered, to compression limits, to betting advantages. We have seen that these different perspectives converge on a single concept: uncertainty reduced. And we have seen that this concept has a name—entropy—that connects it to physics.

What we have not done is make any of this precise. We have no formula for entropy. We cannot calculate, in any given situation, "how much" information is present. We have intuitions but not tools.

That will change in the next chapter. We will derive the entropy formula—not as something handed down from above, but as the *unique* answer to the requirements we've articulated. We will see that this formula is more than a definition: it's a key that unlocks doors across science.

With entropy in hand, we will ask: How much can messages be compressed? Can we communicate reliably over noisy channels? What is the connection to thermodynamics? What limits does physics place on computation? What does information theory tell us about learning from data?

These questions have answers, precise and surprising. The mathematics is not difficult—mostly logarithms and probability—but the ideas are deep. We will see that information theory connects steam engines to smartphones, that entropy appears wherever uncertainty is found, that the limits on communication are also limits on physics.

The answer to "Can we communicate reliably over noisy channels?" is one of the great surprises of information theory. The answer is yes—if we're clever enough.

Let us return to where we began: a ship in a storm, a signal lamp flashing through the rain, three words that saved the ship. We can now say something about those words. They were informative precisely because they were unexpected. If the captain had already known about the rocks, the message would have been redundant—it would have told him nothing he didn't already know. The message carried information because it reduced uncertainty, because it answered a question the captain urgently needed answered.

Information is the gap between what you knew and what you learned. Nothing more, nothing less.

Let us see how to make this precise.

2
Shannon Entropy

We ended the last chapter with our hands full of intuitions and no way to measure them. We know that learning the result of a fair coin flip feels more informative than learning the sun rose this morning. We know that twenty questions about a word in the dictionary is harder than twenty questions about a number from one to ten. But "feels more informative" and "is harder" are not numbers. Physics became physics when Galileo started measuring. Chemistry became chemistry when Lavoisier started weighing. What number should we assign to the information in a message?

The question sounds almost philosophical—one of those questions cocktail-party philosophers might debate while the actual work of engineering gets done by people who don't worry about such things. But here is the remarkable fact: the question has an answer. Not an answer decreed by definition, but an answer *forced* on us by simple requirements. If we demand that our measure of information behave sensibly—that independent events combine additively, that certainty contains zero information, that our measure be continuous—then mathematics backs us into a corner. There is exactly one formula that works.

This is unusual in mathematics. Most definitions are choices. But the entropy formula is forced by logic, much like the Pythagorean theorem is forced by the axioms of geometry.

It was discovered by Claude Shannon in 1948, and it is the foundation of everything that follows.

2.1 *What Should "Information" Even Mean?*

Before we can measure information, we must decide what properties our measure should have. We do not start with a definition; we start with requirements. Whatever our measure $H(X)$ turns out to be, it should satisfy certain properties that capture the intuitions we developed in Chapter 1.

Let us build these requirements one by one.

Requirement 1: Continuity. If we change the probabilities slightly, the information should change slightly. A measure that jumps discontinuously would be useless—small errors in estimating probabilities would lead to wild swings in our measure of uncertainty. We want smooth behavior.

Requirement 2: Maximum at uniformity. Return to the Twenty Questions game. If all outcomes are equally likely, we are maximally uncertain—no outcome is more "expected" than another. So $H(X)$ should be largest when all probabilities are equal.

Think of it this way. If you're guessing which card I drew from a deck, and all 52 cards are equally likely, you're maximally uncertain. But if I tell you it's probably the ace of spades—say, 99% likely—your uncertainty plummets. You already "know" the answer, in a sense. The measure should reflect this.

Requirement 3: Additivity for independent events. This is the crucial requirement. If I flip a coin and roll a die, and these events are independent—knowing the coin result tells you nothing about the die—then the total uncertainty should be the sum of the individual uncertainties:

$$H(\text{coin and die}) = H(\text{coin}) + H(\text{die}).$$

You might object: why addition? Why not multiplication, or some other rule for combining uncertainties? The answer is that we want information to scale sensibly. If I send you two independent messages, the total information should be twice what one message contains, not squared. Addition is the natural operation for quantities that should scale with repetition.

Why addition and not multiplication? Because we want information to scale with the "size" of the system. Two independent messages should carry twice the information, not the square of it.

Requirement 4: The grouping property. Here is the subtlest requirement. Suppose we have three outcomes A, B, C with probabilities p, q, r (where $p + q + r = 1$). We can think about our uncertainty in two equivalent ways.

First way: we're uncertain among three outcomes directly.

Second way: we first ask "Is it A or not-A?" This is a binary question with probabilities p and $(1 - p)$. If the answer is "not-A," we then ask "Is it B or C?" among the remaining outcomes, with conditional probabilities $q/(1 - p)$ and $r/(1 - p)$.

The total uncertainty should be the same either way. In the second approach, we pay the uncertainty of the first question, plus—with probability $(1 - p)$—the uncertainty of the second question. This gives us:

$$H(p, q, r) = H(p, 1 - p) + (1 - p) \cdot H\left(\frac{q}{1 - p}, \frac{r}{1 - p}\right).$$

This is called the *grouping axiom* or *recursivity property*. It captures the idea that uncertainty can be decomposed hierarchically without changing the total.

The grouping property says that it doesn't matter how we decompose a decision tree—the total uncertainty comes out the same.

These four requirements—continuity, maximality at uniformity, additivity for independence, and grouping—seem modest. They merely formalize intuitions we already have. But watch what happens when we insist on all four simultaneously.

2.2 *Backed Into a Corner*

We now derive the unique formula satisfying our requirements. This is not a proof to be verified but a discovery to be made. Let us work through it together.

Step 1: The uniform case.

Define $A(n)$ to be the uncertainty of a uniform distribution over n equally likely outcomes:

$$A(n) = H\left(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right).$$

By requirement 2, this should increase with n—more outcomes means more uncertainty. Now consider a uniform distribution over nm outcomes. We can think of this as first choosing uniformly among n groups, then choosing uniformly among m items within the chosen group. These choices are independent, so by requirement 3:

$$A(nm) = A(n) + A(m).$$

This is a functional equation. What continuous functions satisfy $f(nm) = f(n) + f(m)$?

Let us check that the logarithm works. Indeed, $\log(nm) = \log(n) + \log(m)$. Does any other function work?

The logarithm is the *only* continuous function satisfying $f(xy) = f(x) + f(y)$. This is a theorem in analysis, not obvious, but we will take it as given.

Suppose f is continuous and satisfies $f(nm) = f(n) + f(m)$ for all positive integers. Setting $n = m$ gives $f(n^2) = 2f(n)$. Setting $m = n^2$ gives $f(n^3) = 3f(n)$. In general, $f(n^k) = kf(n)$. Taking $n = 2$, we have $f(2^k) = kf(2)$.

Now, for any integer n, we can write n approximately as $2^{\log_2 n}$. Continuity forces $f(n) = f(2) \cdot \log_2(n)$. So f must be a constant times a logarithm.

Therefore:

$$A(n) = K \log(n)$$

for some constant $K > 0$. The constant K just sets the units; we'll choose it shortly.

Step 2: Binary distributions.

Now consider a binary distribution $(p, 1 - p)$ where p is some probability. We need to find $H(p, 1 - p)$.

The trick is to approximate p by a rational number. Suppose $p = m/n$ for integers m and n. Consider a uniform distribution over n outcomes, grouped into two sets: the first m outcomes (probability

$m/n = p$) and the remaining $n - m$ outcomes (probability $(n-m)/n = 1 - p$).

By the grouping property:

$$A(n) = H(p, 1-p) + p \cdot A(m) + (1-p) \cdot A(n-m).$$

Wait—let me explain this carefully. The left side is the uncertainty of a uniform distribution over n outcomes. The right side decomposes this as: first, the uncertainty of which group (probability p for the first group, $1 - p$ for the second); then, given the group, the uncertainty within that group (which is $A(m)$ if we're in the first group, $A(n-m)$ if we're in the second).

We're using grouping "backwards"—decomposing uniform uncertainty into binary choice plus conditional uncertainty.

Solving for $H(p, 1-p)$:

$$\begin{aligned} H(p, 1-p) &= A(n) - p \cdot A(m) - (1-p) \cdot A(n-m) \\ &= K \log(n) - \frac{m}{n} K \log(m) - \frac{n-m}{n} K \log(n-m) \\ &= K \left[\log(n) - \frac{m}{n} \log(m) - \frac{n-m}{n} \log(n-m) \right] \\ &= K \left[-\frac{m}{n} \log \frac{m}{n} - \frac{n-m}{n} \log \frac{n-m}{n} \right] \\ &= -K \left[p \log(p) + (1-p) \log(1-p) \right]. \end{aligned}$$

In the fourth line, we used $\log(n) = \frac{m}{n} \log(n) + \frac{n-m}{n} \log(n)$ and combined terms.

By continuity (requirement 1), this formula extends from rational p to all p in $[0, 1]$.

Step 3: The general case.

For any distribution $(p_1, p_2, \ldots, p_n)$, we can apply the grouping property repeatedly. Peel off outcomes one at a time:

$$H(p_1, \ldots, p_n) = H(p_1, 1-p_1) + (1-p_1) \cdot H\left(\frac{p_2}{1-p_1}, \ldots, \frac{p_n}{1-p_1} \right).$$

Continuing this recursion and collecting terms (the algebra is tedious but straightforward), we arrive at:

$$H(p_1, \ldots, p_n) = -K \sum_{i=1}^{n} p_i \log(p_i).$$

And there it is. We demanded sensible behavior, and mathematics handed us this formula. We didn't *define* entropy to be $-\sum p \log p$; we *discovered* that it must be.

2.3 *Choosing Units*

The constant K is arbitrary—it just sets the units. With $K = 1$ and logarithm base 2, we measure entropy in *bits*. With $K = 1$ and natural logarithm, we measure in *nats*. With $K = 1/\ln 2$ and natural logarithm, we also get bits.

Shannon chose bits, and so shall we. The definition is:

$$\boxed{H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)}$$

This is *Shannon entropy*, measured in bits. It deserves a moment of appreciation. This single formula will drive everything in this book—compression, communication, thermodynamics, learning. It connects the flip of a coin to the heat death of the universe. It is a jewel.

The "bit" was named for binary digit, but Shannon showed it has this deeper meaning: the uncertainty resolved by one fair coin flip.

2.4 *What Entropy Means*

Let us unpack this formula from several angles.

Interpretation 1: Average surprise.

Consider the quantity $-\log_2(p)$ for an event with probability p. When p is small (unlikely event), $-\log_2(p)$ is large. When p is close to 1 (near-certain event), $-\log_2(p)$ is small. This quantity measures the "surprise" or "self-information" of the event.

For example, if $p = 1/2$, the surprise is $-\log_2(1/2) = 1$ bit. If $p = 1/8$, the surprise is $-\log_2(1/8) = 3$ bits. If $p = 1$, the surprise is $-\log_2(1) = 0$ bits—no surprise at all.

Self-information $-\log_2(p)$ is also called "surprisal." It measures how surprised you are when an event of probability p occurs.

Entropy is the *expected* surprise:

$$H(X) = \sum_i p_i \cdot (-\log_2 p_i) = \mathbb{E}[-\log_2 P(X)].$$

This connects directly to Chapter 1: rare events are more surprising, and entropy averages this over all possibilities weighted by their probabilities.

Interpretation 2: Number of yes-or-no questions.

For a uniform distribution over n outcomes, $H(X) = \log_2(n)$. This is exactly the number of yes-or-no questions needed in the optimal strategy for Twenty Questions—each question halves the possibilities.

For non-uniform distributions, entropy gives the *average* number of questions needed, if we ask questions cleverly. We'll make this precise in Chapter 4.

Interpretation 3: Minimum average code length.

Entropy $H(X)$ is the minimum number of bits per symbol needed to encode messages from source X, on average. We'll prove this in Chapter 4, but it's worth stating now: entropy has an operational meaning. It's not just an abstraction—it answers a concrete engineering question.

A convention. What is $0 \cdot \log_2(0)$? The expression $\log_2(0) = -\infty$, but $0 \cdot (-\infty)$ is indeterminate. However, $\lim_{p \to 0^+} p \log_2(p) = 0$, so we define $0 \log_2(0) = 0$. This makes entropy continuous at the boundary where some probabilities vanish.

2.5 The Filing Cabinet

Let us develop a metaphor that will serve us throughout this chapter.

Imagine a vast filing cabinet with n drawers. You need to find a particular document, but you don't know which drawer it's in. Your uncertainty depends on your prior knowledge about where the document might be.

If you have no idea—all drawers equally likely—your uncertainty is maximal: $H(X) = \log_2(n)$. With binary search, you'd need to check about $\log_2(n)$ drawers.

Binary search: check the middle drawer; if too high, search the lower half; if too low, search the upper half. Each step halves the remaining possibilities.

If someone tells you "it's probably in drawer 1"—say, 90% likely—your uncertainty drops dramatically. You check drawer 1 first and are usually right.

If some drawers are more likely than others, your uncertainty lies between 0 (complete knowledge) and $\log_2(n)$ (complete ignorance).

We will return to this filing cabinet. Conditioning is like someone telling you "it's in the left half"—this reduces your uncertainty. Joint entropy is like having two filing cabinets, one for each of two variables. The chain rule says: total uncertainty about which drawer and which folder equals uncertainty about the drawer plus (given the drawer) uncertainty about the folder.

2.6 Worked Examples with Actual Numbers

Let us calculate. Theory without calculation is hollow.

The Fair Coin

Let X be the result of a fair coin flip: $P(\text{heads}) = P(\text{tails}) = 1/2$.

$$\begin{aligned} H(X) &= -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} \\ &= -\frac{1}{2}(-1) - \frac{1}{2}(-1) \\ &= \frac{1}{2} + \frac{1}{2} \\ &= 1 \text{ bit.} \end{aligned}$$

One bit. This is where the name comes from: a fair binary choice carries exactly one bit of information. The "bit" was named for binary digit, but Shannon showed it has this deeper meaning—the uncertainty in a fair coin flip.

The Biased Coin

Now suppose $P(\text{heads}) = 0.9$ and $P(\text{tails}) = 0.1$.

Note that $\log_2(0.9) \approx -0.152$ and $\log_2(0.1) \approx -3.322$. You can verify these with a calculator: $2^{-0.152} \approx 0.9$ and $2^{-3.322} \approx 0.1$.

$$\begin{aligned} H(X) &= -(0.9)\log_2(0.9) - (0.1)\log_2(0.1) \\ &= -(0.9)(-0.152) - (0.1)(-3.322) \\ &= 0.137 + 0.332 \\ &= 0.469 \text{ bits.} \end{aligned}$$

Less than half a bit! The coin is predictable, so learning its outcome tells us less. If you bet on heads every time, you'd be right 90% of the time. There's not much surprise left.

The Binary Entropy Function

The entropy of a coin with bias p is:

$$h(p) = -p\log_2(p) - (1-p)\log_2(1-p).$$

This is called the *binary entropy function*. It's worth seeing its shape.

At $p = 0$ and $p = 1$, entropy is zero—the coin is deterministic. At $p = 0.5$, entropy reaches its maximum of 1 bit—maximum uncertainty. The curve is symmetric around $p = 0.5$, which makes sense: a coin that lands heads 90% of the time has the same entropy as one that lands heads 10% of the time. Both are equally predictable, just with opposite biases.

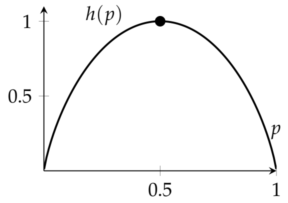


Figure 2.1: The binary entropy function $h(p)$ peaks at $p = 0.5$ with value 1 bit, and drops to zero at $p = 0$ and $p = 1$.

The Fair Die

Let X be the result of rolling a fair six-sided die. Each face has probability $1/6$.

$$\begin{aligned} H(X) &= -6 \times \frac{1}{6}\log_2\frac{1}{6} \\ &= -\log_2\frac{1}{6} \\ &= \log_2(6) \\ &\approx 2.585 \text{ bits.} \end{aligned}$$

More than two bits, less than three. Sensible: a die has more outcomes than two coin flips (4 outcomes, $\log_2 4 = 2$ bits) but fewer than three (8 outcomes, $\log_2 8 = 3$ bits).

The entropy $\log_2(6) \approx 2.585$ means that in principle, we could encode die rolls using about 2.585 bits per roll, on average. We can't quite achieve this with a single roll, but with many rolls, we can get arbitrarily close.

English Text

Here is where things get interesting. Let X be a letter drawn from English text.

First approximation—uniform. If all 27 symbols (26 letters plus space) were equally likely:

$$H(X) = \log_2(27) \approx 4.76 \text{ bits per letter.}$$

Second approximation—actual letter frequencies. But English letters are far from uniform. The letter E appears about 12.7% of the time, T about 9.1%, while Z appears only 0.07% of the time.

Summing over all 27 symbols:

$$H_1 \approx 4.11 \text{ bits per letter.}$$

Already lower than the uniform case—English has structure.

Letter	Freq	$-p \log_2 p$
E	0.127	0.380
T	0.091	0.314
A	0.082	0.294
O	0.075	0.278
I	0.070	0.266
N	0.067	0.260
⋮	⋮	⋮
Z	0.0007	0.007

Table 2.1: Contribution to entropy from selected letters.

Third approximation—digram frequencies. But letters in English are not independent. After Q, U is nearly certain. After T, H is common. Shannon estimated that accounting for letter pairs:

$$H_2 \approx 3.56 \text{ bits per letter.}$$

Fourth approximation—human experiments. Shannon conducted a clever experiment. He showed people English text one letter at a time and asked them to guess the next letter. When they guessed wrong, he revealed the correct letter and continued. By measuring how often people guessed correctly, he estimated the true entropy of English.

His conclusion:

$$H_\infty \approx 1.0 \text{ to } 1.5 \text{ bits per letter.}$$

English is highly redundant. If letters were independent and uniform, we'd need 4.76 bits per letter. But the actual information rate is closer to 1 bit per letter. English is about 75–80% redundant!

Shannon's experiment essentially measured human ability to predict English text. Good predictions mean low entropy; poor predictions mean high entropy.

This explains why you can read "cn y rd ths sntnc wtht vwls?" The missing vowels were never really carrying much information—they were predictable from context.

2.7 *The Algebra of Uncertainty*

Entropy satisfies several important properties. Let us develop them, returning to our filing cabinet metaphor where helpful.

Property 1: Non-negativity.

$H(X) \geq 0$, with equality if and only if X is deterministic (some outcome has probability 1).

You cannot have negative uncertainty. Either you know something, or you don't—but "negative not-knowing" doesn't make sense.

Property 2: Maximum at uniformity.

For a random variable over n outcomes:

$$H(X) \leq \log_2(n),$$

with equality if and only if X is uniformly distributed.

Maximum uncertainty is the uniform distribution. You cannot be *more* uncertain than knowing nothing about which outcome is favored.

This can be proved with Lagrange multipliers: maximize $-\sum p_i \log p_i$ subject to $\sum p_i = 1$.

Property 3: Joint entropy.

For two random variables X and Y, the *joint entropy* is:

$$H(X, Y) = -\sum_{x,y} P(x, y) \log_2 P(x, y).$$

This measures our total uncertainty about both X and Y together. In the filing cabinet metaphor, it's like having two filing cabinets and not knowing which drawer in either contains your document.

Property 4: Conditional entropy.

The *conditional entropy* $H(X \mid Y)$ is the uncertainty remaining about X after learning Y:

$$H(X \mid Y) = \sum_{y} P(y) H(X|Y = y) = -\sum_{x,y} P(x, y) \log_2 P(x|y).$$

In the filing cabinet: if someone tells you which section the document is in (that's learning Y), your remaining uncertainty about the specific drawer (that's X) is the conditional entropy.

Property 5: Chain rule.

This is fundamental:

$$H(X, Y) = H(X) + H(Y \mid X) = H(Y) + H(X \mid Y).$$

The uncertainty about X and Y together equals the uncertainty about X plus the remaining uncertainty about Y given X. Or equivalently, the uncertainty about Y plus the remaining uncertainty about X given Y.

The chain rule says uncertainty decomposes: total uncertainty = initial uncertainty + remaining uncertainty.

In the filing cabinet: total uncertainty about drawer and folder = uncertainty about drawer + (given drawer) uncertainty about folder.

This generalizes:

$$H(X_1, \ldots, X_n) = \sum_{i=1}^{n} H(X_i \mid X_1, \ldots, X_{i-1}).$$

Peel off uncertainty one variable at a time.

Property 6: Conditioning reduces entropy.

$$H(X \mid Y) \leq H(X),$$

with equality if and only if X and Y are independent.

"Information never hurts"—or more precisely, extra information can never *increase* your uncertainty on average.

Learning Y can only reduce (or maintain) your uncertainty about X. Information never hurts. This is intuitive: more knowledge should not make you *more* uncertain.

Property 7: Subadditivity.

$$H(X,Y) \leq H(X) + H(Y),$$

with equality if and only if X and Y are independent.

Joint entropy is at most the sum of individual entropies. If X and Y are related, knowing one tells you something about the other, so the joint uncertainty is less than if they were independent.

2.8 A Mathematical Theory of Communication

Let us pause for history. Claude Shannon was 32 years old in 1948. He worked at Bell Telephone Laboratories, where the practical problem was telephone communication. But Shannon was interested in something more fundamental: what are the ultimate limits of communication?

Bell Labs in the 1940s was perhaps the most remarkable research institution ever assembled. Shannon's colleagues included John Bardeen, Walter Brattain, and William Shockley, who would invent the transistor.

The paper appeared in two parts in the Bell System Technical Journal, July and October 1948. It was 55 pages long and invented a field. The title was modest: "A Mathematical Theory of Communication." The content was revolutionary.

What Shannon did:

1. Defined information mathematically (entropy)
2. Proved you can compress messages to their entropy (source coding theorem)
3. Proved you can communicate reliably over noisy channels up to a certain rate (noisy channel coding theorem)

It is hard to overstate how remarkable this is. Shannon didn't just solve a problem—he defined a field, proved its fundamental theorems, and provided the conceptual vocabulary that everyone still uses.

The reception was immediate and enthusiastic, but the implications took decades to absorb. Engineers initially didn't believe the noisy channel coding theorem. How could you communicate reliably over a noisy channel without error? Shannon proved it was possible but didn't say *how*. Constructing codes that actually achieved Shannon's limits occupied engineers for the next fifty years. We'll take up that story in Chapters 7 and 8.

Shannon himself was playful. He built maze-solving robots, juggling machines, and a calculator that operated in Roman numerals. His approach to research was guided by curiosity rather than applications. He once said that he worked on problems because they were interesting, not because they were useful—and the most interesting problems often turned out to be the most useful.

The 1948 paper has the quality of inevitable mathematics, as if Shannon had merely uncovered what was always there. But it emerged from years of tinkering and wondering, from thinking hard about what we mean by communication, from asking questions that others hadn't thought to ask.

2.9 *The Subjectivity of Uncertainty*

Let us turn philosophical for a moment. Entropy depends on probabilities. But where do probabilities come from?

This question—where do probabilities come from?—has occupied philosophers and statisticians for centuries. The Bayesian view: probabilities represent degrees of belief. The frequentist view: probabilities are limiting frequencies.

Consider this puzzle. I flip a coin and cover it. To me, the entropy is 1 bit—I don't know whether it landed heads or tails. But you peek under the cover and see it's heads. To you, the entropy is 0 bits—you know the answer.

The *same physical coin*, at the *same moment*, has different entropies depending on who's measuring. Entropy is not a property of the coin. It is a property of your *state of knowledge* about the coin.

You might say: "That's troubling. Shouldn't information be objective?" But this subjectivity is a feature, not a bug. Information theory is about *communication*—about one person learning what another person knows. The receiver's uncertainty is exactly what matters. A message is informative precisely when it tells the receiver something they didn't already know.

Shannon was explicit about this. His famous opening sentence: "The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point." Note the two points—sender and receiver. Information flows from one to the other. The receiver's prior uncertainty determines how informative the message is.

There is another issue. Shannon entropy measures uncertainty but ignores meaning. The message "The enemy will attack at dawn" and a random string of the same length have the same entropy. Shannon deliberately separated the engineering problem (transmitting bits) from the semantic problem (what bits mean).

"The semantic aspects of communication are irrelevant to the engineering problem." This was a strategic choice, not a philosophical claim. By ignoring meaning, Shannon could focus on what telephone lines actually need to transmit: bits, regardless of what those bits represent.

2.10 *What Happens When You Flip a Coin Many Times*

We now turn to one of the most important results in information theory: the *asymptotic equipartition property*, or AEP. It gives entropy an operational meaning beyond the abstract.

Consider flipping a biased coin n times, where $P(\text{heads}) = p$. There are 2^n possible sequences of outcomes. But they're not all equally likely.

With a fair coin ($p = 0.5$), all 2^n sequences are equally likely. But with a biased coin, sequences with "too many" or "too few" heads become rare.

The typical sequences. Most sequences will have approximately np heads and $n(1-p)$ tails. Define the *typical set* as sequences whose empirical frequency of heads is close to p—within some tolerance ϵ, say.

For a typical sequence $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ with about np heads and $n(1-p)$ tails:

$$\begin{aligned} P(\mathbf{x}) &= p^{(\text{number of heads})} \cdot (1-p)^{(\text{number of tails})} \\ &\approx p^{np} \cdot (1-p)^{n(1-p)}. \end{aligned}$$

Taking the logarithm:

$$\begin{aligned} \log_2 P(\mathbf{x}) &\approx np \log_2(p) + n(1-p)\log_2(1-p) \\ &= -n \cdot h(p) \\ &= -n \cdot H(X), \end{aligned}$$

where X is a single coin flip.

Therefore:

$$P(\mathbf{x}) \approx 2^{-nH(X)}.$$

This is the heart of the AEP: typical sequences all have approximately the same probability, namely 2^{-nH}.

The Asymptotic Equipartition Property. For large n:

1. There are approximately $2^{nH(X)}$ typical sequences.
2. Each typical sequence has probability approximately $2^{-nH(X)}$.
3. The total probability of typical sequences approaches 1.
4. Atypical sequences (those with the "wrong" proportion of heads) become vanishingly rare.

Let us put numbers to this. Suppose $n = 100$ flips of a biased coin with $p = 0.9$ (lands heads 90% of the time).

Total possible sequences: $2^{100} \approx 1.27 \times 10^{30}$.

Entropy per flip: $h(0.9) = 0.469$ bits.

Number of typical sequences: $2^{100 \times 0.469} = 2^{46.9} \approx 1.4 \times 10^{14}$.

Ratio: about 1 in 10^{16}.

One in 10^{16} is roughly one in ten quadrillion. Most conceivable sequences never happen.

Almost all conceivable sequences are impossible sequences! The biased coin concentrates probability on a tiny fraction of all possible outcomes—the typical ones, with about 90 heads out of 100.

Here is something wonderful. There are 2^n possible sequences, but only about 2^{nH} of them will actually occur. If $H < 1$—if the source is not maximally random—then most sequences never happen. Entropy measures how fast the set of "actually possible" outcomes grows.

2.11 A Glimpse of Things to Come

The AEP has immediate consequences for compression.

We've shown that only about 2^{nH} sequences actually occur. To label 2^{nH} things, we need only nH bits—we assign each typical sequence a unique binary name.

This is the key insight: if only 2^{nH} sequences occur, we need only nH bits to distinguish them, not the n bits we started with.

If we list only the typical sequences and assign each a short binary name, we can compress n symbols into approximately nH bits. The entropy H is not just a measure of uncertainty—it is the answer to a concrete question: *how much can this source be compressed?*

Chapter 4 will prove the source coding theorem:

- We cannot compress below H bits per symbol (on average).
- We can get arbitrarily close to H bits per symbol.

But even without the proof, the AEP makes it plausible. Entropy counts the typical sequences; naming typical sequences is compression.

2.12 *Four Views of Entropy*

Let us gather what entropy means:

1. *Average surprise per symbol.* Entropy is the expected self-information: $H(X) = \mathbb{E}[-\log_2 P(X)]$.

2. *Yes-or-no questions per symbol.* Entropy is the average number of binary questions needed to identify the outcome, using an optimal strategy.

3. *Minimum bits per symbol for compression.* Entropy is the fewest bits per symbol needed to encode messages from the source, on average.

4. *Rate at which typical sequences grow.* The number of typical length-n sequences is approximately 2^{nH}.

These four interpretations are not four different things. They are four ways of seeing the *same* thing. This convergence is what makes entropy fundamental.

These are not four different things. They are four ways of seeing the same thing. This is what it means for a concept to be fundamental: it appears wherever you look.

2.13 *Looking Ahead*

We started with vague intuitions about surprise and uncertainty. We demanded that any measure of information satisfy simple requirements—continuity, maximality at uniformity, additivity for independence, and proper grouping. Mathematics forced the formula $H(X) = -\sum p_i \log_2 p_i$ on us.

We computed entropies for coins, dice, and English text. We found that entropy has an operational meaning: it governs the rate at which typical sequences grow, and hence how much messages can be compressed. We paused to appreciate Shannon's 1948 paper and to reflect on the subjectivity of uncertainty.

But entropy measures uncertainty about a *single* variable. Communication involves *two* variables: the message sent and the message received. If a channel adds noise, the received message isn't the same as the sent message.

We need a way to measure not just uncertainty, but *shared* uncertainty. How much does knowing Y tell us about X? This quantity—*mutual information*—is the subject of the next chapter. It will turn out to be the master concept, even more fundamental than entropy itself.

If entropy measures what we don't know, mutual information measures what we can learn. From uncertainty to communication. From isolation to connection.

Let us continue.

3
Mutual Information and Communication Channels

Entropy measures uncertainty about one thing. But communication involves two things: a sender and a receiver. If I shout across a crowded room, you hear something—but is it what I said? Telephones crackle, radio signals fade, letters get smudged. The question is not how uncertain I am about my message (I know what I said), nor how uncertain you are about what you heard (that depends on the noise). The question is: how much of *your* uncertainty gets resolved by knowing *mine*?

This is the fundamental question of communication, and answering it requires a new concept. We need to measure not just uncertainty, but *shared* uncertainty—the information that connects two things. Shannon called this quantity *mutual information*, and it turns out to be the key to understanding everything about communication. It governs how much data can flow through a channel, how much compression is possible, and when reliable transmission becomes impossible.

Mutual information is arguably more fundamental than entropy itself. Entropy is mutual information between a variable and itself: $H(X) = I(X; X)$.

Before we dive into formulas, let us sit with the problem.

Imagine you are a spy. You have arranged a signal with your handler: if the coast is clear, you will place a lamp in your window; if danger approaches, no lamp. Simple enough. But your handler is watching from across a busy street. Car headlights reflect off windows. Street lamps flicker. Sometimes your handler sees a light when there is none; sometimes the light you placed goes unnoticed.

How much information actually transfers? This is not the same as asking how much information you sent (that is the entropy of your signal—one bit, since two possibilities). It is not the same as asking how uncertain your handler is about what they saw (that depends on how noisy the street is). It is asking: how much does what the handler observed *depend on* what you did?

The mathematics does not care if the message is life-or-death intelligence or idle chatter. It only cares whether the output depends on the input. Let us make this precise.

3.1 The Problem with Entropy Alone

Return to the filing cabinet metaphor from Chapter 2. You are looking for a document in a cabinet with many drawers. Your uncertainty about which drawer contains the document is $H(X)$.

Now suppose there is a second filing cabinet—the one your colleague is searching. Your colleague's uncertainty about their document is $H(Y)$.

Two entropies do not tell us whether the filing cabinets are related. The documents might be copies, or entirely independent.

Do these two uncertainties tell us anything about the relationship between your search and theirs? No. You could both be uncertain, yet searching for completely unrelated documents. Or you could both be uncertain, but if one of you finds your document, it reveals something about where the other's must be.

Consider this concrete example. Alice sends one of four messages with equal probability. She encodes them as the numbers 1, 2, 3, 4. Bob, at the other end of a communication channel, receives one of four signals with equal probability: the numbers 1, 2, 3, 4.

Question: Does this mean 2 bits of information transferred?

You might think so. After all, $H(X) = \log_2 4 = 2$ bits and $H(Y) = \log_2 4 = 2$ bits. But consider two scenarios.

Scenario A: The channel is perfect. If Alice sends 1, Bob receives 1. If Alice sends 2, Bob receives 2. And so on. In this case, knowing what Bob received tells Alice's message exactly. Two bits transferred.

Scenario B: The channel is completely broken. Alice sends her message, the universe ignores it, and Bob receives a random number independent of what Alice sent. In this case, knowing what Bob received tells us *nothing* about Alice's message. Zero bits transferred.

Both scenarios have $H(X) = 2$ bits and $H(Y) = 2$ bits. The individual entropies are identical. What differs is the *relationship* between X and Y.

This is the insight: what matters for communication is not how uncertain each party is in isolation, but how much one party's uncertainty is *explained* by the other's. We need a measure of shared information.

3.2 Mutual Information as Distance from Independence

Before we develop mutual information through conditional entropy, let me show you a striking fact that illuminates what we are really measuring.

Consider two random variables X and Y with joint distribution $P(X,Y)$. If they were independent, their joint distribution would be $P(X)P(Y)$—the product of the marginals. How "far" is the actual distribution from this independent version?

This "distance" is not a metric in the mathematical sense—it is not symmetric. But it measures something profound: how much the actual world differs from a hypothetical independent one.

The answer is the *Kullback-Leibler divergence* (or *relative entropy*):

$$D\left(P(X,Y)\|P(X)P(Y)\right) = \sum_{x,y} P(x,y)\log_2 \frac{P(x,y)}{P(x)P(y)}.$$

Here is the remarkable fact: this is *exactly* the mutual information.

$$I\left(X;Y\right) = D\left(P(X,Y)\|P(X)P(Y)\right).$$

Let us verify this. Starting from the definition of KL divergence:

$$\begin{aligned} D\left(P(X,Y)\|P(X)P(Y)\right) &= \sum_{x,y} P(x,y)\log_2 \frac{P(x,y)}{P(x)P(y)} \\ &= \sum_{x,y} P(x,y)\log_2 P(x,y) - \sum_{x,y} P(x,y)\log_2 P(x) - \sum_{x,y} P(x,y)\log_2 P(y) \\ &= -H\left(X,Y\right) + H\left(X\right) + H\left(Y\right). \end{aligned}$$

This is exactly Definition 3 of mutual information.

What does this mean? Mutual information measures *how surprised you would be* if you assumed X and Y were independent but they actually were not. It quantifies the "extra structure" in the joint distribution beyond what the marginals alone predict.

Think of it this way. If I give you the marginal distributions $P(X)$ and $P(Y)$ alone, your best guess for the joint distribution (without any other information) is $P(X)P(Y)$. The mutual information tells you how wrong this guess is. Zero mutual information means independence is a perfect guess. High mutual information means the variables are intertwined in ways you could not predict from the marginals.

KL divergence has the property that $D\left(P\|Q\right) \geq 0$, with equality iff $P = Q$. This immediately proves that $I\left(X;Y\right) \geq 0$, with equality iff X and Y are independent.

This perspective will become crucial in later chapters when we discuss coding and compression. For now, it provides another lens on what mutual information captures.

3.3 Conditional Entropy: What Remains Unknown

Let us build toward mutual information in stages. The first step is *conditional entropy*.

We defined conditional entropy briefly in Chapter 2. Now let us develop it more carefully. The conditional entropy $H\left(Y \mid X\right)$ answers the question: on average, how much uncertainty about Y remains after we learn X?

Formally:

Think of conditional entropy as "leftover uncertainty"—what you still don't know about Y even after learning X.

$$H\left(Y \mid X\right) = \sum_{x} P(x)\cdot H\left(Y|X=x\right) = -\sum_{x,y} P(x,y)\log_2 P(y|x).$$

The intuition is this. Before learning X, our uncertainty about Y is $H\left(Y\right)$. After learning X, some of that uncertainty may be resolved—or not, depending on how X and Y are related.

Extreme case 1: Y is completely determined by X. Then $H(Y \mid X) = 0$. Once we know X, we know Y exactly; no uncertainty remains.

Extreme case 2: Y is independent of X. Then $H(Y \mid X) = H(Y)$. Learning X tells us nothing about Y; all the original uncertainty remains.

General case: $0 \leq H(Y \mid X) \leq H(Y)$. Learning X reduces our uncertainty about Y by some amount, possibly zero.

Let us work through an example with actual numbers.

A Channel with One Confusing Case

Consider a simple channel where X can be 0 or 1, each with probability $1/2$.

If $X = 0$: the output $Y = 0$ deterministically.

If $X = 1$: the output Y is uniformly distributed over $\{1, 2\}$.

This channel transmits 0 perfectly but "confuses" the signal when transmitting 1.

What is the conditional entropy $H(Y \mid X)$?

When $X = 0$: Y is deterministic (always 0), so $H(Y|X = 0) = 0$.

When $X = 1$: Y is uniform over two values, so $H(Y|X = 1) = \log_2 2 = 1$ bit.

The conditional entropy is the average:

$$H(Y \mid X) = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1 = 0.5 \text{ bits.}$$

Half a bit of uncertainty remains about Y even after we learn X.

Now, what is $H(Y)$ by itself? We need the marginal distribution of Y.

$P(Y = 0) = P(X = 0) = 1/2$ (only $X = 0$ produces $Y = 0$).
$P(Y = 1) = P(X = 1) \cdot P(Y = 1|X = 1) = \frac{1}{2} \cdot \frac{1}{2} = 1/4$.
$P(Y = 2) = P(X = 1) \cdot P(Y = 2|X = 1) = \frac{1}{2} \cdot \frac{1}{2} = 1/4$.
So:

$$H(Y) = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{4} \log_2 \frac{1}{4} - \frac{1}{4} \log_2 \frac{1}{4} = \frac{1}{2} + \frac{1}{2} + \frac{1}{2} = 1.5 \text{ bits.}$$

The difference tells us something important:

$$H(Y) - H(Y \mid X) = 1.5 - 0.5 = 1 \text{ bit.}$$

This difference—what we learn about Y by knowing X—is exactly what we mean by mutual information.

More Worked Examples

Let us build intuition by computing mutual information for several different joint distributions.

Example 2: Correlated coin tosses. Suppose X is a fair coin (0 or 1 with probability 1/2 each). Given X, we generate Y as follows: with probability 0.9, $Y = X$; with probability 0.1, $Y = 1 - X$.

This is just the binary symmetric channel with $p = 0.1$. We have $H(X) = 1$ bit. The conditional entropy is $H(Y \mid X) = h(0.1) \approx 0.469$ bits. So:

$$I(X;Y) = 1 - 0.469 = 0.531 \text{ bits.}$$

The BSC with $p = 0.1$ transmits about half a bit per channel use. That is the cost of 10% noise.

Example 3: Three-way correlation. Suppose (X, Y) takes the values $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$ with probabilities $1/3$, $1/6$, $1/6$, $1/3$ respectively.

First, the marginals. $P(X = 0) = 1/3 + 1/6 = 1/2$. Similarly, $P(X = 1) = 1/2$, and by symmetry $P(Y = 0) = P(Y = 1) = 1/2$. So $H(X) = H(Y) = 1$ bit.

Now the joint entropy:

$$H(X,Y) = -\frac{1}{3}\log_2\frac{1}{3} - \frac{1}{6}\log_2\frac{1}{6} - \frac{1}{6}\log_2\frac{1}{6} - \frac{1}{3}\log_2\frac{1}{3}.$$

Computing: $-\frac{1}{3}\log_2\frac{1}{3} = \frac{1}{3}\log_2 3 \approx 0.528$ and $-\frac{1}{6}\log_2\frac{1}{6} = \frac{1}{6}\log_2 6 \approx 0.431$.

So $H(X,Y) \approx 2(0.528) + 2(0.431) = 1.918$ bits.

Using Definition 3:

$$I(X;Y) = H(X) + H(Y) - H(X,Y) = 1 + 1 - 1.918 = 0.082 \text{ bits.}$$

A small but nonzero correlation. The diagonal entries $(0,0)$ and $(1,1)$ are slightly more probable than expected under independence, creating a weak tendency for X and Y to agree.

Scenario	$I(X;Y)$
Independent uniform	0.000
Weak correlation (Ex. 3)	0.082
BSC, $p = 0.1$ (Ex. 2)	0.531
Deterministic $Y = X$	1.000
One reveals other partially	varies

Table 3.1: Mutual information for various joint distributions on two binary variables.

Example 4: Asymmetric dependence. Let X be uniform on $\{0,1,2,3\}$ (four values, $H(X) = 2$ bits). Define $Y = X \mod 2$ (the parity of X).

Then Y is uniform on $\{0,1\}$, so $H(Y) = 1$ bit. What is the mutual information?

Given X, we know Y exactly: $H(Y \mid X) = 0$. So $I(X;Y) = H(Y) - H(Y \mid X) = 1 - 0 = 1$ bit.

But notice: given Y, we do not know X exactly. If $Y = 0$, then $X \in \{0,2\}$—one bit of uncertainty remains. So $H(X \mid Y) = 1$ bit, and $I(X;Y) = H(X) - H(X \mid Y) = 2 - 1 = 1$ bit.

Same answer, as guaranteed by symmetry! But the information flows asymmetrically in a causal sense. X completely determines Y, but Y only partially determines X.

3.4 Mutual Information Defined

We now have all the pieces. Mutual information measures how much learning one variable tells us about another.

$$\boxed{I(X;Y) = H(Y) - H(Y \mid X)}$$

In words: mutual information is the reduction in uncertainty about Y that comes from learning X.

Read $I(X;Y)$ as "the mutual information between X and Y." The semicolon distinguishes it from conditional entropy.

For our example with the confusing channel: $I(X;Y) = 1.5 - 0.5 = 1$ bit. Despite the noise, one bit of information transfers through the channel.

But here is something remarkable. We could just as well have defined mutual information the other way around:

$$I(X;Y) = H(X) - H(X \mid Y).$$

This says: mutual information is the reduction in uncertainty about X that comes from learning Y.

Symmetry: A Surprising Fact

These two definitions are equal. Let us verify this for our example.

We have $H(X) = 1$ bit (uniform over $\{0, 1\}$).

What is $H(X \mid Y)$? This is the uncertainty about X after learning Y.

If $Y = 0$: then we know $X = 0$ with certainty. $H(X|Y = 0) = 0$.

If $Y = 1$: then we know $X = 1$ with certainty. $H(X|Y = 1) = 0$.

If $Y = 2$: then we know $X = 1$ with certainty. $H(X|Y = 2) = 0$.

So $H(X \mid Y) = 0$, and:

In this example, observing Y completely determines X. But this will not always be the case.

$$H(X) - H(X \mid Y) = 1 - 0 = 1 \text{ bit.}$$

The same answer! This is not a coincidence. It is a theorem:

Mutual information is symmetric: $I(X;Y) = I(Y;X)$.

This is philosophically striking. Information flow, as measured by mutual information, is the same in both directions—even when causation has a clear direction. Alice sends a message that determines Bob's received signal (up to noise). Causation flows from Alice to Bob. But mutual information is the same whether we ask "how much does knowing Alice's message tell us about Bob's signal?" or "how much does knowing Bob's signal tell us about Alice's message?"

You might say: "That's strange. Surely communication has a direction?" Yes, communication has a direction. But *correlation* does not. Mutual information measures correlation, not causation. It tells us how much one variable constrains the other, regardless of which caused which.

This symmetry will matter when we study statistical inference in later chapters. The information that data provides about parameters equals the information that parameters provide about data.

Three Equivalent Definitions

Let us collect the ways to express mutual information.

Definition 1 (learning about Y):

$$I(X;Y) = H(Y) - H(Y \mid X).$$

Definition 2 (learning about X):

$$I(X;Y) = H(X) - H(X \mid Y).$$

Definition 3 (symmetric form):

$$I(X;Y) = H(X) + H(Y) - H(X,Y).$$

The third form comes from the chain rule. Recall that $H(X,Y) = H(X) + H(Y \mid X)$, so:

$$H(Y) - H(Y \mid X) = H(Y) - (H(X,Y) - H(X)) = H(X) + H(Y) - H(X,Y).$$

The third form is particularly intuitive. The joint entropy $H(X,Y)$ measures our total uncertainty about both variables together. If X and Y were independent, this would equal $H(X) + H(Y)$—the sum of individual uncertainties. But if they are dependent, the joint uncertainty is less (we don't have to count the overlap twice). The mutual information is exactly this "saved" uncertainty—the overlap.

Definition 3 says: mutual information is the overlap between individual uncertainties. If X and Y were independent, $H(X,Y) = H(X) + H(Y)$ and $I(X;Y) = 0$.

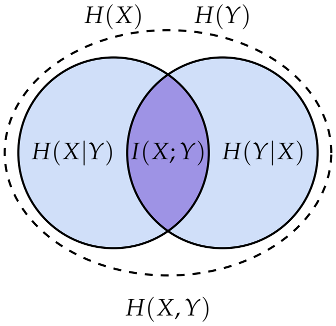


Figure 3.1: Entropy Venn diagram. The left circle is $H(X)$, the right is $H(Y)$. Their overlap is $I(X;Y)$. The dashed outer boundary encloses $H(X,Y)$.

The Venn Diagram Picture

It helps to visualize these relationships as a Venn diagram. Draw two overlapping circles, one for $H(X)$ and one for $H(Y)$. The overlap is $I(X;Y)$. The total area of both circles (counting overlap once) is $H(X,Y)$.

Reading off regions:

- Left circle minus overlap: $H(X \mid Y)$ (uncertainty about X that remains after learning Y)
- Right circle minus overlap: $H(Y \mid X)$ (uncertainty about Y that remains after learning X)
- Overlap: $I(X;Y)$ (shared information)
- Entire region: $H(X,Y)$ (total uncertainty about both)

This picture is useful, but treat it with some caution. Unlike actual set areas, which cannot be negative, there are quantum situations where the conditional entropy $H(X \mid Y)$ can be negative—meaning the "region" has negative area. For classical probability, the picture is safe.

Properties of Mutual Information

Let us establish the key properties.

Property 1: Non-negativity. $I(X;Y) \geq 0$, with equality if and only if X and Y are independent.

You cannot learn a negative amount from an observation. If variables are independent, learning one tells you nothing about the other: $I(X;Y) = 0$. Conversely, if $I(X;Y) = 0$, the variables must be independent.

The proof of non-negativity uses Gibbs' inequality, which says $\sum p \log p \geq \sum p \log q$ when p and q are probability distributions. We will use this result repeatedly.

Property 2: Symmetry. $I(X;Y) = I(Y;X)$.

Already discussed. Information flows equally in both directions, even when cause and effect do not.

Property 3: Upper bounds. $I(X;Y) \leq \min\{H(X), H(Y)\}$.

You cannot learn more about X than there is to know about X. Mutual information cannot exceed the entropy of either variable.

Property 4: Relationship to entropy. $I(X;X) = H(X)$.

The mutual information between a variable and itself is just its entropy. This makes sense: learning X tells you everything about X.

Property 5: Chain rule. $I(X;Y,Z) = I(X;Y) + I(X;Z \mid Y)$.

Mutual information can be decomposed. The information that (Y,Z) together provide about X equals the information Y provides, plus the additional information Z provides given Y.

Here $I(X;Z \mid Y)$ is the *conditional mutual information*: how much Z tells us about X when we already know Y.

3.5 The Chain Rule in Depth

The chain rule for mutual information deserves closer attention. It is the tool that lets us decompose complex dependencies into simpler pieces.

The chain rule is to mutual information what factorization is to probabilities. Both let us break complicated things into manageable parts.

Formally, the conditional mutual information between X and Z given Y is:

$$I(X;Z \mid Y) = H(X|Y) - H(X \mid Y,Z).$$

In words: how much does learning Z reduce our uncertainty about X, when we already know Y?

Let us prove the chain rule. Start with:

$$I(X;Y,Z) = H(X) - H(X \mid Y,Z).$$

We can write $H(X \mid Y,Z) = H(X|Y) - [H(X|Y) - H(X \mid Y,Z)] = H(X|Y) - I(X;Z \mid Y)$.

So:

$$I(X;Y,Z) = H(X) - H(X|Y) + I(X;Z \mid Y) = I(X;Y) + I(X;Z \mid Y).$$

A Worked Example of the Chain Rule

Suppose we have three binary variables. X is uniform on $\{0,1\}$. Given X, we generate Y through a BSC with $p = 0.1$ (so $Y = X$ with probability 0.9). Independently of Y given X, we generate Z through another BSC with $p = 0.2$.

Question: How much do (Y, Z) together tell us about X?

We already know $I(X; Y) = 1 - h(0.1) \approx 0.531$ bits.

What about $I(X; Z \mid Y)$? This asks: after learning Y, how much additional information does Z provide about X?

Since Y and Z are conditionally independent given X, knowing Y does not change the relationship between Z and X. But Y changes what we already know about X!

This requires care. Given Y, we already have partial information about X. The residual uncertainty $H(X|Y) = h(0.1) \approx 0.469$ bits (from the BSC calculation).

Now, given both Y and Z, how much uncertainty about X remains? This depends on whether Y and Z agree or disagree.

Rather than compute this directly (which involves careful bookkeeping), let us use a shortcut. By the chain rule applied the other way:

$$I(X; Y, Z) = I(X; Z) + I(X; Y \mid Z).$$

We know $I(X; Z) = 1 - h(0.2) \approx 0.278$ bits.

By symmetry of the setup and using the chain rule both ways:

$$I(X; Y, Z) = 0.531 + I(X; Z \mid Y) = 0.278 + I(X; Y \mid Z).$$

For this particular setup (conditionally independent channels), one can show $I(X; Y, Z) \approx 0.723$ bits. This gives $I(X; Z \mid Y) \approx 0.192$ bits.

Notice: $I(X; Z \mid Y) < I(X; Z)$. Learning Y first "steals" some of the information that Z would have provided. The two observations partially overlap in what they reveal about X.

3.6 Conditional Mutual Information: When Conditioning Helps or Hurts

A subtle point: conditioning can either increase or decrease mutual information. Let us explore when each occurs.

When Conditioning Decreases Mutual Information

Consider the example above. We had $I(X; Z) = 0.278$ bits, but $I(X; Z \mid Y) = 0.192$ bits. Conditioning on Y decreased the mutual information between X and Z.

This happens because Y and Z carry overlapping information about X. Once we know Y, some of what Z tells us is redundant.

When Conditioning Increases Mutual Information

Conditioning can "unlock" hidden relationships. X and Y might seem unrelated until you know Z.

More surprisingly, conditioning can also *increase* mutual information. Here is a classic example.

Let X and Y be independent fair coins. Let $Z = X \oplus Y$ (the XOR, or sum mod 2).

What is $I(X; Y)$? Zero—they are independent by construction.

What is $I(X;Y \mid Z)$? Given Z, knowing X completely determines Y (since $Y = X \oplus Z$). So $I(X;Y \mid Z) = H(Y|Z) - H(Y \mid X, Z) = 1 - 0 =$ 1 bit.

Conditioning on Z created a perfect correlation between X and Y! Before conditioning, they were independent. After conditioning on their XOR, each determines the other exactly.

This phenomenon is called *explaining away* in the causal inference literature. X and Y are marginally independent, but become dependent when we condition on their common effect Z.

The General Pattern

In general:

- Conditioning on a *common cause* tends to decrease mutual information (the cause explains the correlation).
- Conditioning on a *common effect* tends to increase mutual information (learning the effect links the causes).
- Conditioning on a *mediator* ($X \to Z \to Y$) tends to decrease mutual information (the mediator screens off the relationship).

These are tendencies, not absolute rules—the details depend on the specific distributions involved.

These rules follow from the structure of Bayesian networks. The *d*-separation criterion makes them precise.

3.7 Information Diagrams for Three Variables

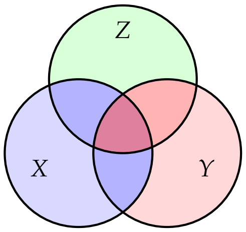


Figure 3.2: Information diagram for three variables. The central region where all three overlap represents the *co-information*, which can be negative.

The Venn diagram picture extends naturally to three variables, but with an important twist.

For three variables X, Y, Z, draw three overlapping circles. The regions represent various entropy quantities:

- Each circle: $H(X)$, $H(Y)$, $H(Z)$
- Pairwise overlaps (minus center): conditional mutual informations
- The central region: a quantity called *co-information* or *interaction information*

Here is the twist: the central region can be *negative*.

For the XOR example ($Z = X \oplus Y$ with independent X, Y), the co-information is -1 bit. The Venn diagram "area" is negative. This is not a failure of the picture—it reflects a real phenomenon. The XOR creates a kind of anti-redundancy: X, Y, and Z together are less informative than you would expect from their pairwise relationships.

Negative co-information signals synergy: the whole is less than the sum of its parts. This occurs when variables share information only in three-way combinations.

For more than three variables, the picture becomes even more complex. There are $2^n - 1$ regions for n variables, and many can be negative.

The Venn diagram remains a useful mental picture for two variables, but should be used with caution for three or more.

3.8 *The Wayward Translator*

Now we are ready for communication channels. Let us develop a metaphor that will serve us through this chapter and beyond.

Imagine you have hired a translator. You speak English; your audience speaks French. You tell the translator your message; the translator speaks to the audience. The question is: how much of your meaning gets through?

The wayward translator: not malicious, just occasionally confused. This is the essence of a noisy channel.

This translator has an annoying habit. They translate correctly most of the time, but occasionally substitute a random word. Not maliciously—they don't try to change your meaning—they just sometimes get confused. "The ship sails at dawn" might become "The ship sails at dusk," or "The sheep sails at dawn," or (rarely) something completely garbled.

What we call a *communication channel* is a mathematical model of such a translator. It specifies exactly how likely each possible output is for each possible input.

Formally, a channel is a conditional probability distribution $P(Y|X)$:

- *Input alphabet* $\mathcal{X}$: what the sender can say
- *Output alphabet* $\mathcal{Y}$: what the receiver can hear
- *Transition probabilities* $P(y|x)$: how likely is output y given input x

We assume the channel is *memoryless*: each use is independent of previous uses. Our translator doesn't hold grudges; each word is a fresh opportunity for confusion.

Let us examine three fundamental channels, each illustrating a different type of noise.

3.9 *The Binary Symmetric Channel*

The simplest noisy channel: you send a bit, and with some probability, the channel flips it.

The *binary symmetric channel* (BSC) with crossover probability p:

- Input: $\{0, 1\}$
- Output: $\{0, 1\}$
- $P(Y = X) = 1 - p$ (correct transmission)
- $P(Y \neq X) = p$ (bit flip)

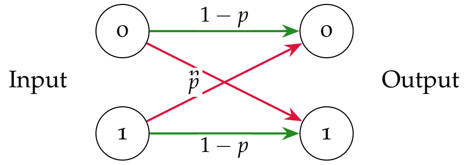


Figure 3.3: The binary symmetric channel (BSC) with crossover probability p. Each bit is flipped with probability p, transmitted correctly with probability $1 - p$.

The channel is "symmetric" because both bits are equally likely to flip. Our wayward translator makes the same kind of error regardless of which word you said.

Mutual Information for the BSC

Let us compute the mutual information. Suppose the input X has probability q for 1 and probability $1 - q$ for 0.

Step 1: Compute $H(Y \mid X)$.

Given X, the output Y is either X (with probability $1 - p$) or $1 - X$ (with probability p). This is a binary distribution with parameter p, regardless of what X was. So:

$$H(Y \mid X) = h(p) = -p \log_2 p - (1-p) \log_2 (1-p).$$

This quantity is called the *equivocation*—the uncertainty about the output given the input. It measures the "confusion" added by the channel.

The equivocation depends only on the channel's noise level p, not on the input distribution. This is a special property of the BSC.

Step 2: Compute $H(Y)$.

What is the distribution of Y? We have:

$$P(Y=1) = P(X=1)P(Y=1|X=1) + P(X=0)P(Y=1|X=0) = q(1-p) + (1-q)p.$$

Let $r = q(1-p) + (1-q)p = q - 2pq + p = p + q(1-2p)$.

Then $H(Y) = h(r)$.

Step 3: Mutual information.

$$I(X;Y) = H(Y) - H(Y \mid X) = h(p + q(1-2p)) - h(p).$$

For the special case of *uniform input* ($q = 1/2$):

$$r = p + \frac{1}{2}(1-2p) = p + \frac{1}{2} - p = \frac{1}{2}.$$

So $H(Y) = h(1/2) = 1$ bit, and:

$$I(X;Y) = 1 - h(p).$$

p	$I(X;Y)$
0	1.000
0.01	0.919
0.05	0.714
0.10	0.531
0.20	0.278
0.30	0.119
0.40	0.029
0.50	0.000

Table 3.2: Mutual information for BSC with uniform input, for various crossover probabilities.

Putting in Numbers

Let us see what these formulas say for specific values of p.

Perfect channel ($p = 0$): $I(X;Y) = 1 - h(0) = 1 - 0 = 1$ bit.

The full bit of input information gets through. No confusion at all.

Slightly noisy channel ($p = 0.1$): $h(0.1) = -0.1 \log_2 0.1 - 0.9 \log_2 0.9 \approx 0.469$.

$I(X;Y) \approx 1 - 0.469 = 0.531$ bits.

About half the information is lost to noise.

Very noisy channel ($p = 0.3$): $h(0.3) \approx 0.881$.

$I(X;Y) \approx 1 - 0.881 = 0.119$ bits.

Most information is lost. The output barely depends on the input.

Completely random channel ($p = 0.5$): $h(0.5) = 1$.

$I(X;Y) = 1 - 1 = 0$ bits.

The output is completely independent of the input! The channel might as well not exist.

Perfect inverter ($p = 1$): $h(1) = 0$.

$I(X;Y) = 1 - 0 = 1$ bit.

Wait—full information? Yes! A channel that always flips the bit is just as good as one that never flips. You know exactly what came in by inverting what came out. It's only when the channel is genuinely random that information is lost.

This is a philosophically important point. A channel that lies consistently is no worse than one that tells the truth—as long as you know which you have. Randomness, not wrongness, destroys information.

The worst BSC is $p = 0.5$, not $p = 1$. A reliable inverter is useful; a coin flip is not.

3.10 The Binary Erasure Channel

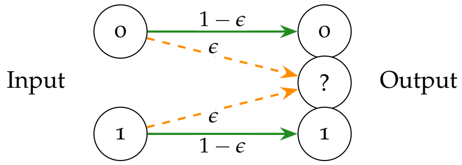


Figure 3.4: The binary erasure channel (BEC) with erasure probability ϵ. Bits are either transmitted correctly or erased (replaced with ?).

Our wayward translator has a more honest cousin: one who, instead of making mistakes, simply says "I didn't catch that."

The *binary erasure channel* (BEC) with erasure probability ϵ:

- Input: $\{0, 1\}$
- Output: $\{0, 1, ?\}$
- $P(Y = X) = 1 - \epsilon$ (correct transmission)
- $P(Y =?) = \epsilon$ (erasure)

The "?" means "I don't know what was sent." Unlike the BSC, the erasure channel never lies—it either tells the truth or admits ignorance.

Mutual Information for the BEC

This channel has a beautiful property: when you receive 0 or 1, you're *certain* it's correct. The channel never makes errors, only erasures.

Let us compute mutual information for uniform input.

Step 1: Compute $H(X \mid Y)$.

This is easier than $H(Y \mid X)$ for the BEC. Given what we received:

- If $Y = 0$: we know $X = 0$ with certainty. $H(X|Y = 0) = 0$.
- If $Y = 1$: we know $X = 1$ with certainty. $H(X|Y = 1) = 0$.
- If $Y =?$: we have no information. With uniform input, $H(X|Y =?) =$ 1 bit.

The probabilities: $P(Y=0) = P(Y=1) = (1-\epsilon)/2$ and $P(Y=?) = \epsilon$.

So:

$$H(X \mid Y) = \frac{1-\epsilon}{2} \cdot 0 + \frac{1-\epsilon}{2} \cdot 0 + \epsilon \cdot 1 = \epsilon.$$

Step 2: Mutual information.

$$I(X;Y) = H(X) - H(X \mid Y) = 1 - \epsilon.$$

The BEC is "honest about its ignorance." When it knows, it's certain; when it doesn't know, it says so.

Beautifully simple. The mutual information is exactly the probability that the bit gets through.

For $\epsilon = 0$: $I(X;Y) = 1$ bit (perfect channel).

For $\epsilon = 0.3$: $I(X;Y) = 0.7$ bits.

For $\epsilon = 1$: $I(X;Y) = 0$ bits (everything erased).

BSC vs. BEC

Compare the two channels at the same "error rate."

At $p = \epsilon = 0.1$:

- BSC: $I(X;Y) = 1 - h(0.1) \approx 0.531$ bits
- BEC: $I(X;Y) = 1 - 0.1 = 0.9$ bits

The erasure channel transmits more information! This makes intuitive sense. With the BSC, you receive every bit but can't be sure which are wrong. With the BEC, you receive fewer bits but know exactly which ones you have. Certainty about what you know is valuable.

Knowing when errors occur (the BEC) is much better than having errors occur unpredictably (the BSC).

This has practical implications. When you download a file and some packets are lost, the protocol can ask for retransmission. But if packets arrive corrupted without any indication they're wrong, you might not even know you have errors.

3.11 The Gaussian Channel

Real-world communication—radio, cell phones, wifi—doesn't use discrete bits directly. It uses continuous signals corrupted by continuous noise.

The *additive white Gaussian noise* (AWGN) channel:

$$Y = X + Z, \quad \text{where } Z \sim \mathcal{N}(0, N) \text{ is independent of } X.$$

You send a signal X; it arrives corrupted by Gaussian noise Z with variance N.

"White" noise means equal power at all frequencies. "Gaussian" means normally distributed. This is the standard model for thermal noise in electronic systems.

Our wayward translator adds random static to everything you say. The louder you speak, the more reliably you're heard—but there's a catch.

The power constraint: You can't shout arbitrarily loud. There's a limit on the average signal power: $\mathbb{E}[X^2] \leq P$.

The key quantity is the *signal-to-noise ratio*: $\text{SNR} = P/N$.

Capacity of the Gaussian Channel

Here is a remarkable result, which we state now and prove in Chapter 6.

For the Gaussian channel with power constraint P and noise variance N, the maximum mutual information (achieved when X is Gaussian with variance P) is:

$$C = \frac{1}{2} \log_2 \left(1 + \frac{P}{N}\right) \quad \text{bits per channel use.}$$

This is the *capacity* of the Gaussian channel. It tells us the maximum rate at which information can flow through.

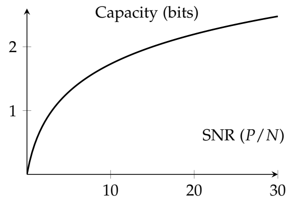


Figure 3.5: Capacity of the Gaussian channel vs. signal-to-noise ratio. The capacity grows logarithmically with SNR.

Let us put in numbers.

SNR = 1 (signal and noise have equal power): $C = \frac{1}{2}\log_2(2) = 0.5$ bits.

SNR = 10: $C = \frac{1}{2}\log_2(11) \approx 1.73$ bits.

SNR = 100: $C = \frac{1}{2}\log_2(101) \approx 3.33$ bits.

SNR = 1000: $C = \frac{1}{2}\log_2(1001) \approx 4.98$ bits.

Notice the logarithmic growth. Increasing SNR by a factor of 10 adds only about 1.66 bits of capacity. You cannot simply blast your way to infinite throughput by shouting louder. The universe imposes fundamental limits.

This formula—the Shannon capacity of the Gaussian channel—governs every wireless signal you have ever received. Your cell phone's data rate, your wifi bandwidth, satellite communications, deep space probes talking to Earth—all are constrained by this equation. When engineers talk about approaching the "Shannon limit," this is what they mean.

3.12 The Data Processing Inequality

We now come to one of the most important results in information theory. It tells us something that sounds obvious but has profound implications: you cannot create information by processing.

Markov Chains

First, some terminology. We say that $X \to Y \to Z$ forms a *Markov chain* if:

$$P(Z|X, Y) = P(Z|Y).$$

In words: once you know Y, learning X tells you nothing new about Z. All the information that X has about Z "flows through" Y.

The notation $X \to Y \to Z$ is suggestive: information from X passes through Y to reach Z. But remember, we're talking about statistical dependence, not necessarily causation.

Example: Alice sends a message X through a noisy channel to Bob, producing Y. Bob then processes Y to produce Z. Here $X \to Y \to Z$ is a Markov chain: once we know what Bob received (Y), the original message (X) tells us nothing more about Bob's processed output (Z).

The Theorem

If $X \to Y \to Z$ is a Markov chain, then:

$$I(X;Z) \leq I(X;Y).$$

Processing cannot increase information about the source.

Proof: Use the chain rule for mutual information.

$$I(X;Y,Z) = I(X;Y) + I(X;Z \mid Y).$$

We can also expand the other way:

$$I(X;Y,Z) = I(X;Z) + I(X;Y \mid Z).$$

Since $X \to Y \to Z$ is Markov, knowing Y makes X and Z conditionally independent: $I(X;Z \mid Y) = 0$.

From the first expansion: $I(X;Y,Z) = I(X;Y) + 0 = I(X;Y)$.

Substituting into the second:

$$I(X;Y) = I(X;Z) + I(X;Y \mid Z).$$

Since $I(X;Y \mid Z) \geq 0$ (mutual information is non-negative):

$$I(X;Y) \geq I(X;Z). \quad \square$$

The proof is short but the implications are vast. No algorithm can recover information lost in transmission.

What This Means

The data processing inequality says: if information is lost in transmission from X to Y, no amount of clever processing can recover it.

Consider the television show trope where investigators "enhance" a blurry security camera image. They zoom in on a reflection, sharpen, enhance, and suddenly see a face clearly enough to identify the criminal. This is fiction. The data processing inequality explains why.

"Enhance!" makes for good television but bad science. You cannot create detail that the image never contained.

If X is the true scene and Y is the blurry image, then $I(X;Y)$ is the information the image contains about the scene. Any processing $Y \to Z$ (enhancement, sharpening, interpolation) cannot increase this. The best you can do is $I(X;Z) = I(X;Y)$—preserve all the information—but you cannot exceed it.

What enhancement algorithms actually do is make *guesses* based on prior knowledge (faces have certain shapes, text has certain patterns). They hallucinate details that are plausible given what is known. Sometimes the guesses are right; sometimes they are wrong. But they are not recovering information from the image—they are adding assumptions.

More Applications of the Data Processing Inequality

The data processing inequality appears throughout science and engineering. Let us examine a few more examples.

Telephone game. In the children's game, a message passes from person to person. Each transmission adds noise. By the data processing inequality, each step can only lose information:

$$I(X; Y_n) \leq I(X; Y_{n-1}) \leq \cdots \leq I(X; Y_1).$$

The final message cannot contain more information about the original than any intermediate version did. This explains why the final message is often unrecognizable—each step loses a little, and losses accumulate.

The telephone game is a Markov chain: $X \to Y_1 \to Y_2 \to \cdots$. Information degrades monotonically.

Lossy compression. When you convert a WAV audio file to MP3, information is discarded. If you then convert the MP3 to a different format, you cannot recover the lost frequencies. The data processing inequality makes this precise: the information about the original recording in any downstream format is bounded by the information in the MP3.

This is why audio engineers keep "lossless masters." Once you compress, you cannot uncompress back to the original quality.

Scientific measurement. When a physicist measures a quantity θ using instrument X that produces noisy reading Y, and then summarizes the data with statistic $T(Y)$, the data processing inequality says:

$$I(\theta; T(Y)) \leq I(\theta; Y).$$

No summary can contain more information about θ than the raw data. If T is sufficient, equality holds; otherwise, the summary discards relevant information.

Neural network layers. Consider a deep neural network trained for classification. Let X be the input image, Y be the activations of layer k, and Z be the output classification. The data processing inequality says:

$$I(X; Z) \leq I(X; Y).$$

Early layers preserve more information about the input; later layers have compressed it down to what is relevant for the task. This is sometimes called the "information bottleneck" view of deep learning.

The information bottleneck hypothesis suggests that networks learn by first memorizing, then compressing to retain only task-relevant features.

When Does Equality Hold?

The data processing inequality is an inequality. When is it tight?

Equality holds when $X \to Y \to Z$ and Z is a *sufficient statistic* for X given Y. That is, Z captures everything about Y that is relevant to X.

More precisely: $I(X; Z) = I(X; Y)$ if and only if $X \to Z \to Y$ also forms a Markov chain. In that case, Y and Z contain exactly the same information about X, just possibly in different forms.

Sufficient Statistics

The data processing inequality has an important corollary. Sometimes processing loses *no* information.

A statistic $T(Y)$ is *sufficient* for X if $X \to T(Y) \to Y$ forms a Markov chain. That is, once you know $T(Y)$, the original data Y tells you nothing more about X.

For sufficient statistics, the data processing inequality becomes an equality:

$$I\left(X; T(Y)\right) = I\left(X; Y\right).$$

A sufficient statistic compresses the data without losing any information about the parameter of interest. This is the ideal of statistical inference.

You can throw away the original data and keep only the sufficient statistic, losing nothing about X.

Example: Suppose $Y_1, \ldots, Y_n$ are independent draws from a normal distribution with unknown mean μ and known variance σ^2. The sample mean $\bar{Y} = \frac{1}{n} \sum Y_i$ is sufficient for μ.

Why? Because $P(Y_1, \ldots, Y_n | \bar{Y}, \mu) = P(Y_1, \ldots, Y_n | \bar{Y})$. Once you know the sample mean, knowing μ doesn't help you predict the individual observations any better.

This means: if your goal is to learn about μ, you can replace all n data points with a single number—the sample mean—and lose nothing. The sample mean contains everything the data has to say about μ.

3.13 *A Mathematical Theory of Communication*

Let us pause for history.

In the summer of 1948, Claude Shannon was 32 years old. He worked at Bell Telephone Laboratories in New Jersey, where the practical problem was telephone communication. How much information could flow through a wire? How should signals be encoded to resist noise? These were engineering questions, but Shannon saw something deeper.

Bell Labs in the 1940s was extraordinary. Shannon's colleagues included the inventors of the transistor, information theory, and the laser.

His paper, "A Mathematical Theory of Communication," appeared in two parts in the Bell System Technical Journal that July and October. It was 55 pages long and invented a field.

What Shannon did was audacious. He separated the problem of communication into distinct layers. The *meaning* of a message—what it's about, whether it's true, whether it's important—was declared irrelevant to the engineering problem. What mattered was the statistical structure of the source and the physical properties of the channel.

"The fundamental problem of communication," Shannon wrote, "is that of reproducing at one point either exactly or approximately a message selected at another point. Frequently the messages have *meaning*; that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem."

This was strategic clarity. By ignoring meaning, Shannon could focus on what wires actually need to transmit: bits, regardless of what those bits represent.

Shannon's decision to ignore semantics was controversial but brilliant. It allowed precise theorems where before there was only vague intuition.

Mutual information appears in Section 8 of the paper, introduced almost casually: "The rate of actual transmission, R, is the entropy of the source less the equivocation." In our notation: the rate is $I(X;Y) = H(X) - H(X \mid Y)$.

Shannon called $H(X \mid Y)$ the "equivocation"—the average ambiguity of the received signal. It is a perfect word. To equivocate is to be unclear, to say something that could mean multiple things. The equivocation measures how unclear the channel makes the message.

The paper had immediate impact. Engineers recognized that Shannon had given them the right concepts and the right theorems. But the proofs were existence proofs—Shannon showed that good codes *exist* without saying how to construct them. Finding codes that actually achieved Shannon's limits occupied the next fifty years. We'll take up that story in Chapters 7 and 8.

Shannon himself was playful about his work. He built maze-solving robots, juggling machines, and a calculator that operated in Roman numerals. He unicycled through the halls of Bell Labs. He worked on problems because they were interesting, not because they were useful—and the most interesting problems often turned out to be the most useful.

3.14 What Does "Information Transfer" Really Mean?

Let us turn philosophical. We speak of information "flowing" through channels, of data being "transmitted" and "received." But what is actually happening?

When you speak into a telephone, sound waves vibrate a membrane, electrical signals propagate through wires, and speakers vibrate on the other end. Something physical clearly moves. But when we say "one bit of information transferred," we don't mean a bit-object traveled from sender to receiver.

Information is not a substance that flows. It is a relationship between random variables—a correlation structure.

Mutual information measures *correlation*, not *causation*. $I(X;Y) > 0$ means that learning X tells you something about Y—but it doesn't mean X caused Y, or that anything traveled from X to Y.

Consider this thought experiment. Alice flips a coin in New York. Bob, in London, flips a completely independent coin. We have $I(X;Y) = 0$. Now suppose they flip the *same* coin remotely—say, by prior arrangement, they both flip when a shared random beacon triggers. Now $I(X;Y) > 0$, even though no signal traveled between them at the moment of flipping.

The mutual information changed because the *correlation structure* changed, not because information physically moved at that instant.

In quantum mechanics, this becomes even stranger. Entangled particles can have $I(X;Y) > 0$ without any signal passing between them—and provably, no signal can pass faster than light. The correlation exists, but it wasn't "transmitted" in any physical sense.

Three Views of Information

Let me sketch three philosophical positions on what information "is." None is definitively correct; each illuminates a different aspect of the concept.

The epistemic view. Information is about knowledge. Entropy measures our ignorance; mutual information measures how much learning one thing teaches us about another. On this view, information is inherently observer-relative. The entropy of a coin is 1 bit to you but 0 bits to someone who has seen it. Information doesn't exist "out there" in the world—it exists in the relationship between an observer and the world.

The epistemic view traces back to Jaynes, who argued that probability itself is a measure of knowledge, not a physical quantity.

This view explains why Shannon could ignore semantics. What matters is not what messages *mean* but how much we can learn from them. A gibberish message and a profound message can have the same entropy if they are equally unpredictable.

The physical view. Information is as physical as energy or momentum. When you erase a bit of memory, you must dissipate energy (Landauer's principle). When you measure a quantum system, you gain information and disturb the system. The universe has a finite information capacity, bounded by the Bekenstein limit. On this view, information is not merely about what we know—it is a fundamental physical quantity.

The physical view connects information theory to thermodynamics. Maxwell's demon, who seems to violate the second law by sorting fast and slow molecules, fails because acquiring information about molecules has an entropic cost. The demon's memory fills up, and erasing it generates heat. Information and entropy are two sides of the same coin.

The structural view. Information is pattern, structure, constraint. A random string has no structure and hence no information (in the Kolmogorov sense). A structured sequence—even if unpredictable—has information *in* it because it has low Kolmogorov complexity. On this view, mutual information measures how much structure is shared between two systems.

Kolmogorov complexity measures the length of the shortest program that produces a string. This is incomputable in general but conceptually powerful.

Each view captures something true. The epistemic view explains why information theory works for communication. The physical view explains why information cannot be created or destroyed without physical consequences. The structural view explains why we find information theory useful for understanding complex systems.

Does Information Flow?

We say information “flows” through channels. But consider carefully: what flows?

In a telephone call, electromagnetic signals flow through wires. But the information—the mutual information between Alice’s speech and Bob’s perception—doesn’t “flow” in the same sense. It is a statistical relationship that exists because of how the physical channel works.

Here is an analogy. Suppose Alice and Bob each have a copy of the same book. There is high mutual information between Alice’s copy and Bob’s copy—knowing one tells you a lot about the other. But nothing flows between them. The mutual information exists because of their common origin (the publisher), not because of any ongoing connection.

When we say “Alice sent Bob one bit of information,” we mean something operational: Alice made a choice, the channel transformed it according to its transition probabilities, and Bob’s observation became correlated with Alice’s choice. The mutual information quantifies this correlation. But nothing called “information” traveled from Alice to Bob like a package.

Mutual information is symmetric: $I(X;Y) = I(Y;X)$. If information “flowed,” we would expect a direction. But the mathematics is agnostic about direction.

This might seem like philosophical hairsplitting, but it matters for understanding what information theory can and cannot tell us. Information theory tells us about correlations, not about mechanisms. It tells us that a channel can support a certain rate of reliable communication, but it does not tell us how the channel works physically.

Meaning and Information

Shannon explicitly excluded meaning from his theory. “The semantic aspects of communication,” he wrote, “are irrelevant to the engineering problem.”

This was strategic brilliance, not philosophical naïveté. By ignoring meaning, Shannon could prove powerful theorems that apply to *any* communication system, regardless of what is being communicated. The same capacity formula applies whether you are transmitting Shakespeare or grocery lists.

A channel cannot tell whether the bits passing through it represent poetry or noise. It only knows the statistical structure.

But we might ask: is there a deeper theory that includes meaning? Information theory tells us how many bits can be transmitted, but not which bits matter. When you compress an image, the algorithm doesn’t know that the face is more important than the background—unless you tell it.

Some researchers have proposed *semantic information* theories that try to quantify not just how much is transmitted but how much of *what matters* is transmitted. These are active research areas, but none has achieved the universality and mathematical elegance of Shannon’s theory.

Perhaps meaning is irreducibly context-dependent in a way that resists general mathematical treatment. Or perhaps we simply haven't found the right framework yet.

The Unreasonable Effectiveness of Information Theory

"The universe doesn't care about information," one might say. "Information is what *we* care about." Entropy and mutual information are measures of our knowledge and ignorance, our ability to predict and infer. They are not properties of physical systems independent of observers, but relationships between physical systems and those who observe them.

And yet—information theory works. It predicts the limits of compression and transmission with astonishing accuracy. It provides the mathematical foundation for modern communication systems. It connects to physics through thermodynamics and quantum mechanics.

Wigner wrote about "the unreasonable effectiveness of mathematics in the natural sciences." Information theory is another example.

This is not a defect of information theory but a feature. Shannon's theory works precisely because it makes no metaphysical claims about the nature of information. It defines useful quantities, proves theorems about them, and applies them to engineering. The philosophical questions remain open, but they don't block the engineering.

Perhaps the deepest lesson is this: you don't need to know what information *is* to know what you can *do* with it. Shannon gave us a calculus of communication that works regardless of our philosophical commitments. The theorems are true whether information is epistemic, physical, or something else entirely.

3.15 Looking Forward: The Source Coding Theorem

We have now built two fundamental tools. Entropy measures uncertainty about a single source. Mutual information measures the shared information between two variables—the information that survives transmission through a channel.

These are not mere abstractions. They answer concrete questions about the physical world.

In the next chapter, we turn to the first of Shannon's great theorems: the *source coding theorem*. It answers a question people have asked since the invention of writing: How much can a message be compressed?

The source coding theorem: entropy is exactly the limit of compression. Not approximately—exactly.

Here is a preview. Suppose you have a source producing symbols with entropy H bits per symbol. The source coding theorem says:

- You cannot compress below H bits per symbol on average.
- You can get arbitrarily close to H bits per symbol with sufficiently clever coding.

The entropy is *exactly* the compression limit. Not an approximation, not an upper bound, but the precise answer.

This is remarkable. Entropy, which we derived from abstract requirements about how uncertainty should behave, turns out to be the answer to a concrete engineering problem. The same quantity appears in compression, in communication, in thermodynamics, in statistical inference. It is not that we defined entropy to equal the compression limit; it is that the compression limit *must* equal entropy, because both are measuring the same thing.

The source coding theorem also sets the stage for the even more surprising channel coding theorem of Chapter 7: that reliable communication over noisy channels is possible, as long as we communicate slowly enough. The maximum reliable rate is the channel capacity—the maximum mutual information between input and output.

Both theorems will use the ideas we've developed: entropy, mutual information, typical sequences. The conceptual foundations are in place. Now we will see what they can build.

From uncertainty to shared uncertainty. From isolation to connection. From the abstract mathematics of probability to the concrete limits of what can be communicated.

Let us continue.

4
The Source Coding Theorem

How much can a message be compressed?

This question has been asked since the invention of writing. Ancient scribes developed abbreviations. Medieval copyists used contractions. Telegraph operators invented codes—Morse gave common letters short patterns, rare letters long ones. Everyone who has tried to fit a thought into a limited space has wrestled with this problem.

But the question has a different character depending on how you ask it. "How do I compress this particular message?" is an engineering question with many answers. "What is the best compression possible for any message from this source?" is a scientific question. It asks not what we can do, but what the universe allows.

In 1948, Shannon gave the definitive answer. He proved that there is a fundamental limit on compression, that this limit can be achieved, and that it equals exactly the entropy of the source. Not approximately, not in some limit—exactly. Entropy is the compression limit.

The source coding theorem answers a question people have asked for millennia. The answer turns out to be entropy—the same quantity that arose from our abstract requirements about uncertainty.

This is Shannon's first major theorem, and it deserves to be treated as a revelation. We have been building toward it through Chapters 2 and 3, developing entropy and mutual information as measures of uncertainty. Now we discover that entropy is not merely a mathematical convenience. It is the answer to a concrete physical question.

Let us begin with the question itself.

4.1 The Compressibility Question

Imagine you are a telegraph operator in 1880. You pay by the symbol. English text is obviously redundant—"q" is always followed by "u," the letter "e" appears far more often than "z," certain words repeat constantly. Surely you can represent messages more efficiently than by sending every character verbatim.

But how efficiently? Can you compress English to half its length? A third? Is there a theoretical minimum, or can clever enough schemes compress arbitrarily far?

This is not an idle question. The answer determines the capacity of communication infrastructure, the storage requirements for data, the feasibility of applications we haven't imagined yet.

The telegraph operators of the 19th century were practical information theorists. They developed compression schemes by intuition; Shannon provided the theory.

Let us make the question precise.

The Setup

A source produces a sequence of symbols $X_1, X_2, \ldots, X_n$ from some alphabet $\mathcal{X}$. For simplicity, we will mostly consider *independent and identically distributed* (i.i.d.) sources: each X_i is drawn independently from the same probability distribution $P(x)$.

We want to encode this sequence as a binary string—a sequence of 0s and 1s. We want to be able to decode perfectly: given the binary string, we should be able to recover $X_1, \ldots, X_n$ exactly. This is called *lossless compression*.

Lossless means no information is lost. We will briefly discuss lossy compression—where we allow some distortion—at the end of the chapter.

The *rate* of a code is the average number of bits per source symbol:

$$R = \frac{\text{expected length of encoded string}}{n}.$$

The question becomes: what is the minimum achievable rate?

Codes and Decodability

Not every assignment of binary strings to source sequences makes sense. Consider trying to encode two symbols, A and B, as follows: A → 0, B → 00. Now suppose you receive the string 00. Was the message "AA" or "B"? You cannot tell. The code is ambiguous.

A code is *uniquely decodable* if every encoded string corresponds to exactly one source sequence. We will only consider uniquely decodable codes.

A uniquely decodable code lets us recover the original message without ambiguity. This is the minimum requirement for any useful code.

A particularly useful class is *prefix codes*, where no codeword is a prefix of another. Our bad example above fails because "0" is a prefix of "00." In a prefix code, as soon as you see a complete codeword, you know it is complete—no need to look ahead.

Prefix codes have a beautiful structure: they correspond to binary trees. Each codeword is a path from the root to a leaf, with 0 meaning "go left" and 1 meaning "go right." The prefix property ensures that every codeword reaches a leaf—no codeword corresponds to an internal node.

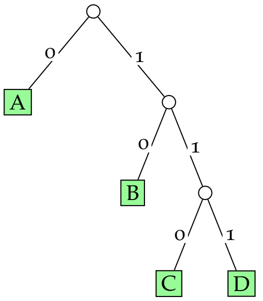


Figure 4.1: A prefix code as a binary tree. Codewords: A → 0, B → 10, C → 110, D → 111. No codeword is a prefix of another.

Here is a fact that simplifies our analysis enormously. The *Kraft inequality* states that for any prefix code with codeword lengths $\ell_1, \ldots, \ell_m$:

$$\sum_{i=1}^{m} 2^{-\ell_i} \leq 1.$$

Conversely, if a set of lengths satisfies the Kraft inequality, there exists a prefix code with those lengths. More remarkably, for any uniquely decodable code—even non-prefix codes—there exists a prefix code with the same lengths.

This means we lose nothing by restricting our attention to prefix codes. They are the natural objects of study.

The Kraft inequality connects codeword lengths to a geometric constraint. The "volume" of all codewords cannot exceed 1.

4.2 Why Entropy Appears: A First Bound

Before proving the full theorem, let us see intuitively why entropy must be the limit.

Symbol-by-Symbol Codes

The simplest approach is to encode each source symbol independently, assigning a fixed binary codeword to each symbol in $\mathcal{X}$. For a prefix code with codeword lengths $\ell_1, \ldots, \ell_m$ assigned to symbols with probabilities $p_1, \ldots, p_m$, the expected length per symbol is:

$$L = \sum_{i=1}^{m} p_i \ell_i.$$

Can we choose lengths to minimize L? There is a constraint from the Kraft inequality. Using Lagrange multipliers (or a direct argument with Jensen's inequality), one can show:

$$L \geq H(X) = -\sum_i p_i \log_2 p_i.$$

The expected codeword length is at least the entropy.

Moreover, equality holds when $\ell_i = -\log_2 p_i$—that is, when we assign $\log_2(1/p_i)$ bits to a symbol with probability p_i. Rare symbols get long codewords; common symbols get short ones.

This inequality is sometimes called the "noiseless coding theorem" for symbol-by-symbol codes. Shannon's full theorem extends it to block codes.

But there is a problem. Codeword lengths must be integers. If $-\log_2 p_i$ is not an integer, we cannot achieve equality.

A Worked Example: Near-Optimal Coding

Consider a source with probabilities $P(\mathrm{A}) = 1/2$, $P(\mathrm{B}) = 1/4$, $P(\mathrm{C}) = 1/8$, $P(\mathrm{D}) = 1/8$.

The entropy is:

$$H = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{4}\log_2\frac{1}{4} - \frac{1}{8}\log_2\frac{1}{8} - \frac{1}{8}\log_2\frac{1}{8} = \frac{1}{2} + \frac{1}{2} + \frac{3}{8} + \frac{3}{8} = 1.75 \text{ bits}.$$

The optimal codeword lengths are $-\log_2 p_i$: for A, this is 1 bit; for B, 2 bits; for C and D, 3 bits each. These are all integers! We can achieve:

- A → 0 (1 bit)

- B → 10 (2 bits)
- C → 110 (3 bits)
- D → 111 (3 bits)

Expected length: $\frac{1}{2}(1) + \frac{1}{4}(2) + \frac{1}{8}(3) + \frac{1}{8}(3) = 0.5 + 0.5 + 0.375 + 0.375 = 1.75$ bits.

We achieved exactly the entropy! This happens when all probabilities are powers of 2.

When Probabilities Do Not Cooperate

Now consider a biased coin with $P(\mathrm{H}) = 0.9$ and $P(\mathrm{T}) = 0.1$.

The entropy is:

$$H = -0.9 \log_2(0.9) - 0.1 \log_2(0.1) \approx 0.137 + 0.332 = 0.469 \text{ bits.}$$

The optimal lengths would be $-\log_2(0.9) \approx 0.152$ bits for H and $-\log_2(0.1) \approx 3.32$ bits for T. These are not integers.

For a very biased source, most of the "information" comes from the rare event. But symbol-by-symbol coding cannot exploit this efficiently.

The best we can do with a symbol-by-symbol prefix code: H → 0 (1 bit), T → 1 (1 bit). Expected length: 1 bit per symbol.

We pay 1 bit per flip, but the entropy is only 0.469 bits. We are wasting more than half our transmission! The problem is that we cannot assign "0.152 bits" to the common outcome H.

This is where block coding helps.

4.3 Block Coding: Overcoming the Integer Barrier

Instead of encoding symbols one at a time, encode entire blocks of n symbols together.

For an i.i.d. source, a block of n symbols has entropy $n \cdot H(X)$. We treat the entire block as a single "super-symbol" from an alphabet of size $|\mathcal{X}|^n$.

Block coding trades the granularity of individual symbols for the smoothness of large numbers. The law of large numbers helps us.

The expected number of bits for the block is at least $n \cdot H(X)$. The rate per original symbol is at least $H(X)$.

But now the "rounding" overhead is shared across n symbols. If we need to round up to the next integer, that extra bit is divided by n when computing the rate. As $n \to \infty$, the rounding overhead vanishes.

This suggests we can approach the entropy bound arbitrarily closely. But showing this rigorously requires a key insight: typical sequences.

4.4 The Library of Babel

Before the formal proof, let me develop a metaphor that will illuminate the entire argument.

Jorge Luis Borges imagined a library containing every possible book of 410 pages. Every arrangement of letters, spaces, and punctuation marks—meaningful text, gibberish, books in every language, books containing every possible truth and every possible falsehood. The total number of books is unimaginably vast: roughly $25^{1{,}312{,}000}$, for a 25-symbol alphabet.

Borges's "Library of Babel" (1941) is a meditation on infinity and meaning. It also illustrates the curse of combinatorics: most possibilities are useless.

The librarians who live in this library face an existential problem. They seek meaningful books among an ocean of noise. The overwhelming majority of books are random character sequences with no meaning in any language.

Now imagine you are such a librarian, tasked with creating a catalog. You need to assign each book a unique call number so it can be retrieved. If you must catalog every possible book, you need about $1{,}312{,}000 \cdot \log_2(25) \approx 6{,}000{,}000$ bits per call number.

But suppose you only care about books written in English. English has structure. Not every letter sequence is English. In fact, the entropy rate of English is estimated at about 1–1.5 bits per character, far below the $\log_2(25) \approx 4.6$ bits that would be needed if every character were equally likely.

The "meaningful" books—those following English statistics—form a tiny subset of all possible books. We might call this the *typical set*: the books that are statistically plausible outputs of the English language source.

How many bits do you need to catalog the typical books? The entropy rate gives the answer. With about $1{,}312{,}000 \times 1.3 \approx 1{,}700{,}000$ bits, you can uniquely identify every typical English book. This is far fewer than the six million bits needed for arbitrary books.

The typical set is exponentially smaller than the full set of possibilities, but it contains almost all the probability mass. This is the key to compression.

This is the source coding theorem in disguise. The typical set is small enough to be indexed efficiently, yet large enough to contain almost everything we will ever encounter.

4.5 Typical Sequences: The Heart of the Matter

Let us make the library metaphor precise.

Definition of Typical Sequences

For an i.i.d. source with entropy $H = H(X)$, a sequence $(x_1, \ldots, x_n)$ is *ϵ-typical* if:

$$2^{-n(H+\epsilon)} \leq P(x_1, \ldots, x_n) \leq 2^{-n(H-\epsilon)}.$$

In words: a typical sequence has probability close to 2^{-nH}.

A typical sequence is one whose probability is "about what you'd expect" from the entropy. Neither surprisingly likely nor surprisingly unlikely.

Why this definition? For i.i.d. sequences:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i).$$

Taking logs:

$$\log_2 P(x_1, \dots, x_n) = \sum_{i=1}^{n} \log_2 P(x_i).$$

Dividing by n:

$$\frac{1}{n} \log_2 P(x_1, \dots, x_n) = \frac{1}{n} \sum_{i=1}^{n} \log_2 P(x_i).$$

The right side is an average of i.i.d. random variables. By the law of large numbers, it converges to:

$$\mathbb{E}\left[\log_2 P(X)\right] = -H.$$

So for large n, the log-probability per symbol concentrates around $-H$, meaning the total probability concentrates around 2^{-nH}.

Properties of the Typical Set

Let $A_\epsilon^{(n)}$ denote the set of ϵ-typical sequences of length n.

Property 1: High probability. For any $\epsilon > 0$ and sufficiently large n:

$$P(A_\epsilon^{(n)}) > 1 - \epsilon.$$

Almost all the probability mass is on typical sequences. Atypical sequences are negligible.

Property 2: Bounded size. The number of typical sequences satisfies:

$$|A_\epsilon^{(n)}| \leq 2^{n(H+\epsilon)}.$$

This follows directly from the definition: each typical sequence has probability at least $2^{-n(H+\epsilon)}$, and probabilities sum to at most 1.

The typical set has at most $2^{n(H+\epsilon)}$ sequences. This is exponentially smaller than the $|\mathcal{X}|^n$ total sequences when the source is not uniform.

Property 3: Lower bound on size. For large enough n:

$$|A_\epsilon^{(n)}| \geq (1-\epsilon) \cdot 2^{n(H-\epsilon)}.$$

The typical set has at least roughly 2^{nH} sequences.

Together, these properties say: there are about 2^{nH} typical sequences, they account for almost all the probability, and each has probability about 2^{-nH}.

4.6 The Impossibility Proof: Why We Cannot Beat Entropy

Now we can prove that compression below entropy is impossible.

The converse theorem: no code can achieve rate below entropy. This is the "you can't do better than the limit" half of the theorem.

Suppose we try to encode n-symbol sequences using a code with rate $R < H - \delta$ for some $\delta > 0$. The expected codeword length is at most $n(H - \delta)$.

For a uniquely decodable code, each sequence must map to a distinct binary string. If the expected length is $n(H - \delta)$, then there can be at most roughly $2^{n(H-\delta)}$ codewords that we use.

But the typical set contains about 2^{nH} sequences, all of which have significant probability. The ratio is:

$$\frac{2^{nH}}{2^{n(H-\delta)}} = 2^{n\delta}.$$

For large n, this ratio is astronomical. There are exponentially more typical sequences than codewords. We cannot assign distinct codewords to all of them.

The pigeonhole argument: there are exponentially more typical sequences than short codewords. Most typical sequences cannot have short representations.

This is the pigeonhole argument in its purest form. If you have more pigeons than pigeonholes, some pigeonhole must contain multiple pigeons. If you have more typical sequences than short codewords, some sequences must share codewords or have no codeword at all.

Either way, we cannot decode correctly for all typical sequences. Since typical sequences account for almost all the probability mass, our error probability is bounded away from zero.

The Formal Statement

Theorem (Source Coding Converse): For any sequence of uniquely decodable codes with block length n and rate R_n:

$$\liminf_{n \to \infty} R_n \geq H.$$

No code can achieve rate below entropy in the limit.

Addressing an Objection

You might say: "But I've seen compression below entropy! I compressed a file and the compressed version is smaller than the entropy of individual characters would suggest."

This is true, but it does not contradict the theorem. Real files have structure beyond individual characters. English text has dependencies between letters, words, and phrases. The relevant quantity is the *entropy rate*—the entropy per symbol accounting for all dependencies—not the single-symbol entropy.

ZIP and similar algorithms exploit correlations. The source coding theorem applies to the true entropy rate of the source, which is lower than the single-symbol entropy when there are dependencies.

For a stationary ergodic source:

$$\bar{H} = \lim_{n \to \infty} \frac{1}{n} H\left(X_1, X_2, \ldots, X_n\right).$$

This entropy rate can be much lower than $H\left(X_1\right)$ when there are dependencies. ZIP files exploit these dependencies. But no algorithm can compress below the entropy rate—that is what the theorem says.

4.7 The Achievability Proof: Reaching the Entropy Bound

The more remarkable half of the theorem is that we *can* achieve rates arbitrarily close to entropy. Shannon's proof is constructive: we build a code and show it works.

The Strategy

1. Enumerate all ϵ-typical sequences.
2. Assign each a short binary codeword.
3. Use a fallback encoding for atypical sequences.
4. Show the average rate approaches H.

The achievability proof is constructive. We build an explicit code and analyze its performance.

The Construction

Fix $\epsilon > 0$ and block length n.

There are at most $2^{n(H+\epsilon)}$ typical sequences. We can enumerate them and assign each an index from 1 to $2^{n(H+\epsilon)}$.

Each index can be represented in binary using $\lceil n(H+\epsilon) \rceil$ bits. We add a 1-bit header (say, "0") to indicate this is a typical sequence encoding.

Total bits for typical sequences: at most $n(H+\epsilon)+2$.

For atypical sequences, we use a fallback: a header bit "1" followed by the raw sequence. If the alphabet has size $|\mathcal{X}|$, this takes $1 + n\log_2|\mathcal{X}|$ bits.

Computing the Rate

The expected number of bits is:

$$\begin{aligned} \text{Expected bits} &= P(\text{typical}) \cdot (\text{bits for typical}) + P(\text{atypical}) \cdot (\text{bits for atypical}) \\ &\leq (1)(n(H+\epsilon)+2) + \epsilon(1 + n\log_2|\mathcal{X}|). \end{aligned}$$

The rate is:

$$R = \frac{\text{Expected bits}}{n} \leq H + \epsilon + \frac{2}{n} + \frac{\epsilon(1 + n\log_2|\mathcal{X}|)}{n}.$$

Typical sequences dominate because they have probability close to 1. The rare atypical sequences contribute negligibly to the average.

As $n \to \infty$:

$$R \to H + \epsilon + \epsilon\log_2|\mathcal{X}|.$$

Since ϵ was arbitrary, we can make the rate as close to H as desired by choosing ϵ small enough and n large enough.

The Formal Statement

Theorem (Source Coding Achievability): For any $\epsilon > 0$, there exists a sequence of uniquely decodable codes with rate:

$$R_n \leq H + \epsilon$$

for all sufficiently large n.

Combined with the converse, this gives us Shannon's source coding theorem: the minimum achievable rate is exactly H.

4.8 A Worked Example: Compressing a Biased Coin

Let us see the theorem in action with actual numbers.

The Source

A biased coin with $P(H) = 0.9$ and $P(T) = 0.1$.

Entropy: $H = -0.9 \log_2(0.9) - 0.1 \log_2(0.1) = 0.469$ bits per flip.

Recall that symbol-by-symbol encoding gives 1 bit per flip—more than double the entropy.

This coin is highly predictable—heads happens nine times out of ten. The entropy reflects this: less than half a bit of surprise per flip.

Block Encoding with $n = 10$

Consider encoding blocks of 10 flips.

Total possible sequences: $2^{10} = 1024$.

Typical sequences: Roughly those with 8, 9, or 10 heads (close to the expected 9 heads).

Let us count precisely. A sequence with k heads has probability $0.9^k \cdot 0.1^{10-k}$.

The 56 sequences with 8–10 heads account for probability $0.349 + 0.387 + 0.194 = 0.930$.

Heads	Count	P(each)	Total P
10	1	0.349	0.349
9	10	0.0387	0.387
8	45	0.00430	0.194
7	120	0.000478	0.057

Table 4.1: Sequences of 10 biased coin flips, grouped by number of heads.

If we call these "typical," we can assign each a 6-bit codeword (since $2^6 = 64 > 56$), plus a 1-bit header.

Bits for typical sequences: 7 bits for 10 symbols = 0.7 bits per symbol.

Bits for atypical sequences: 1 (header) + 10 (raw) = 11 bits = 1.1 bits per symbol.

Expected rate: $0.930 \times 0.7 + 0.070 \times 1.1 = 0.651 + 0.077 = 0.728$ bits per symbol.

We have reduced from 1 bit per symbol (symbol-by-symbol) to 0.728 bits per symbol, approaching the entropy of 0.469.

Block size n	Rate (bits/symbol)
1	1.000
10	0.728
100	0.52
1000	0.48
∞	0.469

Table 4.2: Compression rate approaches entropy as block length increases.

Scaling Up

As we increase n:

- $n = 100$: Rate ≈ 0.52 bits per symbol

- $n = 1000$: Rate ≈ 0.48 bits per symbol

- $n \to \infty$: Rate $\to 0.469$ bits per symbol

The approach to entropy is not just theoretical—it is computable. With large enough blocks, we can compress arbitrarily close to the entropy bound.

Each factor of 10 increase in block length roughly halves the gap between our rate and entropy. This is the power of the law of large numbers.

4.9 *Shannon and the Birth of Information Theory*

Let us pause for history. In the summer of 1948, Claude Shannon was 32 years old. He worked at Bell Telephone Laboratories, where the practical problems involved telephone communication. How much information could flow through a wire? How should signals be encoded to resist noise?

Shannon's paper, "A Mathematical Theory of Communication," appeared in the Bell System Technical Journal that July and October. In 55 pages, he defined entropy, proved the source coding theorem, defined channel capacity, and proved the noisy channel coding theorem. He invented a field.

Shannon's 1948 paper is one of the most influential scientific works of the twentieth century. It created information theory essentially from scratch.

What Shannon did was audacious. He separated communication into layers. The *meaning* of a message—what it's about, whether it's true—was declared irrelevant to the engineering problem. What mattered was the statistical structure of the source and the physical properties of the channel.

"The fundamental problem of communication," Shannon wrote, "is that of reproducing at one point either exactly or approximately a message selected at another point."

The paper was not immediately understood. Some engineers thought it was pure theory; some mathematicians thought it lacked rigor. Both were wrong. Shannon had identified the right concepts and proved the right theorems. Finding practical codes that achieved his limits would occupy the next fifty years.

Shannon himself was playful. He built maze-solving robots, juggling machines, and a calculator that worked in Roman numerals. He unicycled through Bell Labs' corridors. He worked on problems because they were interesting—and the most interesting problems often proved the most useful.

There is a lesson here. The most practical result in communication theory—telling engineers exactly how much they can compress—came from asking the most abstract question: what does physics allow?

4.10 What If We Allow Errors? Rate-Distortion Theory

So far we have demanded perfect reconstruction: the decoder must recover the source sequence exactly. But sometimes we can tolerate errors. Lossy compression—JPEG for images, MP3 for audio—deliberately introduces distortion to achieve smaller files.

How does the fundamental limit change when we allow errors?

Measuring Distortion

We need a way to measure how bad our errors are. A *distortion function* $d(x, \hat{x})$ quantifies the "cost" of reproducing symbol x as $\hat{x}$.

The distortion function encodes what kind of errors we care about. Different applications have different distortion measures.

Common choices:

- *Hamming distortion*: $d(x, \hat{x}) = 0$ if $x = \hat{x}$, otherwise 1. This counts the number of errors.
- *Squared error*: $d(x, \hat{x}) = (x - \hat{x})^2$. This is natural for continuous signals.

The *average distortion* of a code is the expected distortion per symbol:

$$D = \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}\left[d(X_i, \hat{X}_i)\right].$$

The Rate-Distortion Function

For a given maximum average distortion D, define:

$$R(D) = \min_{\substack{P(\hat{X}|X) \\ \mathbb{E}[d(X,\hat{X})] \le D}} I\left(X; \hat{X}\right).$$

The minimum is over all conditional distributions—all possible ways to reconstruct X as $\hat{X}$—subject to the distortion constraint.

$R(D)$ is the minimum rate needed to achieve distortion D. It trades off compression against quality.

The rate-distortion theorem (which we state without proof) says: $R(D)$ is the minimum achievable rate for reproducing the source with average distortion at most D.

Example: Binary Source with Hamming Distortion

Consider a Bernoulli(1/2) source (fair coin) with Hamming distortion.

At $D = 0$: We demand perfect reconstruction. $R(0) = H(X) = 1$ bit.

At $D = 1/2$: We are allowed to be wrong half the time. We can just output a random guess, ignoring the input entirely. $R(1/2) = 0$ bits.

For intermediate D, the rate-distortion function is:

$$R(D) = 1 - h(D) = 1 + D \log_2 D + (1 - D) \log_2(1 - D).$$

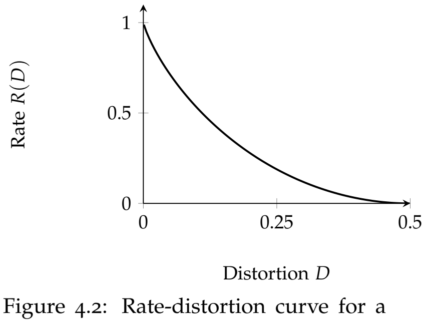


Figure 4.2: Rate-distortion curve for a binary symmetric source. At $D = 0$ we need 1 bit; at $D = 0.5$ we need nothing.

This curve captures a fundamental tradeoff. If you want perfect reproduction, you pay the full entropy. If you can tolerate being wrong some fraction of the time, you pay less. But there is still a floor: even allowing distortion, you cannot escape the laws of information.

4.11 What Does the Theorem Mean?

Let us step back and reflect on what we have proved.

Entropy as a Physical Quantity

The source coding theorem transforms entropy from a mathematical abstraction into a physical reality. Before Shannon, entropy was a formula: $-\sum p \log p$. After Shannon, entropy is the answer to a question you can ask about real data: how many bits do I need?

Imagine explaining this to a 19th-century telegraph operator. "There is a number—computed from the statistics of your messages—that tells you exactly how efficient your compression can be. Not approximately. Exactly." This would seem like magic, or perhaps like numerology. Shannon proved it is neither; it is a theorem.

This is what makes information theory a physical science. Entropy answers a concrete question about the world, not just a mathematical exercise.

The Inevitability of Typical Sequences

The theorem rests on a deep fact: as sequences grow long, almost all the probability concentrates on a small set of typical sequences. This is not a peculiarity of our definitions but a consequence of the law of large numbers.

Consider what this means. For any source with any distribution, long sequences sort themselves into typical and atypical. The typical sequences form a tiny fraction of all possible sequences, but they contain almost all the probability mass. And there are 2^{nH} of them—exactly the number that nH bits can index.

The compression limit is not arbitrary. It reflects the statistical regularity of the world. A source with structure has typical patterns; a source without structure is already incompressible.

The typical set is where probability concentrates. Everything else is negligible. This is why compression is possible and why entropy sets the limit.

Compression and Prediction

There is a deep connection, which we will explore more in Chapter 12: the ability to compress is the ability to predict. If you can predict the next symbol, you don't need to transmit it—you just confirm or deny the prediction.

Consider the extreme case. If a source always produces the same symbol, the entropy is zero. You need not transmit anything; the receiver already knows what's coming. At the other extreme, if all symbols are equally likely and independent, the entropy is maximal; every symbol is a complete surprise and must be transmitted in full.

Entropy measures unpredictability. The source coding theorem says you need exactly as many bits as there is genuine surprise. No more, no less.

Prediction and compression are two faces of the same coin. Good predictions mean low entropy mean high compressibility.

The Unreasonable Effectiveness

Why should there be a single number that answers such a complex question? The theorem could have said: "the minimum rate depends on your encoding scheme, your computational resources, your cleverness." Instead, it says there is a universal limit depending only on the source.

This is a remarkable simplification. Nature could have been more complicated. That it isn't—that entropy alone determines compressibility—is a gift.

4.12 From Theory to Practice

We have proved that entropy is the fundamental limit on compression. Not "a" limit—"the" limit. Given a source, we know exactly how much it can be compressed.

But knowing the limit exists is not the same as achieving it. The proof of achievability is elegant but impractical. It requires knowing the source statistics exactly. It requires exponentially long block lengths. It says nothing about computational efficiency.

This is the gap between existence and construction. Shannon's theorem guarantees that codes approaching entropy exist. But which codes? How do we build them? How fast can we encode and decode?

Shannon proved that good codes exist. Finding them efficiently would occupy the next fifty years.

In the next chapter, we turn from theory to practice. We will develop Huffman codes—optimal among symbol-by-symbol codes. We will develop arithmetic coding, which approaches entropy more closely by avoiding the integer-bit constraint. We will develop the Lempel-Ziv family, algorithms that adapt to unknown sources.

These practical algorithms are guided by Shannon's theorem. It tells us what to aim for, and it warns us when we are wasting effort trying to beat the impossible. The theory constrains the engineering; the engineering vindicates the theory.

Shannon showed that the universe has a speed limit for compression. Now we must learn to drive near that limit.

5
The Art of Compression

Shannon proved that entropy is the compression limit. His proof is beautiful: identify the typical sequences, assign them short codewords, done. The typical set contains about 2^{nH} sequences and accounts for almost all the probability mass. Index them with nH bits each, and you achieve the limit.

But imagine trying to actually *use* this proof. To encode a sequence, you would first need to check whether it belongs to the typical set. This requires knowing the exact probability of the sequence, which means multiplying together n probabilities—and n might be millions. You would need to store a codebook assigning binary strings to each typical sequence. But there are 2^{nH} of them; for English text at 1.3 bits per character, a codebook for 1000-character blocks would have 2^{1300} entries. The codebook itself would be larger than all the atoms in the observable universe.

Shannon's proof shows that good codes *exist*. It does not tell us how to *find* them efficiently. This gap between existence and construction runs throughout mathematics.

Shannon's theorem is like knowing that somewhere in a vast library there exists a book containing exactly the story you want to tell. The theorem guarantees the book exists. But you still need to write it.

This is the gap between proof and practice, between mathematics and engineering. In this chapter, we cross that gap. We will develop three practical compression algorithms, each embodying a different insight about what compression means.

The first is *Huffman coding*: given the symbol probabilities, construct the optimal code one symbol at a time. The second is *arithmetic coding*: escape the constraint of integer-bit codewords by encoding entire messages as intervals on the number line. The third is the *Lempel-Ziv* family: learn the source statistics on the fly, achieving optimal compression without knowing anything in advance.

By the end, we will see how these building blocks combine in systems you use every day—gzip, PNG, JPEG—and we will be ready to confront the problem that compression alone cannot solve: noise.

5.1 Huffman Coding: The Optimal Symbol-by-Symbol Code

Let us begin with the simplest approach: assign each symbol its own codeword.

David Huffman was a graduate student when he invented his algorithm in 1952. The story goes that his professor offered a choice: take the final exam, or solve an open problem. Huffman chose the problem.

Recall from Chapter 4 that for a prefix code with codeword lengths $\ell_1, \ldots, \ell_m$ assigned to symbols with probabilities $p_1, \ldots, p_m$, the expected length is $L = \sum_i p_i \ell_i$. We proved that $L \geq H$, with equality when $\ell_i = -\log_2 p_i$. But codeword lengths must be integers. If the optimal lengths are not integers, we cannot achieve entropy exactly.

The question becomes: given the integer constraint, what is the best we can do?

The Algorithm

Huffman's insight was to build the code tree from the bottom up, always combining the two least probable symbols first.

Given symbols $\{a_1, \ldots, a_m\}$ with probabilities $\{p_1, \ldots, p_m\}$:

1. Create a leaf node for each symbol, labeled with its probability.
2. Find the two nodes with the smallest probabilities.
3. Create a new internal node as their parent, labeled with the sum of their probabilities.
4. Remove the two children from consideration; add the parent.
5. Repeat until only one node remains—the root.
6. Assign codewords by tracing paths from root to leaves: left branch means 0, right branch means 1.

The greedy strategy—always combining the smallest—turns out to be optimal. This is not obvious. Many greedy algorithms fail to find optimal solutions.

This is a greedy algorithm: at each step, make the locally best choice without looking ahead. Such algorithms often fail to find global optima. But for Huffman coding, the greedy choice is provably optimal.

A Worked Example

Let us build a Huffman code for a five-symbol source with probabilities:

$$P(\mathrm{A}) = 0.40, \quad P(\mathrm{B}) = 0.20, \quad P(\mathrm{C}) = 0.20,$$
$$P(\mathrm{D}) = 0.10, \quad P(\mathrm{E}) = 0.10.$$

First, compute the entropy:

$$\begin{aligned} H &= -0.4\log_2(0.4) - 0.2\log_2(0.2) - 0.2\log_2(0.2) - 0.1\log_2(0.1) - 0.1\log_2(0.1) \\ &= 0.529 + 0.464 + 0.464 + 0.332 + 0.332 = 2.122 \text{ bits.} \end{aligned}$$

Now build the tree:

Step 1: The two smallest are D and E, each with probability 0.10. Combine them into a node with probability 0.20. Call it (DE).

Step 2: The remaining nodes have probabilities: A (0.40), B (0.20), C (0.20), (DE) (0.20). The two smallest are any pair among B, C, (DE). Let us combine B and (DE) into (B,DE) with probability 0.40.

Step 3: Remaining: A (0.40), C (0.20), (B,DE) (0.40). Combine the two smallest: C and one of the 0.40 nodes. Combine C with (B,DE) into (C,B,DE) with probability 0.60.

Step 4: Remaining: A (0.40), (C,B,DE) (0.60). Combine into the root with probability 1.00.

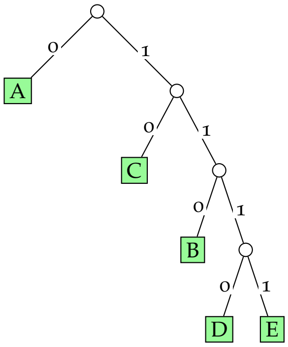


Figure 5.1: Huffman tree for the five-symbol source. The most probable symbol (A) gets the shortest codeword.

Reading off the codewords by tracing paths from root to leaves:

A → 0	(1 bit)
C → 10	(2 bits)
B → 110	(3 bits)
D → 1110	(4 bits)
E → 1111	(4 bits)

The expected codeword length is:

$$L = 0.40(1) + 0.20(2) + 0.20(3) + 0.10(4) + 0.10(4) = 0.4 + 0.4 + 0.6 + 0.4 + 0.4 = 2.2 \text{ bits.}$$

We aimed for 2.122 bits (the entropy) and achieved 2.2 bits. The overhead is 0.078 bits per symbol—about 3.7% above the theoretical limit. Not bad for such a simple algorithm.

Why Huffman Is Optimal

You might ask: "Could there be a cleverer construction that beats Huffman?" No. Among all prefix codes, Huffman achieves the minimum expected length. Here is why.

The optimality proof uses an exchange argument: if Huffman were not optimal, we could swap codewords to make it better, leading to a contradiction.

Consider any optimal prefix code. It must have these properties:

Property 1: The two least probable symbols have the longest codewords. If not, we could swap a short codeword from a rare symbol to a common symbol and reduce the expected length.

Property 2: The two longest codewords have the same length and differ only in the last bit. If not, we could shorten one of them without violating the prefix property, reducing the expected length.

Property 2 means the two least probable symbols are siblings in the code tree—children of the same parent node.

Now here is the key insight. Replace those two symbols with a single "super-symbol" whose probability is the sum of theirs. This reduces the problem by one symbol. By induction, Huffman's greedy construction—always combining the two smallest—builds an optimal tree.

The Gap from Entropy

Huffman is optimal among prefix codes, but it does not always achieve entropy. The gap comes from the integer constraint on codeword lengths.

For any source, the Huffman code satisfies:

$$H \leq L_{\text{Huffman}} < H + 1.$$

The bound $H \leq L < H + 1$ holds for any source. When one symbol has very high probability, the gap can approach 1 bit.

The lower bound is the source coding theorem. The upper bound follows from the Kraft inequality: we can always find a prefix code with lengths $\lceil -\log_2 p_i \rceil$, which exceeds the optimal length $-\log_2 p_i$ by at most 1.

When is the gap large? Consider a highly biased source: $P(\text{A}) = 0.99$, $P(\text{B}) = 0.01$. The entropy is:

$$H = -0.99 \log_2(0.99) - 0.01 \log_2(0.01) = 0.014 + 0.066 = 0.081 \text{ bits.}$$

The optimal codeword lengths would be $-\log_2(0.99) = 0.014$ bits for A and $-\log_2(0.01) = 6.64$ bits for B. But we cannot assign 0.014 bits to any symbol. The shortest possible codeword is 1 bit.

Huffman assigns: A → 0, B → 1. Expected length: 1 bit per symbol. We are paying 1 bit when the entropy is only 0.081 bits—more than 12 times the theoretical limit!

This is where Huffman fails. When one symbol dominates, the integer constraint bites hard.

5.2 Arithmetic Coding: Escaping the Integer Trap

Huffman encodes symbols one at a time, paying at least 1 bit for each symbol, even if the symbol carries almost no information. Can we do better?

The insight behind arithmetic coding is radical: instead of encoding symbols individually, encode the *entire message* as a single number.

The Core Idea

Think of the unit interval $[0, 1)$ as a number line. We will assign each possible message to a sub-interval. The width of the sub-interval equals the message's probability. To encode, we specify any number within the message's interval. To decode, we find which interval contains that number.

Arithmetic coding represents messages as intervals on $[0, 1)$. The more probable the message, the wider its interval. The number of bits needed to specify a point in the interval equals the message's information content.

Why does this work? The number of bits needed to specify a number in an interval of width w is approximately $-\log_2 w$. If we assign intervals proportional to message probabilities, then a message with probability p gets an interval of width p, requiring $-\log_2 p$ bits—exactly the self-information of the message.

No integer constraint. No rounding. The bits needed precisely match the information content.

The Encoding Algorithm

Let me show you how it works with a concrete example.

Suppose our alphabet is $\{A, B, C\}$ with probabilities $P(A) = 0.5$, $P(B) = 0.3$, $P(C) = 0.2$.

We partition $[0, 1)$ according to these probabilities:

- A occupies $[0, 0.5)$
- B occupies $[0.5, 0.8)$
- C occupies $[0.8, 1.0)$

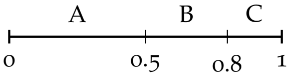


Figure 5.2: Initial partition of $[0, 1)$ according to symbol probabilities.

Now let us encode the message "BAC."

Start: Our interval is $[0, 1)$.

First symbol (B): B occupies $[0.5, 0.8)$ of the current interval. Our new interval is $[0.5, 0.8)$.

Second symbol (A): Within $[0.5, 0.8)$, we subdivide proportionally. The width is $0.8 - 0.5 = 0.3$.

- A: $[0.5, 0.5 + 0.5 \times 0.3) = [0.5, 0.65)$
- B: $[0.65, 0.65 + 0.3 \times 0.3) = [0.65, 0.74)$
- C: $[0.74, 0.8)$

Since the second symbol is A, our new interval is $[0.5, 0.65)$.

Third symbol (C): Within $[0.5, 0.65)$, width $= 0.15$.

- A: $[0.5, 0.575)$
- B: $[0.575, 0.62)$
- C: $[0.62, 0.65)$

Since the third symbol is C, our final interval is $[0.62, 0.65)$.

The final interval $[0.62, 0.65)$ has width 0.03.

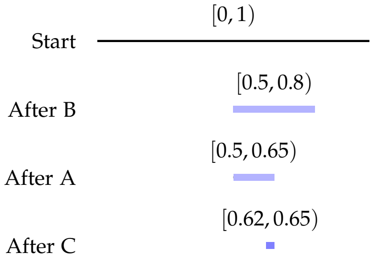


Figure 5.3: The interval narrows with each symbol. After "BAC," we have $[0.62, 0.65)$.

To encode, we can output any number in this interval. A convenient choice is 0.625, which in binary is 0.101. We need about 5 bits to specify this with sufficient precision, plus some framing to indicate where the message ends.

Let us check the information content of "BAC":

$$-\log_2 P(\text{BAC}) = -\log_2(0.3 \times 0.5 \times 0.2) = -\log_2(0.03) = 5.06 \text{ bits.}$$

We needed about 5 bits. The information content is 5.06 bits. Arithmetic coding achieves essentially the theoretical limit.

Decoding

Given the encoded number (say, 0.625) and knowledge of the probability distribution, decoding reverses the process:

1. Find which symbol's interval contains 0.625. Initial partition: A is $[0, 0.5)$, B is $[0.5, 0.8)$, C is $[0.8, 1)$. Since $0.625 \in [0.5, 0.8)$, the first symbol is B.

2. Rescale: map $[0.5, 0.8)$ back to $[0, 1)$. The value 0.625 becomes $(0.625 - 0.5)/(0.8 - 0.5) = 0.125/0.3 = 0.417$.

3. Find which symbol's interval contains 0.417. Since $0.417 \in [0, 0.5)$, the second symbol is A.

4. Rescale: $0.417/0.5 = 0.833$.

5. Find which symbol's interval contains 0.833. Since $0.833 \in [0.8, 1)$, the third symbol is C.

We have recovered "BAC."

Encoder and decoder must use identical probability models. Any mismatch—even in the tenth decimal place—can cause cascading errors.

Why Arithmetic Coding Approaches Entropy

After encoding a message $x_1, x_2, \ldots, x_n$, the final interval has width:

$$\text{width} = P(x_1) \cdot P(x_2) \cdots P(x_n) = P(x_1, \ldots, x_n).$$

The number of bits needed to specify a number in this interval is:

$$\text{bits} \approx -\log_2(\text{width}) = -\log_2 P(x_1, \ldots, x_n) = \sum_{i=1}^{n} (-\log_2 P(x_i)).$$

This is exactly the sum of self-informations—the total information content of the message.

For an i.i.d. source, by the law of large numbers:

$$\frac{1}{n} \sum_{i=1}^{n} (-\log_2 P(X_i)) \xrightarrow{n \to \infty} \mathbb{E}\left[-\log_2 P(X)\right] = H.$$

The rate converges to entropy. The overhead—a few bits for framing and termination—becomes negligible as messages grow long.

Huffman versus Arithmetic: A Comparison

Let us return to the troublesome 99%/1% source.

With Huffman: 1 bit per symbol, but entropy is 0.081 bits. Overhead: 12×.

With arithmetic coding on a message of length n:

- Total bits $\approx n \times 0.081 +$ (framing overhead)
- Rate per symbol $\to 0.081$ as $n \to \infty$

For a 1000-symbol message from the 99%/1% source:

Arithmetic coding shines when distributions are highly skewed. For more balanced sources, Huffman may be preferred for its simplicity and speed.

- Huffman: 1000 bits
- Arithmetic: approximately 81 bits plus overhead, say 85-90 bits total

Arithmetic coding achieves more than 10× better compression on this source.

The tradeoff: arithmetic coding requires more computation. Each symbol involves multiplication and comparison of potentially long numbers. Huffman just does table lookups. For many applications, Huffman's simplicity wins. But when compression really matters, arithmetic coding (or its practical cousin, range coding) is the tool.

5.3 *A Metaphor: Addressing the Library*

Before we continue, let me develop a metaphor that illuminates what these algorithms are doing.

Think of compression as addressing. Huffman gives each neighborhood a name. Arithmetic coding gives GPS coordinates. Lempel-Ziv creates personalized landmarks.

Imagine you need to address houses in a city. One approach: give each neighborhood a name of fixed length. "Downtown" is short because it is large and frequently referenced. "Industrial District North Subsection 7" is long because it is small and rarely needed. This is Huffman coding: assign names according to frequency, but each name is a discrete unit.

A different approach: GPS coordinates. Latitude and longitude pinpoint any location. The precision you need depends on how small the target is. To locate a large park, two decimal places suffice: 40.78, -73.97. To locate a specific mailbox, you might need five: 40.78123, -73.96847. This is arithmetic coding: the message selects a sub-interval, and you specify a point in that interval with just enough precision.

A third approach: create landmarks as you go. "The red house by the bakery on Main Street." The first time you visit, you note the landmark. Later, you reference it: "three blocks past the red house, turn left." This is Lempel-Ziv: build a dictionary of patterns you have seen, and reference them when they recur.

Each approach has virtues. The first is simple and fast. The second is optimally precise. The third adapts to whatever patterns the data contains.

We turn now to the third approach.

5.4 *Lempel-Ziv: Learning the Source*

Huffman and arithmetic coding require knowing the symbol probabilities. But where do these probabilities come from?

For English text, we could estimate letter frequencies from a large corpus. But what if we are compressing a new language? Or DNA sequences? Or software binaries? We would need different probability models for each—and the models themselves take space to transmit.

The chicken-and-egg problem: to compress optimally, we need the source statistics. To learn the statistics, we need to read the data. But reading the data is what we are trying to compress!

There is a more fundamental issue. Real data has structure beyond individual symbols. English text is not just biased toward "e" and away from "z"; it has words, phrases, repeated patterns. The string "the" appears constantly. How do we exploit this?

In 1977 and 1978, Jacob Ziv and Abraham Lempel published two papers that changed everything. Their algorithms—now called LZ77 and LZ78—learn the source statistics on the fly. They achieve optimal compression without knowing anything about the source in advance.

This is called *universal compression*. It is one of the most remarkable results in the field.

The Key Insight: Patterns Repeat

If a source has low entropy, it must have structure. Structure means patterns. Patterns repeat.

Consider English text. The word "the" appears about 7% of the time in typical prose. The digraph "th" is even more common. The phrase "in the" recurs constantly. If we can recognize these repetitions and reference earlier occurrences instead of spelling them out each time, we save bits.

This is the Lempel-Ziv insight: build a dictionary of patterns as you read; when you see a pattern again, just point to the dictionary entry.

Lempel-Ziv algorithms do not estimate probabilities. They just look for repetitions. Yet they achieve optimal compression—a remarkable fact.

LZ77: The Sliding Window

The first Lempel-Ziv algorithm maintains a "sliding window" of recently seen symbols. To encode the current position, find the longest match in the window and encode it as a reference.

The encoding for each step is a triple: (distance back to match, length of match, next symbol).

Example: Encode "ABRACADABRA."

We process left to right, maintaining a window of what we have already seen.

- Position 0: No history yet. Output $(0, 0, A)$—no match, next symbol is A.
- Position 1: Window contains "A." The symbol B does not match. Output $(0, 0, B)$.
- Position 2: Window contains "AB." The symbol R does not match. Output $(0, 0, R)$.
- Position 3: Window contains "ABR." The symbol A matches position

0. Match length 1, next symbol is C. Output $(3, 1, C)$—go back 3 positions, copy 1 symbol, then output C.

- Position 5: Window contains "ABRAC." The symbol A matches position 0 or position 3. Match length 1, next symbol is D. Output $(2, 1, D)$ or $(5, 1, D)$.
- Position 7: Window contains "ABRACAD." Starting from A, we can match "ABRA" from the beginning—4 symbols! Output $(7, 4, -)$ or similar, indicating the end.

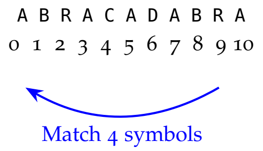


Figure 5.4: LZ77 finding a match: "ABRA" at position 7 matches "ABRA" at position 0.

Instead of transmitting 11 symbols, we transmit a smaller encoded form. The savings grow dramatically with longer texts, because longer matches become possible.

LZ78: The Explicit Dictionary

A year later, Ziv and Lempel published a variant that maintains an explicit dictionary rather than a sliding window.

The algorithm parses the input into phrases, where each phrase extends a previous phrase by exactly one symbol. Each phrase is encoded as (dictionary index, new symbol) and added to the dictionary.

Example: Encode "AABABCABABC."

- See "A." Not in dictionary. Output $(0, A)$. Add "A" as entry 1.
- See "A" again, then "B." Entry 1 is "A," so we have "A" extended by "B" = "AB." Output $(1, B)$. Add "AB" as entry 2.
- See "A" again, then "B," then "C." Entry 2 is "AB," so we have "AB" extended by "C" = "ABC." Output $(2, C)$. Add "ABC" as entry 3.
- And so on.

The dictionary grows to capture recurring patterns. Later occurrences of "ABC" would reference entry 3 directly.

LZW: The Practical Refinement

In 1984, Terry Welch refined LZ78 into what became the most widely used variant: LZW.

The key improvement: initialize the dictionary with all single symbols, so we never need to transmit raw symbols—only dictionary references. This simplifies the encoding and slightly improves compression.

LZW became ubiquitous. The GIF image format used it. The Unix `compress` command used it. And then Unisys, which held the patent, began demanding licensing fees. The resulting controversy helped drive the development of PNG and the broader move toward patent-free formats.

LZW was used in GIF images and the Unix `compress` utility. Patent disputes eventually pushed the web toward PNG, which uses LZ77 variants instead.

The Universality Theorem

Here is the remarkable fact about Lempel-Ziv algorithms:

Theorem (Ziv-Lempel): For any stationary ergodic source, the compression rate of LZ77 (and LZ78) converges to the entropy rate as the input length goes to infinity.

This is universal compression. The algorithm knows nothing about the source—no probability estimates, no model, no assumptions beyond stationarity and ergodicity. Yet it achieves entropy.

Without knowing *anything* about the source statistics, Lempel-Ziv achieves the information-theoretic optimum. It learns the source from the data itself.

Why does this work? The intuition is beautiful:

1. Low-entropy sources have predictable patterns.
2. Predictable patterns must repeat (otherwise they would not be predictable).
3. Repetitions get captured in the dictionary (or found in the sliding window).
4. Referencing dictionary entries is cheap.
5. Therefore, low-entropy sources compress well.

The algorithm does not compute entropy. It does not estimate probabilities. It just looks for repetitions. And yet this is sufficient to achieve the fundamental limit.

5.5 Compression in Practice: What You Use Every Day

Let us connect these algorithms to systems you encounter constantly, often without realizing it.

gzip and DEFLATE

When you download a file from the web, it is often compressed with gzip. When you unzip a ZIP archive, you are using the same algorithm. When your browser loads a PNG image, the same again.

The underlying algorithm is DEFLATE, which combines LZ77 with Huffman coding:

1. Apply LZ77 to find repeated patterns. The output is a stream of literal bytes (symbols that did not match) and (distance, length) pairs (references to earlier occurrences).
2. Apply Huffman coding to this stream. The Huffman code is optimized for the specific data being compressed.

DEFLATE combines the pattern-finding power of LZ77 with the probability-based efficiency of Huffman coding. Each compensates for the other's weaknesses.

Why the combination? LZ77 captures long-range repetitions—entire words and phrases that recur. Huffman squeezes out the remaining redundancy in the references themselves (some distances and lengths are more common than others).

Typical compression ratios:

- English text: 3:1 to 4:1
- Source code: 4:1 to 5:1
- Already-compressed data (JPEG, MP3): essentially 1:1—no further compression possible

bzip2 and the Burrows-Wheeler Transform

A different approach: instead of finding repetitions directly, *transform* the data to group similar characters together, then compress the transformed data.

The Burrows-Wheeler Transform (BWT) is a remarkable reversible permutation. It rearranges the input so that characters from similar contexts cluster together. After BWT, a text that had scattered occurrences of "t" followed by "h" now has many "t"s in a row.

After BWT, apply move-to-front encoding (recently seen symbols get small numbers) and then Huffman coding. The result is bzip2, which typically beats gzip by 10-20% on text—at the cost of slower compression and decompression.

PNG: Lossless Image Compression

Images have structure that text lacks: two-dimensional correlation. A blue pixel is likely to be next to another blue pixel.

PNG exploits this with prediction. Before compressing, it predicts each pixel from its neighbors (above, left, and above-left). It then encodes the *prediction error*—the difference between the actual pixel and the predicted one.

Prediction reduces redundancy before compression. If the predictor is good, the errors are small numbers clustered around zero—easy to compress.

If neighboring pixels are similar, the prediction errors are small. Small numbers concentrated around zero compress far better than the original pixel values.

After prediction, DEFLATE compresses the error stream.

Typical compression ratios:

- Photographs: 2:1 (lots of noise defeats prediction)
- Screenshots with large flat regions: 10:1 or better
- Line art and diagrams: 20:1 or more

JPEG: Crossing into Lossy Territory

So far, everything has been *lossless*—the original can be perfectly reconstructed. JPEG crosses a boundary: it discards information that humans do not notice.

JPEG trades fidelity for compression. The "quality" setting controls how much information is discarded. Low quality means small files but visible artifacts.

The JPEG pipeline:

1. Convert RGB to YCbCr (luminance plus two chrominance channels). The human eye is more sensitive to luminance than to color, so we can treat them differently.
2. Subsample the chrominance channels—keep only every second pixel in each direction. This alone gives 2:1 compression with minimal visible effect.
3. Apply the Discrete Cosine Transform (DCT) to convert from spatial to frequency domain. The image becomes a set of frequency coefficients.
4. Quantize the coefficients: divide by a quantization matrix and round. High-frequency components (fine detail) are quantized more coarsely, often to zero. This is where information is lost.
5. Entropy code the remaining coefficients with Huffman coding.

The quantization step implements rate-distortion theory in practice. The quantization matrix controls the tradeoff: coarser quantization means smaller files but more visible artifacts.

Typical compression: 10:1 to 20:1 with acceptable quality for photographs. At lower quality settings, you see the characteristic JPEG artifacts—blocky regions and ringing around sharp edges.

Video: H.264 and Beyond

Video compression achieves ratios of 100:1 or more by exploiting temporal redundancy: adjacent frames are nearly identical.

Modern video codecs use motion compensation: "This block moved 3 pixels right since the last frame." Only the motion vectors and residual errors need to be encoded.

H.264 (and its successor H.265/HEVC) use:

- Spatial prediction (like JPEG)
- Temporal prediction: predict each block from previous frames
- Motion compensation: when an object moves, encode the motion vector rather than the pixels
- Arithmetic coding (CABAC—Context-Adaptive Binary Arithmetic Coding), which adapts its probability model based on context

A two-hour movie might be 8 GB uncompressed (at 1080p) but compress to 2-4 GB with excellent quality. The savings come largely from exploiting the fact that most frames are similar to their neighbors.

5.6 Actual Numbers: Compression Ratios on Real Data

Let us ground the theory in concrete measurements.

Method	Size	Bits/char
None	5.4 MB	8.0
Huffman	3.0 MB	4.5
gzip	1.9 MB	2.8
bzip2	1.4 MB	2.1
xz	1.3 MB	1.9

Table 5.1: Compressing the complete works of Shakespeare (5.4 MB).

Text Compression

The complete works of Shakespeare: 5.4 MB of plain text.

Character Huffman achieves about 4.5 bits per character—close to the single-character entropy of English. But gzip does much better: 2.8 bits per character. It exploits word-level and phrase-level patterns that character-by-character coding misses.

The best general-purpose compressors (xz, using LZMA) reach 1.9 bits per character. The entropy rate of English, accounting for all context, is estimated at 1.0-1.5 bits per character. We are approaching but not quite reaching the limit.

Why the gap? Practical algorithms make approximations. They use finite windows and dictionaries. They do not perfectly model long-range dependencies. There is room for improvement—but not much.

DNA Sequences

DNA uses four symbols: A, C, G, T. If the sequence were random, entropy would be 2 bits per base (since $\log_2 4 = 2$).

DNA is not random—it has genes, regulatory regions, repeated sequences. Specialized compressors exploit this structure.

General-purpose compressors achieve about 1.5 bits per base on human genome data. Specialized genomic compressors, exploiting the structure of DNA (repeated sequences, reverse complements, similar regions across chromosomes), achieve 0.5-1.0 bits per base.

The gap between general and specialized compressors is larger here than for text. DNA has structure that general algorithms do not exploit well.

What Cannot Be Compressed

Truly random data is incompressible. If bytes are independent and uniformly distributed, entropy is 8 bits per byte, and no algorithm can do better.

Claims of "super-compression" that beats entropy are either fraudulent or measuring the wrong thing. The source coding theorem is not negotiable.

Already-compressed files (JPEG, MP3, ZIP) are essentially incompressible. They have already extracted the structure; what remains is nearly random.

Encrypted data is incompressible without the key. Good encryption makes ciphertext indistinguishable from random—that is the point.

You might ask: "But I have seen programs that claim to compress anything! They must violate information theory."

They do not. Such claims are either fraudulent, measuring against an inefficient baseline, or working only on specific data types. The source coding theorem is a mathematical fact. You cannot compress below entropy any more than you can build a perpetual motion machine.

5.7 What Compression Teaches Us

Let us step back and reflect on what we have learned.

Compression Is Understanding

To compress data is to find its patterns, its regularities, its structure. A good compressor must predict what comes next; the better the prediction, the fewer bits needed to correct it.

Scientific laws are compression algorithms for nature. Newton's laws "compress" the trajectories of all falling objects into a few equations.

This is not just a metaphor. Consider: if you know that a sequence consists of alternating 0s and 1s, you can compress it to almost nothing—just "alternating, starting with 0, length 1000." If you know nothing about a sequence, you must transmit it verbatim.

Knowledge enables compression. Compression reveals knowledge.

There is a provocative claim here: perhaps all understanding is compression—finding the shortest description of the regularities in experience. This is the intuition behind the Minimum Description Length principle, which we will explore in Chapter 12.

The Incompressible Core

Every source has an irreducible core: the entropy. This is the genuine unpredictability, the true surprise. No cleverness can compress it further.

For English text, about 1-1.5 bits per character is irreducible—the genuine uncertainty about which word comes next.

For a physical system, the entropy is the logarithm of the number of microstates compatible with the macrostate—the information we would need to specify exactly which microstate the system occupies.

Entropy measures what we cannot predict. It is both the limit of compression and the measure of our ignorance.

Is this entropy "real" or just a reflection of our ignorance? This is a deep question. In quantum mechanics, the answer seems to be that some unpredictability is fundamental—not a failure of knowledge but a feature of nature. We will return to this in Chapter 14.

The Miracle of Universal Compression

Lempel-Ziv algorithms know nothing about the source. They make no assumptions about probabilities. They just look for repetitions. And yet they achieve the information-theoretic optimum.

This is remarkable. The algorithm does not compute entropy. It does not need to. The patterns are there in the data, and finding them is sufficient.

There is a connection here to machine learning. Modern language models (GPT and its successors) are, at their core, sophisticated compression algorithms. They learn to predict the next token—which is exactly what a good compressor does. The hypothesis that better compression implies better understanding of language is taken seriously by researchers.

But we should be careful. Compression is necessary for understanding, but is it sufficient? A lookup table that memorizes everything compresses perfectly but understands nothing. The relationship between compression, prediction, and intelligence is deep and not fully understood.

5.8 *From Compression to Noise*

We have crossed from theory to practice. Given Shannon's limit—entropy—we now know how to approach it:

- Huffman for quick-and-simple symbol-by-symbol coding
- Arithmetic coding for highly skewed distributions
- Lempel-Ziv for unknown sources

These are not academic exercises. They are the engines inside gzip, PNG, H.264, and every compression system you use.

But all of this assumes something we have not questioned: that communication is perfect. We encode a message, transmit the bits, and assume they arrive uncorrupted. Every 0 sent arrives as 0. Every 1 arrives as 1.

This is a fiction. Real channels have noise. Bits flip. Packets drop. Signals fade in and out. The wireless signal from your phone, the scratched surface of a CD, the fluctuating voltage in a wire—all introduce errors.

Compression removes redundancy to minimize bits. Error correction adds redundancy to survive noise. They seem opposite but are two faces of the same coin.

Is reliable communication possible in the presence of noise? Can we send a message and be confident it arrives correctly?

Shannon's second great theorem answers yes—but only if we do not try to communicate too fast. There is a maximum rate, called the *channel capacity*, beyond which reliable communication is impossible. Below capacity, arbitrary reliability can be achieved with clever coding.

This is the subject of the next three chapters. We will define channel capacity, prove Shannon's noisy channel coding theorem, and develop the error-correcting codes that make modern communication possible.

The source coding theorem told us how much we can compress. The noisy channel coding theorem will tell us how fast we can communicate.

Together, they form the foundation of information theory—and the foundation of the digital world.

6
Channel Capacity

Suppose you need to send a message across a room where someone is playing loud music. You shout; your friend hears something—maybe what you said, maybe not. You could repeat yourself: say each word twice, three times, ten times. Eventually your friend will probably get it right. But here is the catch: you have limited time. The concert starts in five minutes. How many messages can you reliably convey before then?

The problem of communication over noisy channels seems to present an impossible tradeoff: more repetition means more reliability but slower communication. Shannon showed this intuition is wrong.

The naive answer seems depressing. Any amount of noise requires some redundancy to overcome. The lower you want your error probability, the more redundancy you need. In the limit of perfect reliability—zero errors, ever—you would need infinite redundancy. Your rate of actual information transfer would drop to zero. If you want perfection, you can communicate nothing at all.

But this cannot be right. Telephone networks work. Satellite links work. The internet works—even when packets get corrupted and bits get flipped. How?

The answer, discovered by Claude Shannon in 1948, is one of the most remarkable results in all of science. There exists a quantity—the *channel capacity*—below which reliable communication is possible and above which it is not. You do not need infinite repetition to achieve arbitrarily small error probability. You just need to stay below capacity.

This is not obvious. In fact, it is so non-obvious that many engineers refused to believe it at first. The natural intuition is that there should be a smooth tradeoff: lower rate means better reliability, but never perfect reliability, always some errors sneaking through. Shannon showed this intuition is completely wrong. Below capacity, you can have arbitrarily small error probability—as close to zero as you like. Above capacity, errors are inevitable no matter how clever you are.

Shannon's result cleaves communication into two regimes: below capacity (reliable communication possible) and above capacity (reliable communication impossible). There is no middle ground.

The capacity is defined in terms of mutual information—the quantity we developed in Chapter 3. Our task now is to understand what this definition means, compute it for specific channels, and see why it represents a genuine physical limit on communication. In the next chapter, we will prove that capacity is achievable. But first we must understand what we are trying to achieve.

6.1 What Is Channel Capacity?

Let us set up the problem precisely. We have a communication channel described by a conditional probability $P(Y|X)$: given input X, the channel produces output Y according to this distribution. The channel is noisy, meaning Y is not a deterministic function of X—randomness creeps in.

We want to send one of M possible messages through n uses of the channel. The encoder takes a message $m \in \{1, 2, \ldots, M\}$ and produces a sequence of channel inputs $x_1, x_2, \ldots, x_n$. The decoder observes the channel outputs $y_1, y_2, \ldots, y_n$ and produces an estimate $\hat{m}$ of which message was sent.

The rate R measures information per channel use. If we send one of M messages in n uses, we are conveying $\log_2 M$ bits of information in n transmissions, so $R = (\log_2 M)/n$ bits per channel use.

The *rate* of communication is

$$R = \frac{\log_2 M}{n} \quad \text{bits per channel use.}$$

The *error probability* is $P_e = P(\hat{m} \neq m)$—the probability that the decoder gets it wrong.

Now we can ask: what rates are achievable? A rate R is *achievable* if there exist encoding and decoding schemes such that, as the block length n grows, the error probability P_e can be made arbitrarily small. Not just small—arbitrarily small. As small as 10^{-6}, or 10^{-100}, or any positive number you name.

Capacity as Maximum Mutual Information

The channel capacity is defined as

$$C = \max_{P(X)} I(X;Y)$$

where the maximum is over all possible input distributions $P(X)$.

Let us unpack this definition carefully. The channel $P(Y|X)$ is given by nature—it is the physics of our communication medium. We cannot change how noise corrupts our signal. But we can choose how we use the channel. Should we favor 0s over 1s? Should our signal be Gaussian or uniform? Should we use some symbols more than others? The input distribution $P(X)$ is the one thing we control.

The capacity represents the best possible performance with the optimal use of the channel. It is a property of the channel itself, not of any particular coding scheme.

The capacity is what we get when we choose optimally.

Why mutual information? Recall from Chapter 3 that $I(X;Y)$ measures how much the output tells us about the input. It quantifies the statistical dependence between what we send and what we receive.

If $I(X;Y) = 0$, the output is statistically independent of the input. The channel is useless—what comes out has nothing to do with what went in. Looking at the output tells you nothing about the message.

If $I(X;Y) = H(X)$, the output determines the input completely. Given the output, there is no remaining uncertainty about what was sent. The channel is perfect.

For noisy channels, we get something in between. The mutual information quantifies exactly how much information flows through.

Why Maximizing $H(Y)$ Is Not Enough

You might think: "The output carries the information. Should we not just maximize $H(Y)$?"

Consider a noiseless channel where $Y = X$. If we choose $P(X = 0) = 1$ (always send 0), then $H(Y) = 0$. If we choose $P(X)$ uniform over $\{0, 1\}$, then $H(Y) = 1$ bit. So yes, maximizing $H(Y)$ gives the uniform input, and that is indeed optimal here.

But for noisy channels, the story is more subtle. What we want is not large $H(Y)$ per se, but large $I(X;Y) = H(Y) - H(Y|X)$. The term $H(Y|X)$ represents the noise—the randomness in the output that is unrelated to the input. We cannot reduce this; it is intrinsic to the channel. What we can do is choose $P(X)$ to maximize $H(Y)$ while keeping $H(Y|X)$ fixed.

For some channels, the capacity-achieving input distribution is not uniform. The optimization problem can be nontrivial.

What the Coding Theorem Will Say

Let me preview what we will prove in Chapter 7. The channel coding theorem has two parts:

Achievability: If $R < C$, there exist codes with error probability $P_e \to 0$ as $n \to \infty$.

Converse: If $R > C$, every code has error probability bounded away from zero.

In other words, capacity is a sharp threshold. Below it, we can do arbitrarily well—not just "pretty well," but as close to perfect as we want. Above it, no amount of cleverness helps.

The coding theorem says capacity is a sharp threshold. This is not an approximation or a rule of thumb. It is a mathematical theorem.

The remarkable thing is that below the threshold, the improvement is not gradual. Going from rate 0.9C to rate 0.8C does not just give you somewhat better reliability. Both rates can achieve arbitrarily small error probability. The difference is in how long your codewords need to be to achieve a given error level, not in what error level is ultimately achievable.

But I am getting ahead of myself. First, let us compute capacity for some specific channels.

6.2 The Binary Symmetric Channel

The simplest interesting noisy channel is the *binary symmetric channel* (BSC) with crossover probability p. The input is a bit: $X \in \{0, 1\}$. The output is also a bit: $Y \in \{0, 1\}$. With probability $1 - p$, the output equals the input. With probability p, the bit gets flipped.

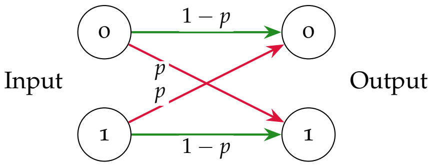


Figure 6.1: The binary symmetric channel with crossover probability p. Each bit is flipped independently with probability p.

Formally:

$$\begin{aligned} P(Y = 0|X = 0) &= 1 - p \\ P(Y = 1|X = 0) &= p \\ P(Y = 0|X = 1) &= p \\ P(Y = 1|X = 1) &= 1 - p \end{aligned}$$

The channel is "symmetric" because it treats 0 and 1 the same way. The probability of an error does not depend on which bit was sent.

Computing the Capacity

Let $q = P(X = 1)$, so $P(X = 0) = 1 - q$. We need to find the value of q that maximizes $I(X; Y)$.

Step 1: Compute $H(Y)$.

What is the probability that $Y = 1$?

$$\begin{aligned} P(Y = 1) &= P(Y = 1|X = 0)P(X = 0) + P(Y = 1|X = 1)P(X = 1) \\ &= p(1 - q) + (1 - p)q \\ &= p - pq + q - pq \\ &= p + q(1 - 2p). \end{aligned}$$

Let us call this quantity $r = p + q(1 - 2p)$. Then $H(Y) = H_b(r)$, where H_b is the binary entropy function:

$$H_b(r) = -r \log_2 r - (1 - r) \log_2(1 - r).$$

Step 2: Compute $H(Y|X)$.

What is the entropy of Y given that we know X?

If $X = 0$: Y is 0 with probability $1 - p$ and 1 with probability p. So $H(Y|X = 0) = H_b(p)$.

If $X = 1$: Y is 0 with probability p and 1 with probability $1 - p$. So $H(Y|X = 1) = H_b(p)$.

Either way, the conditional entropy is $H_b(p)$. This is the entropy of "flip or not flip"—the noise of the channel. It does not depend on what we sent, only on the channel's crossover probability.

The key property of the BSC: the noise entropy $H(Y|X)$ does not depend on the input distribution. The channel flips bits at rate p regardless of what we send.

Therefore:

$$H(Y|X) = P(X = 0)H(Y|X = 0) + P(X = 1)H(Y|X = 1) = (1 - q)H_b(p) + qH_b(p) = H_b(p).$$

Step 3: Compute mutual information.

$$I(X;Y) = H(Y) - H(Y|X) = H_b(r) - H_b(p)$$

where $r = p + q(1-2p)$.

Step 4: Maximize over q.

Since $H_b(p)$ does not depend on q, we just need to maximize $H_b(r)$. The binary entropy function H_b is maximized when its argument is $1/2$. So we want:

$$r = p + q(1-2p) = \frac{1}{2}.$$

Solving for q:

$$q = \frac{1/2 - p}{1-2p} = \frac{1}{2}.$$

The capacity-achieving input distribution is uniform: use 0 and 1 with equal probability.

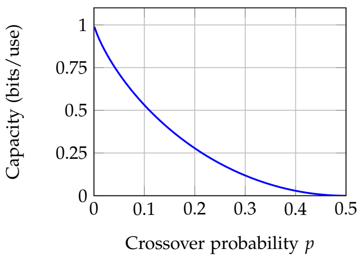


Figure 6.2: Capacity of the BSC as a function of crossover probability p. At $p = 0$, capacity is 1 bit. At $p = 0.5$, capacity is zero.

The BSC Capacity Formula

With $q = 1/2$, we have $r = 1/2$, so $H(Y) = H_b(1/2) = 1$ bit. The capacity is:

$$\boxed{C_{\text{BSC}}(p) = 1 - H_b(p) \text{ bits per channel use.}}$$

Let us compute some actual numbers.

Table 6.1: BSC capacity for various crossover probabilities.

p	$H_b(p)$	$C_{\text{BSC}}(p)$
0	0	1.000
0.01	0.081	0.919
0.05	0.286	0.714
0.10	0.469	0.531
0.20	0.722	0.278
0.30	0.881	0.119
0.50	1.000	0.000

Several observations are worth making.

At $p = 0$ (perfect channel): Capacity is 1 bit per use. Every bit sent is a bit received. There is no noise to overcome.

At $p = 0.5$ (random channel): Capacity is zero. The output is statistically independent of the input—each output bit is equally likely to be 0 or 1, regardless of what was sent. You might as well be shouting into a void.

At $p = 0.1$: One bit in ten gets flipped, on average. Yet capacity is 0.531 bits per channel use—more than half a bit! We can reliably communicate substantial information despite 10% of our bits being corrupted.

At $p = 0.1$, one bit in ten gets flipped, yet we can still reliably communicate at more than half a bit per channel use. The coding theorem will explain how.

You might say, “Half a bit? What does it mean to send half a bit?”

Remember that capacity is about rates over many uses. We cannot send half a bit in one channel use. But over 1000 uses of a BSC with $p = 0.1$, we can reliably send 531 bits of information, even though about 100 of those 1000 transmissions will be corrupted. How is this possible? That is what the coding theorem will tell us. The key is to spread information across many channel uses in a clever way, so that local errors do not destroy the global message.

One more observation: $C(p) = C(1 - p)$. A channel that always flips is just as good as one that never flips—you just need to know which you have. If $p = 0.9$, every bit gets flipped with 90% probability. But that means the output is a noisy version of the *complement* of the input. Decode accordingly, and you have the same capacity as a channel with $p = 0.1$.

6.3 *The Binary Erasure Channel*

Another fundamental channel model is the *binary erasure channel* (BEC) with erasure probability ε. The input is a bit. The output is either that same bit, or a special symbol “?” indicating that the bit was lost.

Formally:

$$
\begin{aligned}
P(Y = 0|X = 0) &= 1 - \varepsilon \\
P(Y = ?|X = 0) &= \varepsilon \\
P(Y = 1|X = 1) &= 1 - \varepsilon \\
P(Y = ?|X = 1) &= \varepsilon
\end{aligned}
$$

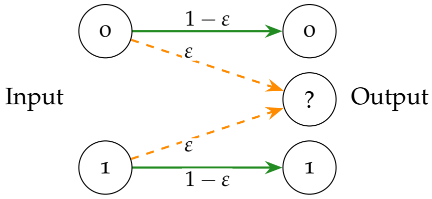


Figure 6.3: The binary erasure channel with erasure probability ε. Bits are either delivered perfectly or replaced with “?”.

The channel either delivers the bit perfectly or confesses that it lost it. It never lies—when you receive a 0 or 1, you know it is correct. The uncertainty is confined to the erasures.

Computing the Capacity

Let $q = P(X = 1)$. We could compute $H(Y)$ and $H(Y|X)$ as before, but there is an easier approach. Let us use the alternative expression $I(X;Y) = H(X) - H(X|Y)$.

We have $H(X) = H_b(q)$.

For $H(X|Y)$, we need to think about what we know about X after observing Y:

- If $Y = 0$: We know with certainty that $X = 0$. So $H(X|Y = 0) = 0$.
- If $Y = 1$: We know with certainty that $X = 1$. So $H(X|Y = 1) = 0$.
- If $Y = ?$: We know nothing new. Our uncertainty about X is still $H_b(q)$.

The output probabilities are:

$$\begin{aligned}P(Y=0) &= (1-\varepsilon)(1-q)\\ P(Y=1) &= (1-\varepsilon)q\\ P(Y=?) &= \varepsilon\end{aligned}$$

Therefore:

$$\begin{aligned}H(X|Y) &= P(Y=0)\cdot 0 + P(Y=1)\cdot 0 + P(Y=?)\cdot H_b(q)\\ &= \varepsilon \cdot H_b(q).\end{aligned}$$

The mutual information is:

$$I(X;Y) = H_b(q) - \varepsilon \cdot H_b(q) = (1-\varepsilon)H_b(q).$$

To maximize over q: $H_b(q)$ is maximized at $q = 1/2$, giving $H_b(1/2) = 1$.

The erasure channel is honest: when it delivers a bit, the bit is correct. When it loses a bit, it tells you. This honesty makes the channel easier to analyze—and easier to achieve capacity on.

The BEC Capacity Formula

$$\boxed{C_{\text{BEC}}(\varepsilon) = 1 - \varepsilon \text{ bits per channel use.}}$$

The capacity of the erasure channel is exactly 1 minus the erasure probability. If half your bits get erased, you can reliably send half a bit per channel use. If 10% get erased, you can send 0.9 bits per channel use.

The formula could not be simpler. And it reveals something important: erasures are much less damaging than errors.

Erasures versus Errors

Compare the two channels at the same "loss rate":

- BSC with $p = 0.1$: $C = 0.531$ bits per use
- BEC with $\varepsilon = 0.1$: $C = 0.9$ bits per use

The erasure channel has nearly twice the capacity! Why?

When the BSC flips a bit, you receive a 0 or a 1, but you do not know whether it is correct. You might think you know the message when you do not. This uncertainty is insidious—it spreads confusion.

When the BEC erases a bit, you receive a "?". You know exactly which bits are missing. You can design your code to recover from known missing pieces. The certainty about what you received—even when that certainty is "I received nothing"—has value.

Erasures tell you what you do not know. Errors let you think you know when you do not. The certainty that comes from knowing where the losses are has value.

This insight matters in practice. Packet networks (like the internet) often use protocols that detect corrupted packets and request retransmission. A detected error becomes an erasure—much easier to handle.

6.4 The Gaussian Channel

The BSC and BEC are discrete channels: inputs and outputs take finitely many values. The *Gaussian channel* is continuous, and it models many physical communication systems: radio, telephone, optical fiber.

The model is simple:

$$Y = X + Z$$

where $Z \sim \mathcal{N}(0, N)$ is Gaussian noise with variance N, independent of the input X.

There is one crucial constraint: the input must have bounded average power. We require

$$\mathbb{E}[X^2] \leq P.$$

The Gaussian channel captures the essence of analog communication. The input is a signal, the noise is thermal fluctuations, and the output is the corrupted signal.

Without this constraint, we could communicate any amount of information by simply increasing the signal strength. The power constraint is the physics: every transmitter has limited power.

Why Gaussian Input Is Optimal

For continuous random variables, we use differential entropy h instead of discrete entropy H. The mutual information is

$$I(X;Y) = h(Y) - h(Y|X).$$

Since $Y = X + Z$ where Z is independent of X:

$$h(Y|X) = h(X+Z|X) = h(Z|X) = h(Z) = \frac{1}{2}\log_2(2\pi e N).$$

The noise entropy $h(Z)$ is fixed—we cannot change the noise. To maximize $I(X;Y)$, we must maximize $h(Y)$.

Now, a fundamental result from information theory: among all distributions with a given variance, the Gaussian distribution has the maximum differential entropy. If $\text{Var}(Y) = \sigma^2$, then $h(Y) \leq \frac{1}{2}\log_2(2\pi e\sigma^2)$, with equality if and only if Y is Gaussian.

What is the variance of Y? Since $Y = X + Z$ with X and Z independent:

$$\text{Var}(Y) = \text{Var}(X) + \text{Var}(Z) = \mathbb{E}[X^2] + N \leq P + N.$$

The Gaussian input maximizes the output entropy, and hence the mutual information. This is a beautiful consequence of the entropy-maximizing property of the Gaussian distribution.

The variance is maximized when $\mathbb{E}[X^2] = P$ (using all available power). And $h(Y)$ is maximized when Y is Gaussian, which happens when X is Gaussian (since the sum of independent Gaussians is Gaussian).

Therefore: the capacity-achieving input is $X \sim \mathcal{N}(0, P)$.

The Capacity Formula

With Gaussian input $X \sim \mathcal{N}(0, P)$:

$$h(Y) = \frac{1}{2}\log_2(2\pi e(P+N))$$
$$h(Y|X) = \frac{1}{2}\log_2(2\pi eN)$$

The capacity is:

$$\begin{aligned} C &= h(Y) - h(Y|X) \\ &= \frac{1}{2}\log_2(2\pi e(P+N)) - \frac{1}{2}\log_2(2\pi eN) \\ &= \frac{1}{2}\log_2\frac{P+N}{N} \end{aligned}$$

$$\boxed{C = \frac{1}{2}\log_2\left(1 + \frac{P}{N}\right) \text{ bits per channel use.}}$$

This is Shannon's famous formula. The ratio SNR $= P/N$ is the signal-to-noise ratio—the fundamental quantity governing analog communication.

Actual Numbers

Let us compute capacities for various signal-to-noise ratios.

SNR (P/N)	SNR (dB)	C (bits/use)	Comment
0.1	-10	0.069	Very noisy
0.5	-3.0	0.292	
1.0	0	0.500	Signal = noise
3.0	4.8	1.000	One bit per use
7.0	8.5	1.500	
15.0	11.8	2.000	Two bits per use
31.0	14.9	2.500	
63.0	18.0	3.000	

Table 6.2: Gaussian channel capacity for various signal-to-noise ratios. SNR in dB is $10\log_{10}(\text{SNR})$.

Several patterns emerge.

At SNR $= 1$ (0 dB): Signal and noise have equal power, and capacity is exactly 0.5 bits per use. This is the breakeven point.

At SNR $= 3$ (about 5 dB): Capacity is 1 bit per use. Getting that first bit requires about 5 dB of SNR.

Doubling the capacity requires roughly quadrupling the SNR. To go from 1 bit/use to 2 bits/use, we need to go from SNR $= 3$ to SNR $= 15$—an increase of a factor of 5, or about 7 dB. There is a law of diminishing returns: the first bit is cheap, but each additional bit costs more power.

The signal-to-noise ratio is measured in decibels (dB): $\text{SNR}_{\text{dB}} = 10\log_{10}(P/N)$. Every 3 dB doubles the power ratio.

At high SNR, the formula simplifies:

$$C \approx \frac{1}{2}\log_2 \frac{P}{N} = \frac{1}{2} \cdot \frac{\text{SNR}_{\text{dB}}}{3.01} \approx \frac{\text{SNR}_{\text{dB}}}{6}.$$

You get about one bit per 3 dB of signal-to-noise ratio in the high-SNR regime.

A Real-World Calculation

Let me make this concrete with a practical example. Consider a cellular data link with:

- Bandwidth: $W = 10$ MHz
- SNR: 20 dB (which means $P/N = 100$)

Modern cellular systems (LTE, 5G) routinely achieve 50-75% of the Shannon limit. This represents 70+ years of progress in coding theory.

For a bandwidth W, we can make about W channel uses per second (this is a deep result from sampling theory—the Nyquist rate). So the channel capacity in bits per second is:

$$C_{\text{bits/sec}} = W \cdot C_{\text{bits/use}} = W \cdot \frac{1}{2}\log_2(1 + \text{SNR}).$$

With our numbers:

$$C = 10^7 \cdot \frac{1}{2}\log_2(101) = 10^7 \cdot \frac{6.66}{2} \approx 33 \text{ Mbps}.$$

Modern LTE achieves around 20-25 Mbps under similar conditions—about 70% of the theoretical limit. This represents 70 years of progress since Shannon's 1948 paper.

The Shannon Limit

The curve $C = \frac{1}{2}\log_2(1 + P/N)$ represents an absolute bound. No communication system can operate above this curve. Engineers call it the *Shannon limit*.

For decades, practical systems operated far below the Shannon limit. The gap between what was theoretically possible and what could actually be built seemed unbridgeable. In 1993, turbo codes achieved rates within 0.7 dB of capacity—the first practical codes to approach the limit. Around the same time, LDPC codes (originally proposed by Gallager in 1960 but ignored for decades) were rediscovered and shown to also approach capacity.

Shannon proved in 1948 that capacity was achievable. It took until 1993 to find codes that actually approached it. This is the gap between existence and construction—a recurring theme in information theory.

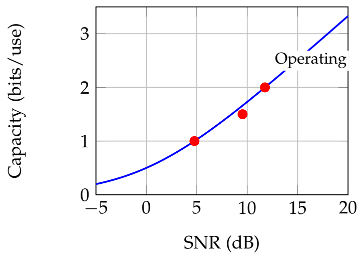


Figure 6.4: The Shannon limit for the Gaussian channel. No system can operate above this curve. Red points show typical operating regimes.

6.5 Water-Filling for Parallel Channels

So far, we have considered single channels. But many communication systems use multiple channels in parallel. A radio might transmit on several frequency bands simultaneously. A DSL modem uses thousands of parallel sub-channels. How should we allocate resources across channels?

Suppose we have K independent parallel Gaussian channels:

$$Y_k = X_k + Z_k, \quad k = 1, 2, \ldots, K$$

where each $Z_k \sim \mathcal{N}(0, N_k)$ has its own noise variance. We have a total power budget P to distribute among the channels:

$$\sum_{k=1}^{K} P_k \leq P, \quad P_k \geq 0.$$

The question is: how should we allocate power to maximize total capacity?

The Water-Filling Solution

Let me give you an image that captures the solution perfectly.

Imagine the noise levels $N_1, N_2, \ldots, N_K$ as the depths of K containers. You pour water (power) into these containers. Water naturally seeks its own level. The resulting allocation is:

The water-filling metaphor: pour water (power) into containers of different depths (noise levels). Water seeks its own level, putting more power where noise is low.

- Channels with less noise (shallower containers) get more power.
- Channels with more noise (deeper containers) get less power.
- Very noisy channels might get no power at all—it is not worth using them.

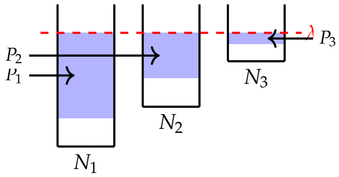


Figure 6.5: Water-filling: noise levels N_k are container depths. Power fills up to a common level λ. Noisier channels get less power.

Mathematically, the optimal power allocation is:

$$P_k = \max(\lambda - N_k, 0)$$

where λ (the "water level") is chosen so that $\sum_k P_k = P$.

Channels with $N_k < \lambda$ receive power $P_k = \lambda - N_k$. Channels with $N_k \geq \lambda$ receive no power—they are too noisy to be worth using.

A Worked Example

Consider three parallel channels with noise variances $N_1 = 1$, $N_2 = 2$, $N_3 = 5$. We have total power $P = 6$.

Attempt 1: Assume all three channels are used. The water level λ satisfies:

$$(\lambda - 1) + (\lambda - 2) + (\lambda - 5) = 6$$

$$3\lambda - 8 = 6 \implies \lambda = \frac{14}{3} \approx 4.67.$$

This gives:

$$\begin{aligned} P_1 &= 4.67 - 1 = 3.67 \\ P_2 &= 4.67 - 2 = 2.67 \\ P_3 &= 4.67 - 5 = -0.33 \end{aligned}$$

Negative power! Channel 3 is too noisy—at this water level, we should not use it at all.

Attempt 2: Use only channels 1 and 2.

$$(\lambda - 1) + (\lambda - 2) = 6$$

$$2\lambda - 3 = 6 \implies \lambda = 4.5.$$

This gives:

$$\begin{aligned} P_1 &= 4.5 - 1 = 3.5 \\ P_2 &= 4.5 - 2 = 2.5 \\ P_3 &= 0 \end{aligned}$$

All non-negative. This is the optimal allocation.

The noisiest channel gets no power at all. It is better to concentrate resources on good channels than to spread them thin.

Compute the total capacity:

$$\begin{aligned} C &= \frac{1}{2}\log_2\left(1 + \frac{3.5}{1}\right) + \frac{1}{2}\log_2\left(1 + \frac{2.5}{2}\right) + 0 \\ &= \frac{1}{2}\log_2(4.5) + \frac{1}{2}\log_2(2.25) \\ &\approx 1.09 + 0.59 = 1.68 \text{ bits per use.} \end{aligned}$$

Compare this to equal power allocation ($P/3 = 2$ per channel):

$$\begin{aligned} C_{\text{equal}} &= \frac{1}{2}\log_2(3) + \frac{1}{2}\log_2(2) + \frac{1}{2}\log_2(1.4) \\ &\approx 0.79 + 0.50 + 0.24 = 1.53 \text{ bits per use.} \end{aligned}$$

Water-filling gains about 10% over equal allocation in this example. The gain can be larger when channel qualities vary more dramatically.

Applications

Water-filling appears throughout modern communications.

OFDM (Orthogonal Frequency Division Multiplexing) is used in WiFi, LTE, 5G, and DSL. It divides the available bandwidth into many parallel sub-channels, each a narrow frequency band. The noise varies across frequencies (some bands are cleaner than others), and water-filling tells you how to allocate power optimally.

MIMO (Multiple-Input Multiple-Output) systems use multiple antennas to create parallel spatial channels. Water-filling determines how to split power across these spatial dimensions.

The principle is always the same: put more resources where they do the most good.

6.6 The Historical Moment

It is worth pausing to appreciate what Shannon accomplished in his 1948 paper, "A Mathematical Theory of Communication."

Shannon's 1948 paper is among the most influential scientific papers ever written. It created a new field and established results that engineers still rely on today.

Before Shannon, engineers thought about communication in terms of signals and noise. They knew that noise limited how well you could communicate, but they had no precise way to quantify the limit. They knew that redundancy could help combat errors, but they did not know how much redundancy was enough, or whether there was a fundamental tradeoff between rate and reliability.

Shannon changed everything. He showed that:

1. Information can be quantified precisely, using entropy.
2. Every channel has a capacity—a maximum rate of reliable communication.
3. Below capacity, reliable communication is possible. Not "mostly reliable," not "pretty good"—arbitrarily reliable, with error probability as small as you like.
4. Above capacity, reliable communication is impossible, no matter how clever you are.

The theorem was so counterintuitive that many engineers did not believe it. The idea that you could communicate with vanishingly small error over a noisy channel—as long as you stayed below capacity—seemed like magic.

"The fundamental problem of communication," Shannon wrote, "is that of reproducing at one point either exactly or approximately a message selected at another point."

"Frequently the messages have meaning... These semantic aspects of communication are irrelevant to the engineering problem." Shannon separated the physics of communication from its meaning—a crucial insight.

This framing was itself revolutionary. Shannon separated the problem of communication from the problem of meaning. The message might be a love letter or a stock price or a genome sequence. It does not matter. Information theory treats them all the same way: as probability distributions over possible messages.

The theorem was proved in 1948. But the proof was not constructive—it showed that good codes exist without showing how to find them. For the next 45 years, engineers searched for practical codes that approached capacity. Hamming codes (1950) corrected single errors.

Reed-Solomon codes (1960) became the workhorse of digital storage. Convolutional codes with Viterbi decoding (1967) powered deep-space communication.

But a significant gap remained between what these codes achieved and what Shannon said was possible.

In 1993, Berrou, Glavieux, and Thitimajshima at France Telecom introduced turbo codes. The key insight was an interleaver structure that enabled iterative decoding. Turbo codes operated within 0.5 dB of capacity—forty-five years after the theorem, the limit was finally within reach.

Around the same time, Robert McEliece at Caltech and David MacKay at Cambridge independently rediscovered LDPC codes, which Gallager had proposed in his 1960 PhD thesis but which had been ignored for decades. LDPC codes also approach capacity, and they power modern WiFi and 5G networks.

It took half a century to build what Shannon proved must exist. The theorem was true in 1948, but the engineering took until the 1990s.

6.7 *What Does Capacity Mean?*

Let us step back and ask a philosophical question: what does capacity really represent?

The *operational* meaning is clear: capacity is the rate below which reliable communication is possible and above which it is not. This is a theorem, proved mathematically, with no loopholes or approximations.

But why should there be such a sharp threshold?

You might expect a smooth tradeoff. Lower your rate, improve your reliability—but never achieve perfection, always have some residual error. This is how many engineering problems work. But communication is different.

The sharp threshold comes from the law of large numbers. With long codewords, typical behavior dominates. Below capacity, there are enough distinguishable codewords; above, there are not.

The resolution comes from the law of large numbers. The channel coding theorem relies on the statistics of long sequences. When we use the channel n times, the probability of an "atypical" event—an unusual pattern of errors that defeats our code—decays exponentially in n.

Below capacity, we can find 2^{nR} codewords that are sufficiently separated from each other. Each codeword occupies a "region" in the space of possible outputs, and the regions do not overlap (with high probability). When we receive a sequence, we can tell which codeword was sent.

Above capacity, we cannot find enough well-separated codewords. No matter how we choose them, their regions overlap. Different messages become indistinguishable.

Capacity is the rate at which we run out of room. Below it, we can fit all our codewords comfortably. Above it, crowding causes confusion.

Connection to Thermodynamics

There is a deep analogy between capacity and thermodynamic entropy.

Just as entropy measures the number of accessible microstates in a physical system, capacity measures the number of distinguishable codewords that can fit through a channel.

Just as the second law of thermodynamics describes typical behavior of large systems, the channel coding theorem describes typical behavior of long codes.

This connection is not a coincidence. Information entropy and thermodynamic entropy are the same thing viewed from different angles. We will explore this fully in Chapter 9.

Just as thermodynamic equilibrium maximizes entropy subject to constraints (fixed energy, volume, etc.), the capacity-achieving distribution maximizes mutual information subject to the channel constraints.

This is not mere analogy. As we will see in Chapter 9, information entropy and thermodynamic entropy are mathematically identical. The capacity theorem is, in a sense, a statement about the physics of communication.

6.8 Looking Ahead: The Coding Theorem

We have now computed the capacity of several important channels:

- **BSC**: $C = 1 - H_b(p)$
- **BEC**: $C = 1 - \varepsilon$
- **Gaussian**: $C = \frac{1}{2}\log_2(1 + P/N)$

We have seen that capacity is a meaningful quantity—it tells us exactly how much information can flow through a noisy channel. But we have not proved that capacity is achievable. So far, capacity is just a number: the maximum mutual information. Why should there exist codes that actually achieve this rate with vanishing error probability?

The next chapter proves the crown jewel of information theory: reliable communication is possible below capacity. Shannon's proof is one of the most beautiful in all of mathematics.

In the next chapter, we prove Shannon's noisy channel coding theorem—perhaps the most surprising result in all of information theory. The theorem has two parts.

The converse: If $R > C$, reliable communication is impossible. Any code operating above capacity has error probability bounded away from zero.

The achievability: If $R < C$, reliable communication is possible. There exist codes with error probability approaching zero as the block length grows.

The achievability proof is constructive: Shannon showed that *random codes* work. Pick codewords at random, and with high probability you will get a good code. This does not tell you how to find a good code efficiently—that took another 45 years—but it proves good codes exist.

The coding theorem transforms capacity from a mathematical definition into a physical law. It says that the mutual information between input and output is not just a measure of correlation; it is the operational limit on communication. Nature enforces this limit, and clever engineering can approach it.

Let us now see why this is true.

7 The Noisy Channel Coding Theorem

In 1947, if you had asked any engineer how to send reliable messages over a noisy telephone line, you would have received sensible, depressing advice. Repeat yourself. Send each bit three times, or five, or a hundred. The more you repeat, the more likely your message gets through correctly. But here is the cost: to cut your error rate in half, you must double your repetition. To cut it to a tenth, you must repeat ten times as often. In the limit of perfect reliability—zero errors, ever—you would need infinite repetition. Your actual rate of information transfer would drop to zero.

Before Shannon, engineers believed reliable communication required sacrificing rate. The more reliable you wanted to be, the slower you had to communicate. This seemed as inevitable as friction.

This was not just intuition. Engineers had worked with noisy channels for decades, and every practical system confirmed the tradeoff. You could have reliability or speed, but not both. The harder you pushed one, the more you sacrificed the other. It seemed as fundamental as the tradeoff between speed and fuel consumption, or between precision and cost. Some things cannot be escaped.

In 1948, Claude Shannon proved this intuition completely, spectacularly wrong.

Shannon showed that for any noisy channel, there exists a number C—the channel capacity we computed in the last chapter—with an astonishing property. At any rate below C, you can communicate with error probability as small as you like. Not "small but nonzero." Not "small if you are lucky." Arbitrarily small. As close to zero as you want, while communicating at nearly the maximum possible rate.

The cost? Not slower communication. Not more power. Not better hardware. Just cleverness. The right encoding scheme can achieve near-perfect reliability at rates approaching capacity.

Shannon's theorem says you can have both reliability and rate—up to a point. Below capacity, you lose nothing by demanding perfection. Above capacity, perfection is impossible at any rate.

The engineers did not believe it. How could you possibly communicate reliably over a channel that corrupts information? Where does the redundancy come from if not from repetition? What magic erases the noise?

The answer is that there is no magic—only mathematics. And the mathematics is among the most beautiful in all of science. Shannon did not construct a code that achieves capacity. He proved that such codes must exist, by an argument so elegant it feels like cheating. He counted. He showed that if you pick a code at random, it is almost certainly good enough. The actual construction of practical codes would occupy the best minds in coding theory for the next fifty years.

Let us understand both halves of this remarkable theorem: why rates above capacity are impossible, and why rates below capacity are achievable. The first is natural; the second is miraculous.

7.1 The Problem We Face

Let me set up the problem with complete precision, because the theorem's power lies in its exactness.

We have a sender who wants to communicate one of M possible messages to a receiver. The sender encodes message $w \in \{1, 2, \dots, M\}$ as a sequence of n channel inputs: $x^n = (x_1, x_2, \dots, x_n)$. This sequence goes through the noisy channel. The receiver observes the corrupted output $y^n = (y_1, y_2, \dots, y_n)$ and produces an estimate $\hat{w}$ of which message was sent.

An error occurs when $\hat{w} \neq w$—the receiver guesses wrong.

The *rate* of communication is

$$R = \frac{\log_2 M}{n} \quad \text{bits per channel use.}$$

The rate $R = (\log_2 M)/n$ measures information per channel use. We pack $\log_2 M$ bits of information into n channel uses.

If we want to send one of a million messages ($M = 10^6 \approx 2^{20}$) using 100 channel uses, our rate is $R = 20/100 = 0.2$ bits per channel use.

The question is: for what rates R can we make the error probability $P_e = P(\hat{w} \neq w)$ arbitrarily small?

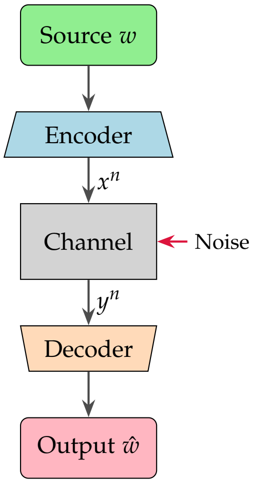


Figure 7.1: The communication system. Message w is encoded into codeword x^n, corrupted by the channel to produce y^n, then decoded to estimate $\hat{w}$.

The Two Parts of the Theorem

Shannon's theorem has two parts, and it is worth stating them separately because they feel so different.

Part I (The Converse—the pessimistic half): If $R > C$, then P_e is bounded away from zero. No matter how clever your encoder and decoder, no matter how long your codewords, you will make errors on a substantial fraction of messages. Reliable communication above capacity is impossible.

Part II (Achievability—the miraculous half): If $R < C$, then there exist codes with $P_e \to 0$ as $n \to \infty$. Reliable communication below capacity is not just possible—it is achievable with vanishing error probability.

Together, these two parts say that capacity is a sharp threshold. Below it, essentially perfect communication. Above it, inevitable errors.

There is no gradual tradeoff, no smooth decay of performance. Capacity is a cliff.

The theorem's structure—impossibility above, existence below—is the signature style of information theory. Define a fundamental limit, prove you cannot exceed it, prove you can approach it.

Notice what the theorem does *not* say. It does not say "here is how to build a good code." It says "good codes exist." This is an existence proof, not a construction. Shannon showed that if you search through all possible codes, you will find ones that work. Finding such codes efficiently—that is another matter entirely, one that took fifty years to resolve.

7.2 *The Converse: Why You Cannot Beat Capacity*

Let us prove the pessimistic half first. This is the natural part of the theorem, the part that confirms our intuition that you cannot push more water through a pipe than it can carry.

The Intuition

Suppose you want to send one of $M = 2^{nR}$ messages through n uses of the channel. Your message contains $\log_2 M = nR$ bits of information. But the channel can carry at most C bits of information per use, or nC bits total.

If $nR > nC$—that is, if $R > C$—you are trying to push more information through the channel than it can carry. Some information must be lost. And if information is lost, you cannot perfectly recover the message.

This is the essence of the converse. But let us make it precise.

The converse says: you cannot transmit more information than the channel can carry. If you try, information gets lost, and errors are inevitable.

Fano's Inequality: Connecting Errors to Information Loss

We need a mathematical tool that connects error probability to information loss. This tool is *Fano's inequality*, and it is one of the workhorses of information theory.

Let W be the message we sent (a random variable uniform over $\{1, \ldots, M\}$), and let $\hat{W}$ be the decoder's estimate. Define the error probability $P_e = P(W \neq \hat{W})$.

Fano's inequality says:

$$H(W|\hat{W}) \leq 1 + P_e \cdot \log_2(M - 1).$$

What does this mean? The left side, $H(W|\hat{W})$, is our remaining uncertainty about the true message W after we have seen the decoder's guess $\hat{W}$. If the decoder is always right, this uncertainty is zero—given $\hat{W}$, we know W exactly. If the decoder is often wrong, this uncertainty is large.

Fano's inequality: if you decode with low error probability, you must have learned most of the information about the message. High errors imply high residual uncertainty.

The right side bounds this uncertainty in terms of the error probability. When P_e is small, $H(W|\hat{W})$ must be small. When P_e is large, $H(W|\hat{W})$ can be large.

Let me sketch why this bound holds. Define an error indicator E: $E = 1$ if $W \neq \hat{W}$, $E = 0$ otherwise. We can decompose the uncertainty about W given $\hat{W}$:

$$\begin{aligned} H(W|\hat{W}) &= H(W, E|\hat{W}) - H(E|W, \hat{W}) \\ &= H(E|\hat{W}) + H(W|E, \hat{W}) \end{aligned}$$

since $H(E|W, \hat{W}) = 0$—once you know both the true message and the guess, you know whether they match.

Now, $H(E|\hat{W}) \leq H(E) \leq 1$ because E is binary. And $H(W|E, \hat{W})$ decomposes further:

- When $E = 0$ (no error): $H(W|E = 0, \hat{W}) = 0$ because $W = \hat{W}$.
- When $E = 1$ (error): $H(W|E = 1, \hat{W}) \leq \log_2(M - 1)$ because W could be any of the $M - 1$ wrong messages.

Putting it together:

$$H(W|\hat{W}) \leq 1 + P_e \cdot \log_2(M - 1) \leq 1 + P_e \cdot nR.$$

The Converse Proof

Now we can prove that communication above capacity fails.

Step 1: Information flows through the channel.

The message W contains $H(W) = \log_2 M = nR$ bits of entropy. After transmission, how much do we know about W from the received sequence Y^n?

By the definition of mutual information:

$$H(W) = I(W; Y^n) + H(W|Y^n).$$

The term $I(W; Y^n)$ is the information that the channel output reveals about the message. The term $H(W|Y^n)$ is our remaining uncertainty.

Step 2: Bound the information that flows.

How much information can the channel carry? The encoder maps W to a codeword X^n, which is then corrupted to produce Y^n. For a memoryless channel (where each symbol is corrupted independently):

$$I(W; Y^n) \leq I(X^n; Y^n) \leq \sum_{i=1}^{n} I(X_i; Y_i) \leq nC.$$

The first inequality is the data processing inequality: $W \to X^n \to Y^n$ is a Markov chain, so W cannot have more information about Y^n than X^n does. The second inequality uses the memoryless property. The third uses that $C = \max_{P(X)} I(X; Y)$.

The data processing inequality: you cannot increase information about W by processing Y^n. The decoder sees only Y^n, so $I(W; \hat{W}) \leq I(W; Y^n)$.

So the channel can carry at most nC bits of information about the message.

Step 3: Connect to the decoder via Fano.

The decoder produces $\hat{W}$ from Y^n. By the data processing inequality again:

$$I(W;\hat{W}) \leq I(W;Y^n) \leq nC.$$

But $I(W;\hat{W}) = H(W) - H(W|\hat{W}) = nR - H(W|\hat{W})$.

So: $H(W|\hat{W}) \geq nR - nC = n(R-C)$.

Step 4: Apply Fano's inequality.

From Fano: $H(W|\hat{W}) \leq 1 + P_e \cdot nR$.

Combining with Step 3:

$$n(R-C) \leq 1 + P_e \cdot nR.$$

Solving for P_e:

$$P_e \geq \frac{n(R-C)-1}{nR} = 1 - \frac{C}{R} - \frac{1}{nR}.$$

The Punchline: For any $R > C$, as $n \to \infty$:

$$P_e \geq 1 - \frac{C}{R} > 0.$$

The error probability is bounded away from zero. If you try to communicate above capacity, you will make errors on at least a fraction $(1 - C/R)$ of your messages. No amount of cleverness in code design can overcome this. The channel simply cannot carry that much information.

If $R > C$, then $P_e \geq 1 - C/R > 0$ for large n. A positive fraction of messages must be decoded incorrectly. No amount of cleverness can overcome this.

7.3 The Achievability: Shannon's Audacious Proof

The converse was natural: you cannot push more through than the pipe can carry. Now comes the miraculous half. Below capacity, you can communicate with arbitrarily small error probability.

This is where Shannon's genius shines. His proof is audacious. He does not construct a good code. He proves that *random* codes work. Pick codewords at random, and with high probability you will get a code that achieves capacity.

Shannon's random coding argument: generate a codebook randomly, prove the average error probability goes to zero, conclude that at least one specific code must be good. Existence without construction.

You might say, "But a random code will be terrible! Random codewords might overlap, or be too close to each other, or have no structure at all."

And you would be right—for small n. For short codes, randomness gives garbage. But as n grows, something remarkable happens. The codewords spread out in the high-dimensional space of all possible sequences. The curse of dimensionality becomes a blessing. There is so much room in high dimensions that random codewords almost never interfere with each other.

Let me show you how this works.

Joint Typicality: The Key Idea

Before proving the theorem, we need to understand a crucial concept: *joint typicality*. This is the decoding criterion Shannon used, and it is the heart of his proof.

Recall from Chapter 2 that a sequence x^n is *typical* for a distribution $P(X)$ if its empirical frequencies match the true probabilities. The typical set contains about $2^{nH(X)}$ sequences, and the probability that a randomly drawn sequence is typical approaches 1 as n grows.

Now extend this to pairs. A pair (x^n, y^n) is *jointly typical* for the joint distribution $P(X, Y)$ if:

- x^n is typical for $P(X)$
- y^n is typical for $P(Y)$
- The empirical joint frequencies of (x_i, y_i) pairs match $P(X, Y)$

Jointly typical sequences look like they came from the true joint distribution. Their empirical statistics match what the probability law predicts.

The *Joint Asymptotic Equipartition Property* tells us two crucial facts.

Fact 1: If (X^n, Y^n) are drawn from the true joint distribution $P(X, Y)$, then $P((X^n, Y^n) \text{ is jointly typical}) \to 1$ as $n \to \infty$.

Truly related sequences—an input and its actual output—are almost certainly jointly typical.

Fact 2: If $\tilde{X}^n$ is drawn independently from $P(X)$ and Y^n is drawn independently from $P(Y)$, then

$$P((\tilde{X}^n, Y^n) \text{ is jointly typical}) \approx 2^{-nI(X;Y)}.$$

Unrelated sequences—a random input that was *not* the one transmitted, paired with the received output—are almost certainly *not* jointly typical. The probability of a coincidental match decays exponentially fast.

The probability of a false match decays like $2^{-nI(X;Y)}$. This exponential decay is what makes joint typicality decoding work.

This is the miracle. Joint typicality is a filter. It lets the true message through and blocks the impostors. The true codeword stands out from the crowd because it is statistically related to the received sequence, while all other codewords are merely random.

The Random Coding Argument

Now let us prove that codes with vanishing error probability exist.

Setting up the random codebook.

Fix a block length n and a rate $R < C$. Let $M = 2^{nR}$ be the number of messages. Generate the codebook randomly:

For each message $w \in \{1, 2, \dots, M\}$, draw a codeword $X^n(w)$ by choosing each symbol independently from the capacity-achieving input distribution $P^*(X)$—the distribution that achieves $I(X; Y) = C$.

The codebook is a collection of M random codewords, each of length n.

The decoding rule.

When the receiver observes Y^n, look for a codeword $X^n(w)$ that is jointly typical with Y^n.

- If exactly one codeword is jointly typical with Y^n, decode as that message.
- If no codeword is jointly typical, or if multiple codewords are jointly typical, declare an error.

Joint typicality decoding: find the codeword that "looks like" it produced the received sequence. With high probability, only the true codeword qualifies.

Analyzing the error probability.

Suppose message w was sent. There are two ways decoding can fail.

Event E_1: The transmitted codeword $X^n(w)$ is not jointly typical with the received Y^n. The true signal got lost.

Event E_2: Some other codeword $X^n(w')$ with $w' \neq w$ is jointly typical with Y^n. An impostor snuck through.

The total error probability is $P_e \leq P(E_1) + P(E_2)$. We need both to vanish.

Bounding $P(E_1)$.

The codeword $X^n(w)$ was actually sent through the channel, so $(X^n(w), Y^n)$ was drawn from the true joint distribution $P(X, Y)$. By Fact 1 of the Joint AEP:

$$P(E_1) \to 0 \quad \text{as } n \to \infty.$$

The true codeword-output pair is almost certainly jointly typical.

Bounding $P(E_2)$.

Now for the crucial part. Consider any specific wrong codeword $X^n(w')$ where $w' \neq w$. This codeword was generated independently of everything—it was not transmitted, so it has no relationship to Y^n. By Fact 2:

$$P(X^n(w') \text{ is jointly typical with } Y^n) \leq 2^{-n(I(X;Y)-\epsilon)} = 2^{-n(C-\epsilon)}$$

for any small $\epsilon > 0$ and large enough n.

How many impostors are there? There are $M - 1 < 2^{nR}$ wrong codewords. By the union bound:

$$P(E_2) \leq \sum_{w' \neq w} P(X^n(w') \text{ is jointly typical with } Y^n) \leq 2^{nR} \cdot 2^{-n(C-\epsilon)} = 2^{-n(C-R-\epsilon)}.$$

The union bound: the probability that *any* impostor gets through is at most the sum of the individual probabilities. With 2^{nR} impostors, each with probability 2^{-nC}, the total is $2^{n(R-C)}$.

The Crucial Inequality.

If $R < C - \epsilon$, then $P(E_2) \to 0$ as $n \to \infty$.

Since ϵ can be arbitrarily small, for any $R < C$:

$$P_e \leq P(E_1) + P(E_2) \to 0 \quad \text{as } n \to \infty.$$

From Average to Existence.

We have shown that the *average* error probability, averaged over all random codebooks, goes to zero. But we want a *specific* code that works.

Here is the trick: if the average error probability is less than ϵ, then at least one codebook must have error probability less than ϵ. Otherwise, every codebook would have error probability at least ϵ, and the average would be at least ϵ—a contradiction.

Therefore: for any $R < C$ and any $\epsilon > 0$, there *exists* a code with error probability less than ϵ.

The Noisy Channel Coding Theorem. *For any discrete memoryless channel with capacity C:*
(Converse) If $R > C$, every code has P_e bounded away from zero as $n \to \infty$.
(Achievability) If $R < C$, there exist codes with $P_e \to 0$ as $n \to \infty$.

7.4 *Understanding Joint Typicality*

The proof went by quickly. Let me dwell on joint typicality, because it is the heart of the matter.

The Geometry of High Dimensions

Picture all possible received sequences y^n as points in a vast space—2^n points if the output alphabet is binary. When you transmit a codeword x^n through the channel, the received sequence Y^n wanders around a neighborhood of x^n. Sometimes it lands close to what was sent; sometimes noise pushes it farther away.

Define the *typical neighborhood* of x^n as all sequences y^n that are jointly typical with x^n. The Joint AEP tells us:

Each codeword has a "cloud" of likely outputs around it. Below capacity, the clouds do not overlap. Above capacity, they must.

1. The received sequence Y^n almost certainly lands in the typical neighborhood of the transmitted X^n.

2. The typical neighborhoods of different codewords almost never overlap.

This second fact is the key. In high dimensions, there is enough room for all the neighborhoods to stay separate. The sphere-packing picture from error correction is not quite right—the neighborhoods are not spheres, they are strangely shaped typical sets—but the intuition is similar.

How big are these neighborhoods? Each typical neighborhood contains roughly $2^{nH(Y|X)}$ sequences—the number of typical outputs given a particular input. The total space has $2^{nH(Y)}$ typical sequences. We are fitting $M = 2^{nR}$ neighborhoods into this space.

For the neighborhoods to fit without overlapping, we need:

$$M \cdot 2^{nH(Y|X)} \lesssim 2^{nH(Y)}.$$

Taking logarithms and dividing by n:

$$R + H(Y|X) \lesssim H(Y)$$

$$R \lesssim H(Y) - H(Y|X) = I(X;Y) \leq C.$$

Below capacity, there is room. Above capacity, there is not.

Why Doesn't Noise Destroy Everything?

This is the deep question. The channel adds noise. Bits get flipped. Symbols get corrupted. Why can the receiver still figure out what was sent?

The answer lies in the law of large numbers. Yes, the channel corrupts individual symbols. But the corruption follows a predictable pattern. A sequence that suffers "typical" noise still looks statistically related to its source. A sequence that suffers "atypical" noise—enough to push it into another codeword's neighborhood—is exponentially rare.

Noise corrupts individual symbols, but cannot corrupt the statistical structure of long sequences. The pattern survives even when individual bits do not.

You might say, "But what if the noise happens to look like a different codeword? What if random bit flips transform codeword 1 into something that looks like codeword 2?"

The answer is: yes, this can happen, but it is exponentially unlikely. There are exponentially many ways to corrupt a sequence. But only one way—statistically speaking—matches any particular wrong codeword. Random noise does not conspire to create meaningful patterns. It creates random patterns, which joint typicality decoding can identify as noise.

The receiver does not need to know which specific bits were flipped. It only needs to recognize that the overall pattern is consistent with a particular codeword. The law of large numbers guarantees that the true relationship between codeword and output will shine through.

7.5 *A Complete Worked Example*

Let us make this concrete with the binary symmetric channel.

The Setup

The BSC with crossover probability $p = 0.1$:

- Each bit is flipped independently with probability 0.1
- Capacity: $C = 1 - H_b(0.1) = 1 - 0.469 = 0.531$ bits per channel use

Consider a block length $n = 100$ and rate $R = 0.5 < C = 0.531$.

For the BSC with $p = 0.1$: 10% of bits are corrupted, yet we can reliably communicate at more than half a bit per channel use.

Counting the Codewords and Clouds

Number of messages: $M = 2^{nR} = 2^{50} \approx 10^{15}$.

Each codeword has a typical neighborhood—the set of received sequences that are jointly typical with it. For the BSC, this is approximately the set of sequences within Hamming distance about $np = 10$ of the codeword.

Size of typical neighborhood: roughly $\sum_{k \approx np} \binom{n}{k} \approx 2^{nH_b(p)} = 2^{100 \cdot 0.469} = 2^{46.9}$.

Do the Clouds Fit?

We have 2^{50} codewords, each with a typical neighborhood of size $2^{46.9}$. The total "volume" occupied is:

$$2^{50} \cdot 2^{46.9} = 2^{96.9}.$$

The total space of received sequences has size 2^{100}.

We are fitting $2^{96.9}$ into 2^{100}—plenty of room! The clouds can be non-overlapping.

With 2^{50} codewords and neighborhoods of size 2^{47}, the total volume is 2^{97}, fitting comfortably in the space of size 2^{100}.

What the Theorem Guarantees

At rate $R = 0.5$:

- We send one of 2^{50} messages using 100 channel uses
- About 10 bits will be flipped on average
- Despite these errors, the decoder can identify the correct message with probability approaching 1

At rate $R = 0.6 > C$:

- We would try to send one of 2^{60} messages
- Total cloud volume: $2^{60} \cdot 2^{46.9} = 2^{106.9}$
- This exceeds the space size 2^{100}—clouds must overlap
- Errors are inevitable

Rate R	Messages M	Cloud volume	Fits?	Error P_e
0.3	2^{30}	$2^{76.9}$	Yes	$\to 0$
0.4	2^{40}	$2^{86.9}$	Yes	$\to 0$
0.5	2^{50}	$2^{96.9}$	Yes	$\to 0$
0.531	$2^{53.1}$	2^{100}	Barely	Small
0.6	2^{60}	$2^{106.9}$	No!	> 0

Table 7.1: Sphere-packing picture for BSC with $p = 0.1$, $n = 100$. Below capacity, clouds fit; above, they overflow.

7.6 The Beautiful Symmetry

Let us step back and see a pattern that connects this chapter to Chapter 4.

Two Theorems, One Structure

Source Coding Theorem (Chapter 4):

- You cannot compress below entropy
- You can compress down to entropy

Channel Coding Theorem (this chapter):

- You cannot communicate above capacity
- You can communicate up to capacity

In both cases: a fundamental limit, achievable but not exceedable.

The proofs have the same structure. Both rely on typical sequences. Both use counting arguments. Both exploit the law of large numbers in high dimensions. In a sense, they are the same theorem viewed from two directions.

Source coding removes redundancy; channel coding adds it. They are inverses, and both achieve their limits through the same mechanism: typical sequences.

The Duality

Source coding removes redundancy from a message. Channel coding adds redundancy to protect against noise. They are inverses.

If you combine optimal source coding (compress to entropy H) with optimal channel coding (expand to capacity C), you can transmit H/C source symbols per channel use—exactly the limit.

This separation principle says you can design source and channel codes independently. Compress optimally, then protect optimally. The product of the two rates is the achievable end-to-end rate.

But notice: this only holds asymptotically. For finite block lengths, there can be gains from joint source-channel coding. The separation principle is a theoretical convenience, not an engineering mandate.

7.7 The Fifty-Year Quest

Shannon proved that good codes exist. He did not say how to find them.

This was not a minor detail. Shannon's random codes have complexity that grows exponentially with block length. To decode, you would need to compare the received sequence against all 2^{nR} codewords—an impossible computation for any practical n. The theorem guarantees a needle exists in the haystack; it does not help you find it.

Shannon's proof was non-constructive. He showed codes exist without showing how to build them. This gap between existence and construction took fifty years to close.

The Initial Reaction

At Bell Labs, some engineers were skeptical. The theorem was "interesting but useless," one reportedly said. How could you find these magical codes? How could you decode them in reasonable time?

Shannon himself seemed unconcerned. He had answered the fundamental question: what are the limits of communication? The engineering details would work themselves out.

The Search for Practical Codes

And work themselves out they did—eventually.

1950s: The First Codes. Richard Hamming, also at Bell Labs, developed the first practical error-correcting codes. Hamming codes correct single errors and detect double errors. They were far from capacity-achieving, but they worked. Peter Elias developed convolutional codes, which could be decoded efficiently with the Viterbi algorithm. Still not approaching capacity, but useful.

1960s-1980s: Algebraic Codes. BCH codes and Reed-Solomon codes emerged from the algebraic theory of finite fields. Reed-Solomon codes became the workhorse of digital storage—they protect CDs, DVDs, and QR codes. The Voyager spacecraft used concatenated Reed-Solomon and convolutional codes to send images from Jupiter and Saturn. Performance improved steadily, but Shannon's limit remained distant.

Voyager's cameras reached Jupiter in 1979. The images traveled 600 million kilometers, protected by codes that operated at perhaps 50% of capacity. Good enough for a revolution in planetary science.

The gap between what codes achieved and what Shannon proved possible seemed to be a permanent feature of the landscape. Perhaps, some thought, practical constraints made capacity unachievable. Perhaps Shannon's random codes were a theoretical curiosity with no engineering relevance.

1993: The Turbo Revolution. At a conference in Geneva, three engineers from France Telecom—Berrou, Glavieux, and Thitimajshima—presented a paper on "turbo codes." The name came from the iterative decoding algorithm, which recycled information like a turbocharger recycles exhaust.

The performance was stunning: within 0.5 dB of Shannon's limit. Forty-five years after the theorem, the limit was finally within reach.

The coding theory community was electrified. Some initially doubted the results. But verification came quickly, and the race was on to understand why turbo codes worked so well.

1995-present: LDPC Codes. It turned out that Robert Gallager had invented equally powerful codes in his 1960 PhD thesis: low-density parity-check (LDPC) codes. They had been forgotten for thirty years—the hardware of the 1960s could not implement the iterative decoding algorithm.

LDPC codes were invented in 1960, forgotten for 30 years, then rediscovered to become the standard in WiFi, 5G, and solid-state storage. Sometimes the world has to catch up to the mathematics.

Rediscovered by MacKay and Neal in the 1990s, LDPC codes matched and then exceeded turbo code performance. Today they power WiFi (802.11n/ac/ax), 5G cellular networks, and solid-state drives. They operate within a fraction of a decibel of capacity.

Shannon's "useless" theorem had become the most useful result in communications engineering.

7.8 *What the Theorem Really Means*

Let us step back from the mathematics and ask what this theorem tells us about the nature of communication.

Noise Is Not the Enemy

The naive view: noise is the enemy, and we must fight it. Send stronger signals. Repeat more often. Shout louder.

Shannon's view: noise is a constraint to be respected, not fought. The channel has a capacity—a maximum rate of information flow. Work within that rate, and noise cannot stop you. Exceed it, and nothing can save you.

Noise is not the enemy—it is a constraint. Below capacity, you can work around it. Above capacity, you cannot escape it.

This shift in perspective is profound. Nature gives and takes in a precise way. A noisy channel takes away reliability on individual symbols. But it gives back capacity—a fixed amount of information can still flow. The exchange rate is exact.

The theorem does not say communication is easy. It says communication is *possible*. The difference matters.

Existence Without Construction

Shannon proved codes exist without constructing them. This pattern is common in mathematics: we prove something exists before we know how to find it.

Some find this unsatisfying. What good is a theorem you cannot use?

But the existence proof did something crucial: it told engineers exactly what to aim for. Without Shannon's theorem, they would have been shooting in the dark. How close to capacity can we get? Is 50% efficiency the best possible, or can we do better? Shannon answered: you can get arbitrarily close to 100%.

The existence proof told engineers exactly what to aim for. Without it, they might have settled for 50% efficiency, thinking that was the best nature allowed.

Moreover, the random coding argument provides insight. Good codes are not rare—they are almost universal. Pick a code at random, and it probably works. The hard part is not finding a good code; it is finding one we can decode efficiently.

The Universality of Capacity

Why should there be a single number—capacity—that determines everything?

The theorem could have said: achievability depends on your computational resources, your specific encoding scheme, your decoder's sophistication. Instead, it says: capacity is the boundary, period.

Different codes, different decoders, different technologies—all bump against the same limit. Capacity is not a property of our codes. It is a property of the channel itself.

Here is nature revealing simplicity beneath complexity. A noisy channel is a messy, stochastic object. But its fundamental limitation is captured by one clean number: $C = \max I(X;Y)$.

This is what makes information theory a science and not just engineering.

7.9 Extensions and Caveats

Our proof covered the cleanest case. Let me note what we did and did not prove.

What We Proved

- **Discrete memoryless channels**: Each symbol is corrupted independently according to the same distribution.
- **Asymptotic block length**: We let $n \to \infty$. The theorem says nothing about $n = 100$ or $n = 1000$.
- **Average error probability**: We averaged over all messages. Any particular message might have higher error probability.

The theorem is asymptotic—it describes what happens as block length goes to infinity. For finite n, there is a gap between achievable rate and capacity.

What Remains

Finite block length. For practical n, there is a gap between achievable rate and capacity. Modern "finite block length" analysis quantifies this precisely. The gap shrinks like $1/\sqrt{n}$—not fast, but predictably.

Channels with memory. Real channels have memory: errors come in bursts, signal quality varies over time. The theory extends, but the analysis is more complex.

Maximum vs. average error. With a bit more work, one can show that the maximum error probability (over all messages) also goes to zero. No message is left behind.

Feedback. Does feedback help? If the transmitter can observe the channel outputs, can it do better? Shannon proved that feedback does *not* increase capacity for memoryless channels. You can already achieve capacity without feedback; knowing the channel outputs does not help. Feedback can simplify code design, but it cannot beat the fundamental limit.

Multiple users. Network information theory extends these ideas to multiple senders and receivers. The story becomes richer and is not fully resolved. Some capacity regions are known; others remain open problems.

7.10 Looking Ahead

We have proved the crown jewel of information theory. Reliable communication over noisy channels is possible up to capacity. The proof was non-constructive: we showed good codes exist by counting, not by building.

This existence proof changed how engineers think. It gave them a target: capacity. It told them the target was achievable. It even told them how to think about code design: make codewords "look different" to the channel, so that noise cannot confuse one for another.

Shannon proved the destination exists. Finding the path took fifty years. Chapter 8 explores that path—from Hamming codes to LDPC codes, from theory to practice.

But existence is not construction. Shannon proved codes exist; he did not say how to find them or decode them efficiently. The next chapter takes up this challenge.

We will see Hamming codes—the first step, elegant but limited. We will see linear codes and the beautiful algebra they bring. We will glimpse modern codes—the turbo and LDPC codes that finally, fifty years after Shannon, achieve what he proved possible.

Shannon told us the destination exists. Now we must find the path.

Historical note. Shannon's 1948 paper, "A Mathematical Theory of Communication," is among the most influential scientific papers ever written. It created a new field, established results that engineers still rely on, and connected disciplines from physics to linguistics. When asked late in life about the reception of his work, Shannon reportedly said that the theorem was "surprising to everybody, including me." The surprise was not that there were limits—limits are everywhere. The surprise was that the limits were so generous, and so precisely achievable.

8
Error-Correcting Codes

Imagine a mathematician proving that a treasure exists somewhere on a vast island. Is this useful? Perhaps—you now know the search is not hopeless. But you still have to dig. Shannon's theorem, which we proved in the last chapter, is exactly like this. It guarantees that codes achieving reliable communication at rates up to capacity must exist. It says nothing about how to find them.

Shannon proved that good codes exist. He did not construct any. This gap between existence and construction dominated coding theory for fifty years.

The engineers of 1948 immediately asked: "Show me the code." Shannon could not. His proof worked by counting—most random codes are good enough, so at least one good code must exist. But "most" random codes are also incomprehensible. They are gigantic lookup tables with 2^n entries, impossible to store and impossible to decode in reasonable time. A truly useful code needs *structure*: patterns we can exploit for efficient encoding, patterns we can exploit for efficient decoding, patterns we can analyze mathematically.

This chapter bridges the gap from existence to construction. We will build actual codes that correct actual errors. We start simple—repetition and parity checks, the brute-force approaches that everyone first imagines. We then develop Hamming codes, the first elegant construction, where a beautiful pattern emerges from asking the right question. We generalize to linear codes, a framework that encompasses most practical error-correcting codes. We ask what limits exist on code performance, discovering the sphere-packing bound. And we tell the story of how, after fifty years of searching, engineers finally found codes that achieve Shannon's limit.

The journey from Shannon's theorem (1948) to practical capacity-achieving codes (1993) took forty-five years. This chapter traces that journey.

The codes we develop here are not merely theoretical. Every wireless signal you send, every file you store, every photograph from deep space—all depend on error correction. The mathematics is beautiful; the engineering is everywhere.

8.1 Repetition: The Brute-Force Approach

Let us start with the simplest possible strategy for fighting noise. If the channel might corrupt a single bit, send that bit multiple times. Surely three copies are more reliable than one.

The Three-Repetition Code

We want to send one bit of information—either 0 or 1—over a binary symmetric channel (BSC) with crossover probability p. The BSC flips each bit independently with probability p.

The repetition code: to send 0, transmit 000; to send 1, transmit 111. Decode by majority vote. Simple, but wasteful.

The naive encoding scheme:

- To send 0: transmit 000
- To send 1: transmit 111

The decoder takes a majority vote. If at least two of the three received bits are 0, decode as 0. Otherwise, decode as 1.

Let us calculate the error probability. For $p = 0.1$—a reasonably noisy channel where one in ten bits gets flipped:

$$\begin{aligned} P(\text{correct}) &= P(\text{0 or 1 errors in 3 bits}) \\ &= (1-p)^3 + 3p(1-p)^2 \\ &= 0.729 + 0.243 \\ &= 0.972 \end{aligned}$$

So $P(\text{error}) = 0.028$. Compare this to uncoded transmission, where $P(\text{error}) = 0.1$. The repetition code has cut our error rate by a factor of 3.5.

For $p = 0.01$—a less noisy channel:

$$P(\text{error}) = 3p^2(1-p) + p^3 \approx 3(0.01)^2 = 0.0003$$

The improvement is even more dramatic: from 1% error to 0.03% error.

The Fatal Flaw

You might say, “Wonderful! Let us use more repetition. Five copies, seven copies, a hundred copies. Eventually we can make the error arbitrarily small.”

This is true. With n repetitions (odd n), the error probability is roughly $(np)^{n/2}$ for small p, which can be made as small as desired.

But consider what we sacrifice. We transmitted 3 bits to send 1 bit of information. The *rate* of the code is:

$$R = \frac{\text{information bits}}{\text{transmitted bits}} = \frac{1}{3} \approx 0.333$$

For a BSC with $p = 0.1$, recall from Chapter 6 that the channel capacity is:

$$C = 1 - H_b(p) = 1 - H_b(0.1) \approx 0.531 \text{ bits per channel use}$$

The capacity of a BSC with $p = 0.1$ is about 0.531 bits per use. The 3-repetition code achieves rate 0.333—throwing away nearly 40% of the available capacity.

Our repetition code operates at rate 0.333, far below the capacity of 0.531. We are leaving almost 40% of the channel's capacity unused!

Worse: if we increase repetition to reduce errors further, the rate drops. A 5-repetition code has rate $1/5 = 0.2$. A 7-repetition code has rate $1/7 \approx 0.143$. In the limit of perfect reliability, the rate approaches zero.

This is precisely the tradeoff that Shannon's theorem says we do not have to accept. Below capacity, we should be able to achieve arbitrarily low error probability *without* sacrificing rate. Repetition codes cannot do this. We need something cleverer.

8.2 Parity Checks: The Idea of Structured Redundancy

Instead of mindlessly repeating everything, let us add *just enough* redundancy to detect errors. This leads us to parity checks.

Single Parity Check

A single parity check can detect any single error, but cannot locate it. We know something is wrong, but not what.

Consider 4 bits of data. We add a fifth bit—the *parity bit*—chosen so that the total number of 1s is even.

For example, encoding the message 1011:

- Data bits: 1, 0, 1, 1
- Sum of data bits: $1 + 0 + 1 + 1 = 3$ (odd)
- Parity bit: 1 (to make the total even)
- Codeword: 10111

Now suppose we transmit 10111 and receive 10011—the third bit has flipped. We compute the parity of the received word: $1 + 0 + 0 + 1 + 1 = 3$, which is odd. The parity check fails, so we know an error occurred.

But we do not know *which* bit was corrupted. Was it the third bit? The first? The parity bit itself? A single parity check detects errors without locating them.

The rate is $R = 4/5 = 0.8$, much better than repetition. But we can only detect, not correct.

Multiple Parity Checks

Here is the key insight: what if different parity checks covered different subsets of bits? Then the *pattern* of which checks fail could pinpoint the error location.

The crucial insight: overlapping parity checks. Different checks cover different bit positions. The pattern of failures reveals the error location.

Let us work with 7 bits: 4 data bits and 3 parity bits. We design three parity checks, each covering a different subset:

	Pos 1	Pos 2	Pos 3	Pos 4	Pos 5	Pos 6	Pos 7
Check 1 covers:	×		×		×		×
Check 2 covers:		×	×			×	×
Check 3 covers:				×	×	×	×

Suppose bit 5 is corrupted. Which checks fail?

- Check 1 covers positions 1, 3, 5, 7: bit 5 is included, so Check 1 fails.
- Check 2 covers positions 2, 3, 6, 7: bit 5 is not included, so Check 2 passes.
- Check 3 covers positions 4, 5, 6, 7: bit 5 is included, so Check 3 fails.

The pattern of failures is (fail, pass, fail), or in binary: (1, 0, 1). Reading this as a binary number: $101_2 = 5$. The error is in position 5!

This is not a coincidence. Look at which positions each check covers:

- Check 1 covers positions whose binary representation has a 1 in the first position: 1, 3, 5, 7
- Check 2 covers positions with a 1 in the second position: 2, 3, 6, 7
- Check 3 covers positions with a 1 in the third position: 4, 5, 6, 7

The pattern of check failures literally spells out the error position in binary. Three checks can locate any single error among 7 positions, which is exactly $2^3 - 1 = 7$ positions.

We have just reinvented the Hamming code.

8.3 Hamming Codes: The First Elegant Construction

Richard Hamming developed these codes at Bell Labs in 1950, frustrated by the errors that kept crashing his weekend computing runs. His insight was the same as ours: structure the parity checks so that their failures form the binary address of the error.

The Hamming (7,4) Code

The Hamming (7,4) code: 4 data bits, 3 parity bits, corrects any single error. Named after Richard Hamming of Bell Labs, 1950.

The *Hamming (7,4) code* encodes 4 message bits into 7 codeword bits. The notation $(n, k) = (7, 4)$ means: codeword length $n = 7$, message length $k = 4$.

We place the bits strategically. The parity bits occupy positions that are powers of 2: positions 1, 2, and 4. The data bits fill the remaining positions: 3, 5, 6, and 7.

Position:	1	2	3	4	5	6	7
Binary:	001	010	011	100	101	110	111
Bit type:	p_1	p_2	d_1	p_3	d_2	d_3	d_4

Each parity bit checks all positions whose binary representation has a 1 in the corresponding place:

- p_1 (position 1) checks positions 1, 3, 5, 7 (binary: $**1$)
- p_2 (position 2) checks positions 2, 3, 6, 7 (binary: $*1*$)
- p_3 (position 4) checks positions 4, 5, 6, 7 (binary: $1**$)

Encoding: A Worked Example

Let us encode the 4-bit message $m = 1011$.

Step 1: Place the data bits in positions 3, 5, 6, 7.

Position:	1	2	3	4	5	6	7
Bit:	p_1	p_2	1	p_3	0	1	1

Step 2: Compute each parity bit.

p_1 checks positions 1, 3, 5, 7. Currently: $p_1 + 1 + 0 + 1 = p_1 + 2$. For even parity, we need $p_1 = 0$.

p_2 checks positions 2, 3, 6, 7. Currently: $p_2 + 1 + 1 + 1 = p_2 + 3$. For even parity, we need $p_2 = 1$.

p_3 checks positions 4, 5, 6, 7. Currently: $p_3 + 0 + 1 + 1 = p_3 + 2$. For even parity, we need $p_3 = 0$.

Encoding 1011: place data in positions 3,5,6,7, then compute parity bits. Result: codeword 0110011.

Step 3: The complete codeword is:

Position:	1	2	3	4	5	6	7
Codeword:	0	1	1	0	0	1	1

The codeword for message 1011 is 0110011.

Decoding and Error Correction

Suppose we transmit 0110011 but receive 0110111—bit 5 has flipped from 0 to 1.

Step 1: Compute the *syndrome*—the pattern of parity check results.

Check 1 (positions 1, 3, 5, 7): $0 + 1 + 1 + 1 = 3$ (odd). Result: $s_1 = 1$.

Check 2 (positions 2, 3, 6, 7): $1 + 1 + 1 + 1 = 4$ (even). Result: $s_2 = 0$.

Check 3 (positions 4, 5, 6, 7): $0 + 1 + 1 + 1 = 3$ (odd). Result: $s_3 = 1$.

The syndrome is $(s_1, s_2, s_3) = (1, 0, 1)$.

The syndrome is the pattern of parity check failures. For Hamming codes, the syndrome *is* the error position in binary.

Step 2: Interpret the syndrome as a binary number.

Reading $s_3 s_2 s_1 = 101_2 = 5$ in decimal. The error is in position 5.

Step 3: Correct the error by flipping bit 5.

Received: 0110111. After correction: 0110011.

Step 4: Extract the message from positions 3, 5, 6, 7.

Message: 1, 0, 1, 1. Correct!

Parameters and Performance

Let us summarize what the Hamming (7,4) code achieves:

- Codeword length: $n = 7$
- Message bits: $k = 4$
- Rate: $R = k/n = 4/7 \approx 0.571$
- Error correction: any single bit error
- Minimum distance: $d = 3$

The *minimum distance* deserves explanation. It is the smallest Hamming distance—the number of positions where they differ—between any two distinct codewords. For the Hamming (7,4) code, $d = 3$.

Minimum distance $d = 3$ means any two codewords differ in at least 3 positions. This allows correcting 1 error or detecting 2.

Why does minimum distance $d = 3$ imply single-error correction? If a codeword suffers one bit flip, it moves to a word at distance 1 from the original codeword. But every other codeword is at distance at least 3 from the original. The corrupted word is closer to the true codeword (distance 1) than to any impostor (distance at least 2). The decoder can always identify the correct codeword.

More generally, a code with minimum distance d can correct $t = \lfloor (d-1)/2 \rfloor$ errors. The corrupted word stays within a "sphere" of radius t around the true codeword, and these spheres do not overlap.

The Hamming Family

The Hamming (7,4) code is not unique. For any integer $r \geq 2$, there is a Hamming code with:

$$\begin{aligned} n &= 2^r - 1 && \text{(codeword length)} \\ k &= 2^r - 1 - r && \text{(message bits)} \\ d &= 3 && \text{(minimum distance)} \end{aligned}$$

r	n	k	Rate	Name
2	3	1	0.333	Hamming (3,1)
3	7	4	0.571	Hamming (7,4)
4	15	11	0.733	Hamming (15,11)
5	31	26	0.839	Hamming (31,26)

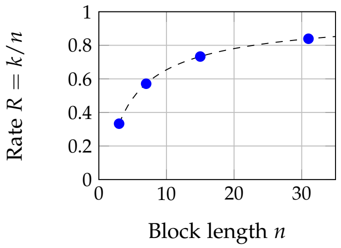


Figure 8.1: Hamming code rates approach 1 as block length increases, but all only correct single errors.

As r increases, the rate $R = (2^r - 1 - r)/(2^r - 1)$ approaches 1. We can get arbitrarily close to rate 1 while correcting single errors!

But notice the catch: all Hamming codes have $d = 3$ and correct only single errors. Higher rate does not mean more powerful error correction. For a highly noisy channel where multiple errors per block are common, even a long Hamming code will fail.

8.4 Linear Codes: A General Framework

Hamming codes have a beautiful property: the set of codewords is *closed under addition*. If c_1 and c_2 are codewords, then $c_1 + c_2$ (computed mod 2, meaning XOR) is also a codeword. This makes them *linear codes*, and linearity unlocks powerful tools.

Linear codes are closed under addition (mod 2). This seemingly simple property enables compact representation and efficient decoding.

The Generator Matrix

A linear code with k message bits and n codeword bits can be described by a *generator matrix* G, a $k \times n$ binary matrix. To encode message m (a k-bit row vector), compute:

$$c = m \cdot G$$

where multiplication is over $\mathbb{F}_2$ (binary arithmetic: $1 + 1 = 0$).

For the Hamming (7,4) code, one choice of generator matrix is:

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Let us verify with our earlier example. Encoding $m = (1, 0, 1, 1)$:

$$\begin{aligned} c &= (1,0,1,1) \cdot G \\ &= 1 \cdot (1,1,0,1,0,0,0) + 0 \cdot (1,0,1,0,1,0,0) + 1 \cdot (0,1,1,0,0,1,0) + 1 \cdot (1,1,1,0,0,0,1) \\ &= (1,1,0,1,0,0,0) + (0,1,1,0,0,1,0) + (1,1,1,0,0,0,1) \\ &= (0,1,0,1,0,1,1) \end{aligned}$$

Wait—this does not match the codeword 0110011 we computed earlier! What went wrong?

Nothing is wrong. The generator matrix I wrote encodes to a different *equivalent* form of the same codeword. Different generator matrices give different representations of the same code. The important point is that the encoding is a simple matrix multiplication.

With the generator matrix, encoding is a matrix multiplication. Different generator matrices can produce the same code in different "systematic" or "non-systematic" forms.

For a *systematic* code—one where the message bits appear unchanged in the codeword—we would use a different generator matrix where the last k columns form an identity matrix.

The Parity-Check Matrix

There is another way to describe a linear code: by the constraints its codewords must satisfy. The *parity-check matrix* H is an $(n - k) \times n$ binary matrix such that c is a codeword if and only if:

$$H \cdot c^T = 0$$

For the Hamming (7,4) code:

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Notice something remarkable: the columns of H are the binary representations of 1 through 7! Column 1 is $(1,0,0)^T = 1$ in binary. Column 5 is $(1,0,1)^T = 5$ in binary. Column 7 is $(1,1,1)^T = 7$ in binary.

This is not a coincidence. It is the whole point of the Hamming construction.

For Hamming codes, the columns of H are binary representations of 1 through n. This is why the syndrome gives the error position directly.

Syndrome Decoding

When we receive a word r, it might be a corrupted codeword: $r = c + e$, where c is the transmitted codeword and e is the error pattern.

Compute the *syndrome*:

$$s = H \cdot r^T = H \cdot (c+e)^T = H \cdot c^T + H \cdot e^T = 0 + H \cdot e^T = H \cdot e^T$$

The syndrome depends only on the error pattern, not on which codeword was sent! Every correctable error pattern produces a unique syndrome.

For a single-bit error in position j, the error vector e has a 1 only in position j. The syndrome is:

$$s = H \cdot e^T = \text{column } j \text{ of } H$$

For the Hamming code, column j of H is the binary representation of j. So the syndrome *is* the error position.

This is the magic of Hamming codes: decoding requires no lookup table. Compute the syndrome (3 parity checks), interpret it as a binary number, flip that bit. Done.

Minimum Distance for Linear Codes

For a linear code, the minimum distance has a simple characterization: it equals the minimum *weight* (number of 1s) of any nonzero codeword.

Why? Because for linear codes, the difference of two codewords is also a codeword. If c_1 and c_2 are codewords, then $c_1 - c_2 = c_1 + c_2$ (in binary) is also a codeword. The Hamming distance between c_1 and c_2 equals the weight of $c_1 + c_2$. So the minimum distance between any two codewords equals the minimum weight of any nonzero codeword.

For linear codes, minimum distance equals minimum weight. This follows because $c_1 - c_2 = c_1 + c_2$ is also a codeword, so distances equal weights.

This is useful for analysis: instead of checking all $\binom{2^k}{2}$ pairs of codewords, we just need to find the minimum-weight nonzero codeword.

8.5 The Sphere-Packing Bound

We have built codes that correct errors. But how good can codes be? Is there a fundamental limit on how many errors we can correct at a given rate?

Yes. The limit comes from a beautiful geometric argument: sphere packing.

Codewords as Sphere Centers

Think of codewords as points in the space of all n-bit binary vectors. Around each codeword, imagine a "sphere" of radius t: all vectors within Hamming distance t of that codeword.

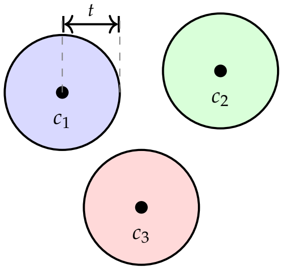


Figure 8.2: Codewords as sphere centers. Each sphere of radius t contains all words within Hamming distance t. Spheres must not overlap.

If the code corrects t errors, then when we receive a word within distance t of a codeword, we decode to that codeword. For this to work unambiguously, the spheres around different codewords must not overlap. Otherwise, a received word could be within distance t of two different codewords, and we would not know which was sent.

So the spheres must be disjoint. And they must all fit within the total space of n-bit vectors. This gives us a bound.

Sphere-packing: if t-error-correcting spheres around codewords must not overlap, and must fit in the space, we get a bound on how many codewords are possible.

The Hamming Bound

How many n-bit vectors are within Hamming distance t of a given codeword? We must count all vectors differing in 0, 1, 2, ..., or t positions:

$$V(n,t) = \sum_{i=0}^{t} \binom{n}{i}$$

There are $M = 2^k$ codewords, each with a sphere of size $V(n,t)$. The total "volume" of all spheres is $M \cdot V(n,t)$. This must fit in the space of all 2^n binary vectors:

$$2^k \cdot V(n,t) \leq 2^n$$

Taking logarithms:

$$k + \log_2 V(n,t) \leq n$$

This is the *Hamming bound* or *sphere-packing bound*.

Example: For $n = 7$ and $t = 1$ (single-error correction):

$$V(7,1) = \binom{7}{0} + \binom{7}{1} = 1 + 7 = 8$$

The bound gives:

$$2^k \cdot 8 \leq 2^7 = 128$$
$$2^k \leq 16$$

$$k \leq 4$$

The Hamming (7,4) code achieves $k = 4$ exactly! The spheres perfectly tile the space with no gaps and no overlaps. Such codes are called *perfect codes*.

A perfect code achieves the sphere-packing bound with equality. The only nontrivial perfect binary codes are Hamming codes (for $t = 1$) and the Golay code (for $t = 3$, $n = 23$).

Perfect Codes Are Rare

You might hope that for any n and t, we could find a perfect code that achieves the bound. Unfortunately, perfection is rare.

The only nontrivial perfect binary codes are:

- The Hamming codes (correcting $t = 1$ error)
- The binary Golay code with $n = 23$, $k = 12$, $t = 3$

That is it. For most combinations of n and t, perfect codes do not exist. The spheres cannot be packed without gaps.

Approaching Capacity

What does the sphere-packing bound say about approaching channel capacity?

For a code correcting a fraction δ of errors (so $t = \delta n$), the bound becomes, for large n:

$$R = \frac{k}{n} \leq 1 - H_b(\delta) + o(1)$$

where H_b is the binary entropy function.

For a BSC with crossover probability p, we need to correct roughly pn errors, so $\delta \approx p$. The bound says:

$$R \leq 1 - H_b(p)$$

But this is exactly the channel capacity $C = 1 - H_b(p)$! The sphere-packing bound, applied asymptotically, agrees with Shannon's theorem. We cannot do better than capacity.

The sphere-packing bound recovers Shannon's capacity as $n \to \infty$. This consistency is reassuring: geometry and information theory agree.

The sphere-packing bound is an upper bound—it says we cannot do better. Shannon's theorem says we can approach this limit. But for fifty years, the practical codes fell far short. Where were the codes that achieved capacity?

8.6 Beyond Single Errors

Before telling the story of how capacity was finally achieved, let us briefly survey the codes developed in the intervening decades.

BCH and Reed-Solomon Codes

Hamming codes correct single errors. What about multiple errors?

BCH codes (Bose-Chaudhuri-Hocquenghem, 1959-1960) generalize Hamming codes to correct multiple errors. They use finite field arithmetic—algebra over $\mathbb{F}_{2^m}$—to design parity checks that can locate multiple error positions.

Reed-Solomon codes are the workhorses of digital storage. They protect CDs, DVDs, QR codes, and deep-space communications.

Reed-Solomon codes are a special class of BCH codes that operate on symbols (groups of bits) rather than individual bits. They are particularly good at correcting burst errors—errors that cluster together. Reed-Solomon codes protect CDs, DVDs, QR codes, and deep-space communications. When the Voyager spacecraft sent images of Jupiter and Saturn across billions of kilometers, Reed-Solomon codes ensured the pictures arrived intact.

We will not develop the algebra of these codes—that would require a chapter on finite fields. But know that they exist, they are practical, and they pushed coding theory forward for decades.

Convolutional Codes

There is another approach entirely: instead of encoding fixed blocks, encode a continuous stream. *Convolutional codes* have memory—the current output depends on past inputs.

Convolutional codes encode streams rather than blocks. The Viterbi algorithm (1967) decodes them efficiently using dynamic programming.

Picture a shift register with XOR gates. As bits stream through, parity bits are computed from a sliding window of inputs. The encoder is simple; the decoder uses the celebrated *Viterbi algorithm* (1967), which finds the most likely transmitted sequence using dynamic programming.

Convolutional codes powered dial-up modems, early cell phones, and satellite communications. They are efficient to decode and perform well in practice.

But neither algebraic codes nor convolutional codes achieved Shannon's limit. By 1990, the best practical codes operated 2-3 dB from capacity—meaning they required 1.5-2 times the signal power of an ideal code. The gap had narrowed since 1948, but seemed stuck.

8.7 The Modern Breakthrough

The breakthrough came in 1993.

Turbo Codes: The Revolution

At the International Conference on Communications in Geneva, Claude Berrou and Alain Glavieux presented a paper claiming near-capacity performance. The coding theory community was skeptical—such claims had been made before and had not held up.

The results held up.

Turbo codes use a deceptively simple structure:

Turbo codes (1993): two simple encoders plus iterative decoding. Within 0.5 dB of Shannon's limit. The coding community was stunned.

1. Two convolutional encoders in parallel
2. The second encoder sees the data in scrambled (interleaved) order
3. An iterative decoder that passes "soft" information between two component decoders

The key is *iteration*. Each component decoder provides probabilistic information about the bits—not hard decisions, but likelihoods. This soft information passes to the other decoder, which refines its estimates. After 10-20 iterations, the combined decoder performs far better than either component alone.

Performance: within 0.5 dB of Shannon's limit. After forty-five years, the goal was in sight.

Berrou later said he did not realize how close to capacity he was until after the fact. He was just trying to build a good code. Sometimes engineering succeeds where theory stumbles.

LDPC Codes: Rediscovering Gallager

Here is a twist in the story. The codes that match turbo code performance were not new—they had been invented in 1962 by Robert Gallager in his PhD thesis.

Low-Density Parity-Check (LDPC) codes have sparse parity-check matrices. Most entries are 0; only a few are 1. But what exactly makes a matrix "low density"?

LDPC codes were invented in 1962, forgotten for 30 years, then rediscovered. Gallager's thesis was ahead of its time—computers could not decode his codes.

Consider the Hamming (7,4) code. Its parity-check matrix has 21 entries (3 rows × 7 columns), of which 12 are ones. That is a density of $12/21 \approx 57\%$—more than half the entries are nonzero. For an LDPC code with, say, $n = 1000$ bits and $n - k = 500$ check equations, a comparable density would mean 285,000 ones in a matrix with 500,000 entries.

An LDPC code might instead have only 3 ones per row and 6 ones per column. That gives $500 \times 3 = 1500$ ones total—a density of $1500/500{,}000 = 0.3\%$. This sparsity is the "low density" that gives LDPC codes their name and their power.

Density comparison: Hamming (7,4) has 57% ones. A typical LDPC code has less than 1% ones. This sparsity enables efficient decoding.

Why does sparsity help? Each parity check involves only a handful of bits—typically 3 to 6 in well-designed codes. This means each check provides focused information about a small subset of bits, and the decoding algorithm can process each check in constant time regardless of block length.

The Tanner Graph: A Crossword Puzzle

The structure of an LDPC code is best visualized as a *Tanner graph*—a bipartite graph with two types of nodes. *Variable nodes* represent codeword bits. *Check nodes* represent parity constraints. An edge connects variable node i to check node j if bit i participates in parity check j.

The crossword puzzle analogy illuminates why this structure helps. In a crossword, each letter belongs to two words—one across, one down. No single clue determines any letter uniquely. But the clues constrain each other. If 3-across must be a five-letter word meaning "swift" and 3-down must start with the same letter and mean "harvest," then the first letter is probably "R" (for RAPID and REAP).

LDPC codes work the same way. Each bit participates in several parity checks (typically 3 to 6). Each parity check constrains several bits (typically 6 to 20). No single check determines any bit. But the checks overlap, and information propagates through these overlaps.

The crossword analogy: each letter is constrained by two words (across and down). Each bit is constrained by multiple parity checks. The overlapping constraints enable inference.

Let us make this concrete with a small example.

A Worked Example: The (6,3) LDPC Code

Consider a tiny LDPC code with $n = 6$ bits and $k = 3$ message bits. The parity-check matrix is:

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

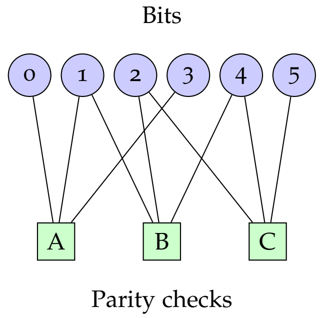


Figure 8.3: Tanner graph for a (6,3) LDPC code. Variable nodes (circles) represent bits. Check nodes (squares) represent parity checks. Each check constrains 3 bits.

Each row has exactly 3 ones (each check involves 3 bits). Columns 1, 3, 4, and 5 have 2 ones each; columns 0 and 2 have 1 and 2 ones. The density is $9/18 = 50\%$—not truly "low density," but small codes cannot be sparse. Real LDPC codes use thousands of bits.

Encoding. The code has $2^k = 8$ codewords. One systematic encoding places message bits in positions 0, 1, 2 and computes parity bits for positions 3, 4, 5. For message $m = (1, 0, 1)$:

- Check A: $c_0 + c_1 + c_3 = 0 \Rightarrow 1 + 0 + c_3 = 0 \Rightarrow c_3 = 1$
- Check B: $c_1 + c_2 + c_4 = 0 \Rightarrow 0 + 1 + c_4 = 0 \Rightarrow c_4 = 1$
- Check C: $c_2 + c_4 + c_5 = 0 \Rightarrow 1 + 1 + c_5 = 0 \Rightarrow c_5 = 0$

The codeword is $c = (1, 0, 1, 1, 1, 0)$.

Encoding the (6,3) code: message 101 becomes codeword 101110. Each parity check determines one parity bit.

Transmission and corruption. We transmit $c = (1, 0, 1, 1, 1, 0)$ over a noisy channel. Suppose bit 4 flips, giving received word $r = (1, 0, 1, 1, 0, 0)$.

Belief propagation decoding. Instead of computing a syndrome and looking up the error, LDPC decoders use *belief propagation*—an iterative algorithm where nodes exchange probabilistic messages.

Initially, each variable node has some belief about its value. For a binary symmetric channel with crossover probability $p = 0.1$, receiving a 0 means $P(\text{bit} = 0) = 0.9$ and $P(\text{bit} = 1) = 0.1$. We work with *log-likelihood ratios*: $L = \log \frac{P(\text{bit}=0)}{P(\text{bit}=1)}$. For a received 0, $L = \log(0.9/0.1) = \log 9 \approx 2.2$. For a received 1, $L = -2.2$.

Initial beliefs from $r = (1, 0, 1, 1, 0, 0)$:

Bit	0	1	2	3	4	5
L_{init}	-2.2	$+2.2$	-2.2	-2.2	$+2.2$	$+2.2$

Negative L means "probably 1"; positive L means "probably 0."

Log-likelihood ratios: $L > 0$ means "probably 0," $L < 0$ means "probably 1." The magnitude indicates confidence.

Round 1: Variable to check. Each variable node sends its belief to each connected check node. Bit 4 sends $L = +2.2$ to checks B and C, saying "I am probably 0."

Round 1: Check to variable. Each check node combines messages from its neighbors and sends back updated beliefs. Check B connects to bits 1, 2, 4. It received:

- From bit 1: $L_1 = +2.2$ (probably 0)
- From bit 2: $L_2 = -2.2$ (probably 1)
- From bit 4: $L_4 = +2.2$ (probably 0)

The parity check says $c_1 \oplus c_2 \oplus c_4 = 0$. If bits 1 and 4 are both 0, then bit 2 must be 0. But bit 2 says it is probably 1! The check senses a contradiction.

The message from check B back to bit 4 incorporates information from bits 1 and 2: "Given what bits 1 and 2 say, you should probably be 1, not 0." The formula involves hyperbolic tangents, but the intuition is: the check aggregates evidence from its other neighbors and tells each bit what value would satisfy the constraint.

After one round, bit 4 has received messages from checks B and C. Both checks, informed by their other neighbors, suggest bit 4 should be 1. This new evidence outweighs the initial channel observation.

One round of belief propagation: check B tells bit 4 "your neighbors suggest you should be 1." The evidence accumulates.

Convergence. After 3–5 iterations on this small example, the beliefs stabilize. Bit 4's total log-likelihood becomes negative (probably 1), while all other bits retain their original signs. The decoder outputs $(1, 0, 1, 1, 1, 0)$—the correct codeword.

Why Belief Propagation Works

Why does iterating messages converge to the right answer? The intuition comes from three observations.

First: Local consistency propagates. Each parity check enforces a local constraint. When beliefs satisfy all local constraints simultaneously, we have found a valid codeword. The iteration drives the system toward this globally consistent state.

Second: Evidence accumulates. Each variable node collects evidence from multiple check nodes, and each check node collects evidence from multiple variable nodes. A single corrupted bit is "outvoted" by its neighbors. Bit 4 in our example received evidence from two check nodes, each of which heard from two other bits—effectively, bit 4's belief incorporated information from bits 1, 2, and 5.

Third: The graph structure matters. Belief propagation gives exact answers on tree-structured graphs. LDPC codes are designed so that, locally, the Tanner graph looks like a tree. Short cycles (where a message can return to its origin after only a few hops) degrade performance; good LDPC codes avoid them.

Why BP converges: local constraints propagate globally; evidence accumulates from neighbors; tree-like graph structure enables exact inference locally.

You might ask, "But real LDPC codes have millions of bits. Does belief propagation still work?" Yes—and here is the remarkable fact. The computational cost per iteration scales linearly with the number of edges in the Tanner graph. A sparse graph with n bits and 3 ones per column has only $3n$ edges. Each iteration costs $O(n)$, and typically 20–50 iterations suffice. Total cost: $O(n)$, compared to $O(n^3)$ for solving linear systems directly.

How Close to Capacity?

The theoretical question that drove fifty years of research was: how close can practical codes get to Shannon's limit?

For a binary symmetric channel with $p = 0.05$ (5% bit flip probability), the capacity is $C = 1 - H_b(0.05) \approx 0.714$ bits per channel use. This means, in principle, we can transmit reliably at any rate below 0.714.

Modern LDPC codes operate within 0.0045 bits of capacity—99.4% of the theoretical limit. The fifty-year quest succeeded.

In 2001, Richardson and Urbanke showed that carefully designed "irregular" LDPC codes—where different variable nodes have different numbers of connections—can operate within 0.0045 bits of capacity. That is, at rate $R = 0.709$ instead of the theoretical maximum $R = 0.714$. We are capturing 99.4% of the channel's capacity.

For the additive white Gaussian noise channel (the model for wireless communication), the story is even better. The Shannon limit says we need a signal-to-noise ratio of about 0 dB to communicate at rate 1/2. Modern LDPC codes achieve reliable communication at 0.0045 dB above this limit—a gap so small it is essentially unmeasurable in practice.

When turbo codes appeared in 1993 operating 0.5 dB from capacity, the community was stunned. LDPC codes subsequently closed the gap by another factor of 100.

The Forgotten Thesis

In 1962, Robert Gallager proposed all of this in his PhD thesis at MIT. He described sparse parity-check matrices, the Tanner graph representation, and iterative decoding. He proved that randomly constructed LDPC codes approach capacity as block length increases.

But in 1962, computers were too slow to implement his decoder. A million-bit code requires billions of message updates per codeword. The coding community focused on algebraic codes—Reed-Solomon, BCH—that admitted fast non-iterative decoders. Gallager's thesis was cited occasionally but largely forgotten.

When turbo codes revived interest in iterative decoding in the 1990s, David MacKay and Radford Neal rediscovered LDPC codes. They showed that Gallager's thirty-year-old construction matched or exceeded turbo code performance. MacKay reportedly said he felt like an archaeologist discovering that an ancient civilization had already solved the problem.

Robert Gallager, by then a professor emeritus at MIT, lived to see his forgotten invention become the foundation of modern communications. His codes now transmit every WiFi signal, every 5G phone call, every photograph from deep space.

Modern Performance

Today, LDPC codes power:

- WiFi (802.11n, 802.11ac, 802.11ax)
- 5G cellular networks
- Solid-state drives (SSDs)
- Deep-space communication (DVB-S2)

The fifty-year quest is over. Shannon's "useless" existence theorem became the most useful result in communications engineering.

8.8 A Complete Worked Example

Let us ground everything in a concrete calculation: encoding a message with the Hamming (7,4) code, transmitting over a noisy channel, and decoding.

The Full Pipeline

Message: $m = (1, 0, 1, 0)$

Step 1: Encoding

Place data bits in positions 3, 5, 6, 7:

Position:	1	2	3	4	5	6	7
Bit:	p_1	p_2	1	p_3	0	1	0

Compute parity bits:

- p_1 (positions 1,3,5,7): $p_1 + 1 + 0 + 0 = 1$, so $p_1 = 1$
- p_2 (positions 2,3,6,7): $p_2 + 1 + 1 + 0 = 2$, so $p_2 = 0$
- p_3 (positions 4,5,6,7): $p_3 + 0 + 1 + 0 = 1$, so $p_3 = 1$

Codeword: $c = (1, 0, 1, 1, 0, 1, 0)$
Step 2: Transmission
We send $c = 1011010$ over a BSC with $p = 0.1$. Suppose bit 3 flips.
Received: $r = (1, 0, 0, 1, 0, 1, 0)$
Step 3: Syndrome Calculation

- Check 1 (positions 1,3,5,7): $1 + 0 + 0 + 0 = 1$ (odd) $\Rightarrow s_1 = 1$
- Check 2 (positions 2,3,6,7): $0 + 0 + 1 + 0 = 1$ (odd) $\Rightarrow s_2 = 1$
- Check 3 (positions 4,5,6,7): $1 + 0 + 1 + 0 = 2$ (even) $\Rightarrow s_3 = 0$

Syndrome: $(s_3, s_2, s_1) = (0, 1, 1) = 011_2 = 3$
Error location: position 3.
Step 4: Correction
Flip bit 3: $(1, 0, 1, 1, 0, 1, 0)$
Step 5: Extract Message
Read positions 3, 5, 6, 7: $(1, 0, 1, 0)$
Original message recovered!

Complete example: encode 1010 to 1011010, corrupt bit 3 to get 1001010, compute syndrome 011 = 3, flip bit 3, recover 1010.

Error Analysis

The Hamming (7,4) code corrects any single error. What is the probability of failure?

For a BSC with $p = 0.1$, failure occurs when 2 or more bits are corrupted:

$$\begin{aligned} P(\text{failure}) &= P(\geq 2 \text{ errors in 7 bits}) \\ &= 1 - P(0 \text{ errors}) - P(1 \text{ error}) \\ &= 1 - (1-p)^7 - 7p(1-p)^6 \\ &= 1 - 0.478 - 0.372 \\ &= 0.150 \end{aligned}$$

Compare to uncoded transmission of 4 bits:

$$P(\text{any error}) = 1 - (1-p)^4 = 1 - 0.656 = 0.344$$

The Hamming code cuts the error rate from 34% to 15%, while achieving rate $4/7 \approx 0.571$—much better than the $1/3$ rate of repetition coding.

For $p = 0.01$:

$$P(\text{failure, Hamming}) \approx \binom{7}{2}(0.01)^2 = 0.0021$$

$$P(\text{error, uncoded}) = 1 - 0.96 = 0.04$$

The Hamming code reduces errors by a factor of 20.

8.9 What Codes Teach Us

Let us step back from the details and ask what error-correcting codes reveal about communication and computation.

The Power of Structure

Error-correcting codes work by imposing structure on transmissions. Instead of using all possible bit strings—which noise turns into chaos—we use a carefully chosen subset. The structure is not arbitrary; it is designed to spread codewords apart in Hamming space.

This is a general principle worth remembering: structured constraints can help, not hinder. The constraints of rhyme and meter do not make poetry harder to remember—they make it easier. The structure of musical scales does not limit composers—it gives them a vocabulary. The structure of codewords does not waste capacity—it creates reliability.

Structure is not a constraint to be minimized. In coding, structure enables reliability. In poetry, rhyme and meter aid memory. The right constraints can help.

Theory Guides Practice

Shannon's theorem told engineers where to aim. Without it, they might have wasted decades trying to beat capacity, or settled for 50% efficiency thinking that was the best nature allowed.

The theorem did not say *how* to achieve capacity. But it said the target was achievable. It defined success. When Berrou's turbo codes appeared in 1993, the community knew immediately how significant they were—they could measure the gap to capacity.

Theory and practice are not separate enterprises. Theory tells us what is possible; practice shows us how to get there. Neither is complete without the other.

Shannon's theorem defined success. Without it, engineers would have been shooting in the dark. With it, they knew exactly what to aim for.

The Surprise of Simplicity

The codes that finally achieved capacity—turbo codes and LDPC codes—are conceptually simple. Two encoders and an interleaver. Sparse matrices and message passing. The individual components are nothing special. The magic lies in the combination.

Gallager's LDPC codes waited thirty years for computers fast enough to decode them. Turbo codes' iterative decoding was not a theoretical breakthrough—it was an engineering trick that worked unexpectedly well.

Sometimes the answer has been hiding in plain sight. Simplicity is worth pursuing, even when it seems insufficient.

8.10 Looking Ahead

We have traveled from existence to construction. Shannon proved good codes exist; we have built them. From repetition codes to Hamming codes to LDPC codes, we have seen how structure enables reliability.

This completes the engineering story of information theory. We know how to compress messages (Chapter 5). We know the limits of noisy channels (Chapters 6-7). We know how to build codes that achieve those limits (this chapter).

Movement II concludes. We have conquered communication—the engineering of information. Movement III reveals that information theory is also physics.

But there is more. Information theory is not just engineering. It connects to physics in ways that surprised even Shannon.

In the next chapter, we turn from engineering to thermodynamics. We will discover that Shannon's entropy and Boltzmann's entropy—the entropy of physics, of heat engines and the second law—are the same thing. Not analogous. Not similar. The same.

Maxwell imagined a demon who could violate the second law by sorting molecules. Landauer asked what happens physically when you erase a bit of information. The answers to these questions connect information to the deepest structures of physical law.

The engineering was beautiful. The physics is profound.

Historical note. The fifty-year gap between Shannon's theorem (1948) and practical capacity-achieving codes (1993) is one of the great stories in applied mathematics. Shannon showed the destination existed; finding the path required contributions from hundreds of researchers across multiple continents. When Gallager's 1962 codes were finally recognized in the 1990s, he was asked how he felt about the thirty-year delay. "Sometimes," he reportedly said, "you're ahead of your time. And then you're not." His codes now transmit billions of messages every second.

9 Information and Thermodynamics

Here is a puzzle that troubled the best minds of the nineteenth century. A box of gas sits in equilibrium. Every molecule darts about, colliding with walls and with other molecules, tracing an unimaginably complex trajectory. If we knew the position and velocity of every molecule—all 10^{23} of them—we could, in principle, predict the future perfectly. The laws of mechanics are reversible; nothing is lost. And yet, when we watch a box of gas, we see irreversibility everywhere. The gas spreads out and never spontaneously contracts. Heat flows from hot to cold and never reverses. The universe runs down. Where does the arrow of time come from? The laws of physics do not have one.

The central puzzle of thermodynamics: the microscopic laws are reversible, yet the macroscopic world runs irreversibly toward equilibrium. Boltzmann's resolution invoked counting.

Boltzmann's answer, in 1877, was entropy. But his entropy was a puzzle in itself—a quantity defined in terms of counting microscopic arrangements, seemingly unrelated to the thermodynamic entropy that Clausius had defined through heat and temperature. Then, seventy years later, Shannon defined his own entropy to measure uncertainty in communication. The formulas look identical. One entropy is about steam engines; the other is about telegraph wires. What could they possibly have in common?

Everything. They are the same thing. This chapter will show you why, and the demonstration will change how you think about both physics and information. The second law of thermodynamics—perhaps the most profound principle in all of science—is, at bottom, a statement about *information*. Entropy increases because we lose track of things. The arrow of time points in the direction of forgetting.

9.1 Boltzmann's Revolution

Before Boltzmann, thermodynamic entropy was an accounting device. Clausius had defined it operationally in the 1850s: integrate the heat transferred divided by temperature along a reversible path. The result, which he called entropy (from the Greek *entropia*, "transformation"), had peculiar properties. It always increased in isolated systems. It told you which processes were possible and which were forbidden. It was the arbiter of the second law. But no one knew what it *meant*.

Boltzmann asked a dangerous question: what if gases are made of atoms?

In 1877, atoms were controversial. Ernst Mach called them "metaphysical." Boltzmann's statistical mechanics was attacked as unphysical speculation.

This was controversial in 1877. Many physicists considered atoms a convenient fiction—a calculational trick with no physical reality. Mach, Ostwald, and others insisted that thermodynamics could be done without reference to invisible, hypothetical particles. Boltzmann disagreed. If gases *are* made of atoms, he reasoned, then a macroscopic state—"the gas has this pressure and temperature"—corresponds to many microscopic states—"molecule 1 is here with this velocity, molecule 2 is there with that velocity. . . "

How many microscopic states? That was the question. And Boltzmann's answer was the formula that would be carved on his tombstone:

$$S = k_B \ln W$$

Here W is the number of microscopic arrangements (what we would now call *microstates*) compatible with a given macroscopic description (the *macrostate*). The constant k_B, now called Boltzmann's constant, is approximately 1.38×10^{-23} joules per kelvin.

You might ask: why the logarithm? Why not just use W itself?

The logarithm converts multiplicative counting to additive entropy. This is the same mathematical structure as Shannon entropy.

The answer is additivity. Consider two independent systems A and B. The number of microscopic arrangements for the combined system is $W_A \times W_B$—each arrangement of A can combine with each arrangement of B. But we want entropy to be additive: $S_{A+B} = S_A + S_B$. The logarithm converts multiplication to addition:

$$S_{A+B} = k_B \ln(W_A W_B) = k_B \ln W_A + k_B \ln W_B = S_A + S_B$$

This reasoning should sound familiar. In Chapter 2, we used exactly the same argument to derive Shannon entropy. The mathematical structure is identical.

A Concrete Example: The Shuffled Deck

Let us make this tangible with a metaphor we will develop throughout the chapter.

Imagine a deck of 52 playing cards. When the deck is perfectly sorted—ace through king in each suit, suits in a fixed order—you know exactly which card is in which position. There is only one arrangement that counts as "sorted." The entropy is:

$$S_{\text{sorted}} = k_B \ln(1) = 0$$

A sorted deck has zero entropy—one arrangement, no uncertainty. A shuffled deck has maximum entropy—52! arrangements, complete uncertainty about which one.

No uncertainty means no entropy.

Now shuffle the deck thoroughly. There are 52! possible orderings, and for a well-shuffled deck, each is equally likely. The entropy is:

$$S_{\text{shuffled}} = k_B \ln(52!) \approx k_B \times 156$$

That is the maximum possible entropy for a deck of cards. You have no information about which arrangement you are holding. The deck could be in any of about 8×10^{67} configurations, and you cannot distinguish among them.

What about a partially shuffled deck—one that has been through a few riffle shuffles but is not yet random? It has intermediate entropy. Some orderings are more probable than others (cards that were nearby tend to stay nearby), but you still have substantial uncertainty.

The second law, applied to card decks, says: if you shuffle randomly, entropy tends to increase. A sorted deck becomes shuffled; a shuffled deck does not become sorted. Why? Not because of any special property of card dynamics—every ordering is mechanically equivalent, obeying the same laws of motion. The reason is pure combinatorics. There are vastly more shuffled arrangements than sorted ones. A random process almost certainly lands in the larger set.

9.2 The Gibbs Entropy

Boltzmann's formula $S = k_B \ln W$ implicitly assumes that all W microstates are equally probable. This is the case for an isolated system in equilibrium—the "microcanonical ensemble" in the language of statistical mechanics.

But what if states are not equally likely? What if the system is in contact with a heat bath at temperature T, and different energy states have different probabilities?

J. Willard Gibbs, working independently in America, developed a more general formulation. Let p_i be the probability that the system is in microstate i. Then the entropy is:

Gibbs generalized Boltzmann's formula to arbitrary probability distributions over microstates. The result is identical to Shannon entropy.

$$S = -k_B \sum_i p_i \ln p_i$$

Stop. Look at that formula again.

Compare it to Shannon entropy, which we derived in Chapter 2:

$$H = -\sum_i p_i \log_2 p_i$$

They are the same formula. The only differences are the constant in front (k_B versus 1) and the base of the logarithm (natural versus base-2). These are just unit conversions—the mathematical structure is identical.

Let us verify that Gibbs entropy reduces to Boltzmann entropy when all states are equally probable. If $p_i = 1/W$ for all i:

$$\begin{aligned} S &= -k_B \sum_{i=1}^{W} \frac{1}{W} \ln \frac{1}{W} \\ &= -k_B \cdot W \cdot \frac{1}{W} \cdot (-\ln W) \\ &= k_B \ln W \end{aligned}$$

The generalization is consistent.

Gibbs understood something profound: statistical mechanics is fundamentally about probability distributions over states, not about tracking individual trajectories. The entropy measures how "spread out" the distribution is—how uncertain we are about which microstate the system occupies.

This is exactly what Shannon entropy measures: uncertainty.

9.3 Maximum Entropy and the Boltzmann Distribution

Here is a foundational question of statistical mechanics: given that a system has a fixed average energy, what probability distribution should we assign to its microstates?

You might say: who are we to assign probabilities? The system is in some definite microstate—we just do not know which one.

True. But that is precisely the point. *Because* we do not know which microstate, we must assign probabilities that reflect our ignorance. And the principle of maximum entropy says: spread your probability as widely as possible, subject to what you actually know.

The principle of maximum entropy: among all distributions consistent with what you know, choose the one with maximum entropy. This is not physics—it is logic.

Let us derive the Boltzmann distribution from this principle.

We want to maximize:

$$S = -k_B \sum_i p_i \ln p_i$$

subject to two constraints:

1. Normalization: $\sum_i p_i = 1$
2. Fixed average energy: $\sum_i p_i E_i = \langle E \rangle$

Using Lagrange multipliers α and β for the two constraints:

$$\mathcal{L} = -k_B \sum_i p_i \ln p_i - \alpha \left(\sum_i p_i - 1 \right) - \beta \left(\sum_i p_i E_i - \langle E \rangle \right)$$

Taking the derivative with respect to p_i and setting it to zero:

$$-k_B(\ln p_i + 1) - \alpha - \beta E_i = 0$$

Solving for p_i:

$$p_i = \exp\left(-\frac{\alpha + k_B}{k_B}\right) \exp\left(-\frac{\beta E_i}{k_B}\right)$$

The normalization constraint determines the first factor. Defining the *partition function*

$$Z = \sum_i \exp\left(-\frac{\beta E_i}{k_B}\right)$$

we find:

$$p_i = \frac{1}{Z} \exp\left(-\frac{\beta E_i}{k_B}\right)$$

This is the canonical Boltzmann distribution. Identifying $\beta = 1/T$ (where T is the temperature):

$$p_i = \frac{1}{Z} \exp\left(-\frac{E_i}{k_B T}\right)$$

The Boltzmann distribution emerges from maximum entropy subject to fixed average energy. We did not assume it—we derived it from the requirement of maximum ignorance.

The probability of a state decreases exponentially with its energy, at a rate set by the temperature.

We did not assume this distribution. We *derived* it from the requirement that entropy be maximized subject to knowing only the average energy. The Boltzmann distribution is the most uncertain distribution consistent with our constraints—the distribution of maximum ignorance.

A Worked Example: The Two-State System

Consider a simple system with two states: a ground state with energy $E_0 = 0$ and an excited state with energy $E_1 = \varepsilon$.

The partition function is:

$$Z = 1 + e^{-\varepsilon/(k_B T)}$$

The probabilities are:

$$p_0 = \frac{1}{Z} = \frac{1}{1 + e^{-\varepsilon/(k_B T)}}$$

$$p_1 = \frac{e^{-\varepsilon/(k_B T)}}{Z} = \frac{e^{-\varepsilon/(k_B T)}}{1 + e^{-\varepsilon/(k_B T)}}$$

At low temperature, the system freezes into the ground state. At high temperature, both states become equally likely. Temperature controls how probability spreads across energy levels.

What happens at different temperatures?

As $T \to 0$: $e^{-\varepsilon/(k_B T)} \to 0$, so $p_0 \to 1$ and $p_1 \to 0$. The system freezes into its ground state.

As $T \to \infty$: $e^{-\varepsilon/(k_B T)} \to 1$, so $p_0 \to 1/2$ and $p_1 \to 1/2$. The states become equally likely.

This is what temperature *means*: it controls how probability spreads across energy levels. Low temperature concentrates probability on low-energy states. High temperature spreads it out.

The partition function Z—this innocent-looking sum—contains all of thermodynamics. Free energy, entropy, specific heat, pressure: everything can be computed from Z. It emerged here as a normalization constant, but it is the master key to statistical mechanics.

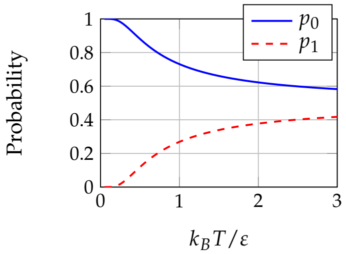


Figure 9.1: Occupation probabilities for a two-state system. At low temperature, the ground state dominates. At high temperature, both states approach equal probability.

9.4 *Shannon Equals Boltzmann*

Now let us make the connection explicit. We have two formulas:

$$\text{Shannon:} \quad H = -\sum_i p_i \log_2 p_i \quad \text{(bits)}$$

$$\text{Gibbs:} \quad S = -k_B \sum_i p_i \ln p_i \quad \text{(joules/kelvin)}$$

The mathematical structure is identical. The differences are:

1. The base of the logarithm (2 versus e)
2. The constant in front (1 versus k_B)

Since $\ln x = \log_2 x \times \ln 2$, we can convert:

$$\begin{aligned} S &= -k_B \sum_i p_i \ln p_i \\ &= -k_B \ln 2 \sum_i p_i \log_2 p_i \\ &= k_B \ln 2 \times H \end{aligned}$$

Thus:

$$\boxed{S = k_B \ln 2 \times H} \qquad (9.1)$$

One bit of Shannon entropy equals $k_B \ln 2$ joules per kelvin of thermodynamic entropy.

One bit of Shannon entropy equals $k_B \ln 2 \approx 10^{-23}$ J/K of thermodynamic entropy. This is the conversion factor between information and physics.

Let us compute this conversion factor:

$$k_B \ln 2 = (1.38 \times 10^{-23}\ \text{J/K}) \times (0.693) \approx 0.96 \times 10^{-23}\ \text{J/K}$$

This tiny number is the bridge between the abstract world of information and the concrete world of thermodynamics.

A Bit Has a Physical Size

Consider a single bit of information—the answer to a yes/no question—at room temperature ($T = 300$ K).

Its entropy is:

$$S = k_B \ln 2 \approx 10^{-23}\ \text{J/K}$$

The energy associated with erasing this bit (as we will see in Chapter 10) is:

$$E = T \times S = 300 \times 10^{-23} \approx 3 \times 10^{-21}\ \text{J}$$

This is about 0.02 electron volts—tiny, but not zero. It is the minimum energy required to erase one bit of information irreversibly. This is Landauer's principle, and we will explore it in the next chapter.

Why Does This Connection Exist?

You might ask: is this just a mathematical coincidence? Why should Shannon's engineering quantity have anything to do with Boltzmann's physics?

The answer is profound: they are both measuring the same thing—*uncertainty about microscopic states*.

Both entropies measure the same thing: uncertainty about microscopic states. Shannon measures uncertainty about messages; Boltzmann measures uncertainty about molecular configurations.

Shannon entropy measures your uncertainty about which message was sent.

Gibbs entropy measures your uncertainty about which microstate the system is in.

Both are measuring how "spread out" a probability distribution is. The formulas are identical because the concept is identical.

This is not an analogy or a metaphor. Information is physical. It is encoded in physical systems—voltages in a computer, spins of electrons, positions of atoms. Processing information means physically manipulating these systems. And physics constrains what manipulations are possible.

Rolf Landauer of IBM stated it memorably in 1961: "Information is physical."

The laws of thermodynamics are not merely analogous to information theory. They *are* information theory, applied to systems with 10^{23} degrees of freedom.

9.5 *The Second Law as Information Loss*

Now we can reinterpret the most important law in thermodynamics.

The classical statement of the second law: "The entropy of an isolated system never decreases."

This is the most universal law in physics. It holds for gases, liquids, solids, plasmas, black holes, the entire universe. It gives time its direction.

But what does it *mean*?

Entropy increase means information loss. We become more uncertain about the microstate, not because information is destroyed, but because we cannot track it.

Entropy measures uncertainty about the microstate. Saying "entropy increases" is the same as saying "we become more uncertain about the microstate."

But wait—if the laws of mechanics are deterministic, how can we become more uncertain? If we knew the exact microstate at time t, we could compute it exactly at time $t + 1$. No information is lost at the microscopic level.

The resolution lies in what we actually know. We never know the exact microstate. We know the macrostate—pressure, temperature, volume. Many microstates correspond to one macrostate. As the system evolves, it visits regions of phase space compatible with macrostates of higher entropy.

Coarse-Graining

Imagine dividing the space of all possible microscopic configurations—called *phase space*—into cells. Each cell corresponds to a macrostate: all the microscopic configurations that we cannot distinguish with our macroscopic measurements.

At time $t = 0$, we know the system is in some cell. As time passes, the microscopic configurations that started in that cell spread to other cells. They explore phase space, entering regions corresponding to different macrostates. Soon they occupy many cells.

Our uncertainty about which cell contains the system has increased. This is entropy increase.

The microscopic dynamics lose no information—they are deterministic and reversible. But we, with our coarse-grained vision, lose track. We can no longer say which cell the system is in.

Entropy increase is not a property of the dynamics. It is a property of our knowledge.

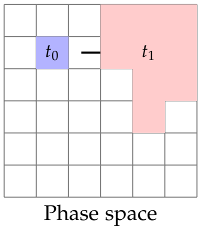


Figure 9.2: Phase space spreading. At t_0, the system is in one cell. At t_1, the microscopic states have spread to many cells. Our uncertainty about which cell contains the system has increased.

The Shuffled Deck Revisited

Return to our deck of cards. Start with a sorted deck—one macrostate, one microstate, zero entropy.

Now shuffle randomly. After each shuffle, the deck could be in many orderings. Your uncertainty grows. Entropy increases.

But notice: the laws of card motion (whatever they are) do not prefer disorder over order. Every ordering is mechanically equivalent. If you filmed a shuffle and ran the film backward, the reversed shuffle would be equally valid.

So why does shuffling increase entropy? Because there are vastly more shuffled configurations than sorted ones. A random process explores configurations without preference, and almost all configurations are "disordered." It is overwhelmingly probable that the deck becomes and stays shuffled.

The second law emerges from combinatorics, not dynamics. There are vastly more disordered configurations than ordered ones. Random exploration almost certainly finds disorder.

This is the essence of the second law. It is not a law of dynamics. It is a law of counting.

The Arrow of Time

This raises a deep puzzle. The microscopic laws of physics are symmetric in time—run them backward, and they still work. A film of molecular collisions played in reverse would obey the same laws. Yet the second law picks out a direction. Entropy increases toward the future, not the past.

Why?

The standard answer invokes cosmology. The universe started in a state of extremely low entropy. The Big Bang was, in a specific technical sense, highly ordered. From that special initial condition, entropy has been increasing ever since.

The arrow of time, on this view, is not built into the laws of physics. It is built into the initial conditions. We see entropy increase because we live in the aftermath of a very special beginning.

This is called the "past hypothesis," and it remains a topic of active research. We are honestly uncertain about the ultimate explanation.

The arrow of time may be set by cosmological initial conditions, not by local physics. This is the "past hypothesis"—a topic of ongoing research and honest uncertainty.

9.6 *Worked Examples with Numbers*

Let us ground these ideas in concrete calculations.

The Entropy of an Ideal Gas

What is the entropy of a mole of helium gas at room temperature and pressure?

For a monatomic ideal gas, the Sackur-Tetrode equation gives:

$$S = Nk_B \left[\ln \left(\frac{V}{N} \left(\frac{4\pi m E}{3Nh^2} \right)^{3/2} \right) + \frac{5}{2} \right]$$

where N is the number of atoms, V is volume, m is atomic mass, $E = \frac{3}{2} N k_B T$ is total thermal energy, and h is Planck's constant.

Helium at room temperature: about 126 J/K per mole, or equivalently about 30 bits of uncertainty per atom about its position and velocity.

For one mole of helium ($N = 6 \times 10^{23}$) at $T = 300$ K and standard pressure ($V = 24.5$ liters):

$$m_{\text{He}} = 4 \times 1.66 \times 10^{-27} \text{ kg} = 6.64 \times 10^{-27} \text{ kg}$$
$$E = \frac{3}{2} \times 6 \times 10^{23} \times 1.38 \times 10^{-23} \times 300 = 3740 \text{ J}$$

Working through the calculation:

$$S \approx 126 \text{ J/K per mole}$$

In bits per atom, using our conversion factor:

$$H = \frac{S}{k_B \ln 2 \times N} \approx \frac{126}{10^{-23} \times 0.693 \times 6 \times 10^{23}} \approx 30 \text{ bits per atom}$$

Each helium atom in the room carries about 30 bits of positional uncertainty. You would need approximately 30 yes/no questions to locate a single atom precisely.

Entropy Change in Heat Transfer

A hot object at $T_H = 400$ K and a cold object at $T_C = 300$ K exchange $Q = 100$ J of heat. What is the total entropy change?

The hot object loses entropy:

$$\Delta S_{\text{hot}} = -\frac{Q}{T_H} = -\frac{100}{400} = -0.25 \text{ J/K}$$

The cold object gains entropy:

$$\Delta S_{\text{cold}} = +\frac{Q}{T_C} = +\frac{100}{300} = +0.333 \text{ J/K}$$

The total entropy change:

$$\Delta S_{\text{total}} = -0.25 + 0.333 = +0.083 \text{ J/K} > 0$$

The second law is satisfied: total entropy increased.

In information terms:

$$\Delta H = \frac{\Delta S_{\text{total}}}{k_B \ln 2} \approx \frac{0.083}{10^{-23} \times 0.693} \approx 10^{21} \text{ bits}$$

Heat flowing from hot to cold creates entropy. In information terms, about 10^{21} bits of information are "lost"—we know less about where the energy came from.

About one sextillion bits of information were "lost" in this simple heat transfer. The universe became more uncertain about where the energy resides.

Free Expansion of a Gas

A gas occupies the left half of an insulated container. A partition is removed, and the gas expands to fill the entire container. What is the entropy change?

Before the expansion, each of the N molecules could be anywhere in the left half (volume $V/2$). After, each could be anywhere in the full container (volume V).

The number of microstates scales with the volume available to each molecule:

$$W_{\text{before}} \propto (V/2)^N$$
$$W_{\text{after}} \propto V^N$$

The entropy change:

$$\Delta S = k_B \ln \frac{W_{\text{after}}}{W_{\text{before}}} = k_B \ln \frac{V^N}{(V/2)^N} = k_B \ln 2^N = N k_B \ln 2$$

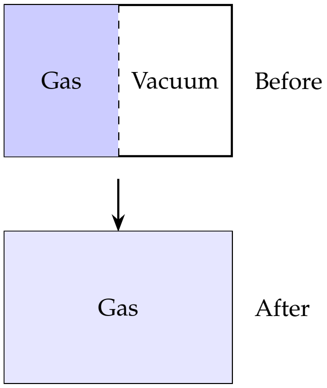


Figure 9.3: Free expansion. The gas expands from volume $V/2$ to V. Each molecule gains one bit of positional uncertainty.

Per molecule:

$$\Delta S_{\text{per molecule}} = k_B \ln 2$$

This is exactly one bit of uncertainty gained per molecule. Before the expansion, we knew each molecule was in the left half. After, we do not—it could be in either half with equal probability. One bit of information, lost.

The Gibbs Paradox

Consider two containers of different gases—say, helium and neon—each at volume V with n moles. A partition is removed, and they mix. What is the entropy change?

Each gas expands to fill volume $2V$:

$$\Delta S = 2 \times nR \ln 2$$

For one mole of each: $\Delta S = 2 \times 8.314 \times 0.693 \approx 11.5$ J/K.

But wait. What if the two gases are *identical*—both helium, say?

Our calculation would give the same answer: $\Delta S = 11.5$ J/K. But this seems wrong. If you cannot tell the atoms apart, is there any difference between "gas A atoms here, gas B atoms there" and "gas A atoms there, gas B atoms here"? The configurations are indistinguishable!

The Gibbs paradox: mixing identical gases should produce no entropy change. Gibbs resolved this by noting that indistinguishable particles must be counted differently—divide by $N!$.

Gibbs resolved this paradox by recognizing that for identical particles, we must divide W by $N!$ to avoid overcounting indistinguishable arrangements. When we do this correctly, mixing identical gases produces no entropy change.

This is surprisingly subtle. It shows that entropy depends not just on physical configuration, but on what we can *distinguish*. Information is in the eye of the beholder—or more precisely, in the capabilities of the observer.

9.7 *E.T. Jaynes and the Maximum Entropy Perspective*

In the 1950s, physicist Edwin Jaynes proposed a radical reinterpretation of statistical mechanics.

The traditional view asks: if atoms bounce around randomly, what probability distributions do we get?

Jaynes asked a different question: given that we know only certain macroscopic facts (like average energy), what probabilities *should* we assign?

His answer: maximize entropy subject to constraints. The Boltzmann distribution does not describe what atoms "actually do." It describes our best inference given limited information.

Jaynes: the Boltzmann distribution is not about what atoms "actually do"—it is about our best inference given limited information. Statistical mechanics is inference, not mechanics.

This is the *maximum entropy principle*: among all probability distributions consistent with what you know, choose the one with maximum entropy. Do not assume more than you know.

Why is this the right choice?

If you use a lower-entropy distribution, you are assuming information you do not have. If that assumed information is wrong, your predictions will be systematically biased. Maximum entropy is the unique distribution that:

1. Is consistent with known constraints

2. Makes no additional assumptions

On this view, the Boltzmann distribution emerges not from physics but from *logic*. Given that you know only $\langle E \rangle$ and nothing else, the Boltzmann distribution is your best guess.

This interpretation remains controversial. Many physicists feel that statistical mechanics describes real physical phenomena—heat baths, thermal fluctuations—not just our ignorance. Others find Jaynes' perspective illuminating, even liberating.

What everyone agrees on: the maximum entropy principle gives correct predictions. It correctly derives equilibrium distributions, explains phase transitions, and matches experiment. Whether this is because nature "maximizes entropy" or because maximum entropy is the correct inference from limited information is a matter of interpretation.

Whether Boltzmann statistics describes physics or inference is debated. What everyone agrees: the maximum entropy principle gives correct predictions.

9.8 Three Giants: Boltzmann, Gibbs, and Shannon

Before we close, let us pause for the history.

Ludwig Boltzmann (1844–1906)

Boltzmann was an Austrian physicist who spent his career defending the atomic hypothesis and developing statistical mechanics. His formula $S = k \ln W$ connected the microscopic world of atoms to the macroscopic world of heat engines.

He was combative, passionate, and frequently depressed. He fought bitter battles against Mach, Ostwald, and others who rejected atoms as metaphysical. His textbook on gas theory was so dense and difficult that few could read it.

In 1906, while on vacation with his family in Trieste, Boltzmann took his own life. He was sixty-two. Three years later, Jean Perrin's experiments on Brownian motion vindicated the atomic hypothesis beyond doubt. Boltzmann never knew.

His tombstone in Vienna bears his formula: $S = k \log W$.

J. Willard Gibbs (1839–1903)

Gibbs was Boltzmann's temperamental opposite—quiet, solitary, working in isolation at Yale. He never married, lived with his sister, and published his work in obscure American journals that few Europeans read.

His 1878 paper "On the Equilibrium of Heterogeneous Substances" developed much of what we now call thermodynamics and statistical mechanics. It was 300 pages long and took years to be appreciated. His 1902 book *Elementary Principles in Statistical Mechanics* introduced the ensemble approach and the generalized entropy formula.

Gibbs worked in isolation at Yale. His 1878 paper on thermodynamics was 300 pages long and took years to be appreciated. Much of his work had to be rediscovered by others.

Gibbs' work was so far ahead of its time that many concepts had to be rediscovered by others who received the credit.

Claude Shannon (1916–2001)

Shannon was an engineer at Bell Labs who, in one paper in 1948, created information theory. He chose the name "entropy" for his uncertainty measure deliberately—reportedly on the advice of John von Neumann.

The story goes that von Neumann told Shannon: "You should call it entropy, for two reasons. First, the function already appears in statistical mechanics under that name. Second, and more importantly, nobody really knows what entropy is, so in a debate you will always have the advantage."

Von Neumann's advice to Shannon: "Call it entropy. Nobody knows what entropy is, so in a debate you will always have the advantage." Whether apocryphal or not, the name was well chosen.

Whether the anecdote is true, the name was perfect. Shannon knew about the connection—in his 1948 paper, he noted that his formula "will be recognized as that of entropy as defined in certain formulations of statistical mechanics." But he was careful to motivate it independently, from communication theory.

Three people, separated by a century and by the Atlantic Ocean, discovered the same formula. Boltzmann found it in gas molecules. Gibbs found it in statistical ensembles. Shannon found it in telegraph messages. The formula was waiting to be found.

9.9 *What Does This Unity Mean?*

We have shown that Shannon entropy and thermodynamic entropy are mathematically identical. But *why*? Why should a formula invented for telegraph engineering be the same as a formula invented for steam engines?

Possible Interpretations

Interpretation 1: Mathematical necessity.

Both Shannon and Boltzmann needed a function satisfying certain requirements—additivity for independent systems, non-negativity, maximality for uniform distributions. The requirements uniquely determine the form. This is the "mathematical coincidence" view: they happen to need the same tool.

Interpretation 2: Both measure uncertainty.

Both entropies measure how spread out a probability distribution is. Shannon measures uncertainty about messages; Boltzmann measures uncertainty about microstates. Same concept, same formula.

Interpretation 3: Information is physical.

This is the deepest view, associated with John Wheeler's slogan "it from bit." Physical systems carry information in the positions of atoms, the spins of electrons, the polarizations of photons. The laws of physics are, in some sense, laws about how information transforms.

Is information fundamental to physics? Wheeler's "it from bit" suggests the universe may be, at its deepest level, an information-processing system. We do not know.

On this view, the universe is not just described by information—it *is* information. Reality may be, at its deepest level, computational.

We do not know which interpretation is correct. Perhaps it does not matter—the mathematics works regardless. But the question haunts physicists and philosophers alike: is information an abstraction we impose on the world, or is the world made of information?

Practical Implications

Whatever the philosophical interpretation, the practical implications are clear:

1. Information processing has physical costs. Erasing a bit dissipates at least $k_B T \ln 2$ of energy. This is Landauer's principle.

2. Physical systems have information-theoretic limits. Channel capacity applies to neural signals, not just fiber optics. The brain is constrained by the same theorems as the internet.

3. The second law constrains computation. You cannot compute for free. Irreversible computation produces heat.

The unity of the two entropies is not just beautiful—it is useful. It tells us that information theory and thermodynamics are not separate subjects. They are the same subject, viewed from different angles.

9.10 The Demon Awaits

Let us take stock.

We began with two entropies that seemed unrelated—one from telegraphy, one from thermodynamics. We showed they are the same formula: $S = k_B \ln 2 \times H$. We derived the Boltzmann distribution from maximum entropy. We reinterpreted the second law as information loss.

The universe's tendency toward disorder is not some mysterious force—it is combinatorics. There are more ways to be spread out than concentrated, more ways to be uncertain than certain, more ways to have forgotten than to have remembered.

But this raises a troubling question. If the second law is just about losing information, what if we do not lose it? What if we track every molecule perfectly? Could we then reverse entropy? Could we violate the second law?

The second law says entropy increases. But what if we track every molecule? What if we refuse to lose information? Can we then reverse entropy? Maxwell imagined a demon who could try.

This question was posed by James Clerk Maxwell in 1867. He imagined a tiny intelligent being—a "demon"—who could observe individual molecules and sort them by speed. Fast molecules to one side, slow to the other. Temperature differences from nothing. Heat engines running on knowledge alone. The second law, apparently violated.

For over a century, physicists debated whether Maxwell's demon could exist. The resolution, when it came, was startling: the demon cannot violate the second law, because the demon itself must be physical. The act of observing and remembering molecule speeds generates entropy. When the demon erases its memory to make room for new observations, it must pay a thermodynamic cost.

Information erasure costs energy. This is Landauer's principle, and it is the subject of our next chapter.

The demon is exorcised not by showing it cannot know, but by showing that knowledge itself has physical consequences.

Historical note. Shannon knew exactly what he was doing when he named his quantity "entropy." In a 1956 interview, he said: "My greatest concern was what to call it. I thought of calling it 'information,' but the word was overly used, so I decided to call it 'uncertainty.' When I discussed it with John von Neumann, he had a better idea. Von Neumann told me, 'You should call it entropy, for two reasons: In the first place your uncertainty function has been used in statistical mechanics under that name, so it already has a name. In the second place, and more important, nobody knows what entropy really is, so in a debate you will always have the advantage.' " Whether von Neumann actually said this, Shannon delighted in repeating the story. The name stuck, and so did the connection.

10
Maxwell's Demon and the Cost of Computation

In 1867, James Clerk Maxwell—the same Maxwell who unified electricity and magnetism, who showed that light was an electromagnetic wave, who ranks alongside Newton and Einstein in the pantheon of physics—imagined a creature so small it could see individual molecules. This creature, sitting by a tiny door between two chambers of gas, would watch molecules approach and open the door only for fast molecules going one way, slow molecules going the other. No work done. Just watching and opening a door.

Maxwell introduced his demon in a letter to Peter Tait in 1867. He called it "a very observant and neat-fingered being." The name "demon" came later.

If this worked, the demon could create a temperature difference from uniform gas—hot on one side, cold on the other—without expending energy. From that temperature difference, you could run a heat engine indefinitely. The second law of thermodynamics—the law that entropy always increases, that heat flows from hot to cold, that perpetual motion is impossible—would be violated.

For over a century, the best physicists in the world attacked this problem. Some said the demon could not see molecules without disturbing them. Others said opening the door required energy. Each proposed solution had flaws. The demon kept escaping.

The answer, when it finally came, was startling. It came not from mechanics but from information theory. The demon must *remember* which molecules it let through. Its memory fills up. And when it erases that memory to make room for more observations—*there* is the cost. Not in seeing, not in sorting, but in forgetting.

The resolution of Maxwell's demon required understanding that information is physical. Erasing information has a thermodynamic cost.

This is the story of how a tiny imaginary creature threatened to overthrow thermodynamics and was finally defeated—not by physics forbidding it to act, but by thermodynamics of thought. The universe does not care what you do; it cares what you forget.

10.1 A Very Observant Being

Let us be precise about what Maxwell's demon does.

The Setup

Imagine a box of gas in thermal equilibrium. The molecules dart about with a range of speeds—some fast, some slow—distributed according to the Maxwell-Boltzmann distribution. The average kinetic energy is the same everywhere; the temperature is uniform.

In equilibrium, molecular speeds follow the Maxwell-Boltzmann distribution. The temperature is uniform because fast and slow molecules are mixed everywhere.

Now place a partition in the middle of the box, with a small door. Station a demon—an intelligent being small enough to see individual molecules—next to this door.

The demon's protocol is simple:

1. Watch molecules approaching the door from either side
2. Note their velocity
3. If a fast molecule approaches from the left, open the door and let it pass right
4. If a slow molecule approaches from the right, open the door and let it pass left
5. Otherwise, keep the door closed

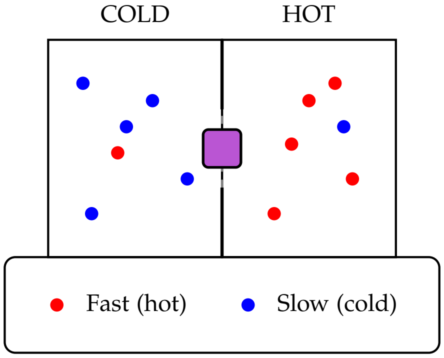


Figure 10.1: Maxwell's demon sorts molecules by speed. Fast molecules (red) accumulate on the right; slow molecules (blue) accumulate on the left.

The door is massless and frictionless; opening it requires no work. The demon does not push molecules; it only decides when to open the door.

After many sortings, the right chamber contains mostly fast molecules (hot), and the left chamber contains mostly slow molecules (cold). A temperature difference has appeared from nowhere.

Why This Threatens Thermodynamics

The second law of thermodynamics, in one formulation, states: heat does not spontaneously flow from a cold body to a hot body. Equivalently: the entropy of an isolated system never decreases.

What has the demon achieved? Starting from thermal equilibrium—maximum entropy for a given total energy—it has created a non-equilibrium state. The gas on the right is hotter; the gas on the left is colder. This is a state of lower entropy.

From a temperature difference, you can run a heat engine. The demon seems to offer perpetual motion of the second kind.

From this temperature difference, you could run a Carnot engine. Extract work. The engine eventually re-equilibrates the gas. Let the demon sort again. Repeat forever. This would be perpetual motion of the second kind—not creating energy from nothing, but converting thermal energy to work with no limit, violating the second law.

Maxwell himself suspected the issue involved knowledge and uncertainty. He noted that the demon violated no mechanical law—every collision is perfectly reversible. The demon simply exploits information to extract work.

A Century of Failed Solutions

Physicists attacked the demon for over a century. Let us examine the main attempts.

The door requires energy to operate. This was the first objection. But in principle, a door can be made arbitrarily light and frictionless. With careful engineering, the energy to open the door can be made negligible.

The demon cannot see molecules without disturbing them. Léon Brillouin pursued this line in 1951. He argued that to detect a molecule, the demon must shine light on it. The photon carries energy and entropy. But consider: a gas in equilibrium already glows with blackbody radiation. The demon could use this ambient light to see molecules without adding anything. The measurement can, in principle, be done without energy cost.

Measurement must increase entropy. This seemed promising. If acquiring information about a molecule increases entropy by at least as much as the sorting decreases it, the second law is saved. But this runs into trouble: it is possible to measure and record information reversibly, without entropy increase. Simply correlating the demon's memory with the molecule's position is a reversible operation.

Brillouin and others focused on the measurement process. But measurement can be done reversibly—the cost is elsewhere.

Each solution had flaws. The demon kept slipping through.

The key was asking the right question. For a century, physicists asked: "Where does the demon expend energy?" The answer, perhaps surprisingly, is: nowhere during sorting. The question should have been: "Where does information get destroyed?"

10.2 Szilard's Engine and the Bit

In 1929, Hungarian physicist Leo Szilard made the crucial step. He stripped Maxwell's demon to its essence: one molecule, one bit of information.

The One-Molecule Engine

Szilard's setup is elegant in its simplicity.

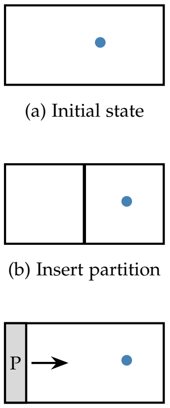
Consider a box containing a single gas molecule in thermal equilibrium with a heat bath at temperature T. The molecule bounces around randomly, colliding with the walls.

Now perform the following cycle:

1. Insert a partition in the middle of the box. The molecule is now trapped in either the left half or the right half.

2. Observe which side the molecule is on. This is one bit of information.

3. Attach a piston to the *empty* side. Remove the partition.



(a) Initial state

(b) Insert partition

(c) Piston on empty side

Figure 10.2: Szilard's one-molecule engine. (a) A single molecule in a box. (b) A partition divides the box. (c) After observing which side contains the molecule, attach a piston to the empty side. The molecule expands isothermally, doing work.

4. The molecule, bouncing around in its half of the box, pushes against the piston. Allow isothermal expansion back to the full volume, extracting work.

How much work is extracted? The expansion is isothermal—heat flows in from the bath to maintain constant temperature—from volume $V/2$ to volume V. The work done is:

$$W = \int_{V/2}^{V} P\,dV = \int_{V/2}^{V} \frac{k_B T}{V}\,dV = k_B T \ln \frac{V}{V/2} = k_B T \ln 2$$

At room temperature ($T = 300$ K), this is:

$$W = (1.38 \times 10^{-23}\text{ J/K})(300\text{ K})(0.693) \approx 2.9 \times 10^{-21}\text{ J}$$

Szilard's engine extracts $k_B T \ln 2$ of work per cycle—exactly the energy equivalent of one bit of information at temperature T.

This is a tiny amount of energy—about 0.018 electron volts. But it is not zero. We extracted $k_B T \ln 2$ of work from thermal equilibrium by knowing one bit of information.

Where Does the Entropy Go?

Szilard correctly identified the puzzle. Before observation, the molecule could be on either side—entropy associated with one bit of uncertainty. After observation, we know exactly where it is—zero uncertainty. The demon's knowledge enabled work extraction.

Szilard proposed that measurement itself must cost at least $k_B T \ln 2$ of energy, exactly compensating the work extracted. This would preserve the second law.

But Szilard was not quite right. There is no fundamental reason measurement must cost energy. Here is why.

Szilard was right that information is central, but he located the cost in the wrong place. Measurement can be reversible; erasure cannot.

Consider what measurement actually does. Before measurement, the molecule is in an unknown position, and the demon's memory is in a "blank" state. After measurement, the molecule is in the same position, and the demon's memory records that position. Information has been *copied* from the molecule to the memory.

Copying can be done reversibly. Given the final state (memory says "right," molecule is on right), you can uniquely determine the initial state (memory was blank, molecule was on right). No information is lost; entropy need not increase.

So measurement is not the problem. But then where is the cost?

You might ask: "If measurement is free and extracting work is free, where is the problem?"

The problem is that the demon's memory is now full. To repeat the cycle, it must clear that memory. And *that* is where the demon meets its doom.

10.3 Landauer's Principle: The Cost of Forgetting

In 1961, Rolf Landauer of IBM published a remarkable paper: "Irreversibility and Heat Generation in the Computing Process." His key insight was deceptively simple: *erasing information has an irreducible thermodynamic cost.*

What Erasure Means

Let us be precise. Erasure is the process of resetting a memory element to a standard state, regardless of what state it was in.

A bit can be 0 or 1. Erasing it means forcing it to 0 (or 1, your choice of convention), no matter what it was before. The final state does not depend on the initial state.

This is *logically irreversible*. Given the output (0), you cannot determine the input (was it 0 or 1?). Information has been destroyed.

Contrast with copying: if I copy bit A to bit B, the final state (both bits equal to A) tells me what A was. Copying preserves information. Erasure destroys it.

Erasure is logically irreversible: knowing the output, you cannot determine the input. This distinguishes it from copying, which preserves information.

The Physical Argument

Consider a physical system representing one bit. A natural example: a particle in a double-well potential. The particle in the left well represents 0; the particle in the right well represents 1.

Before erasure, the particle could be in either well. In the language of statistical mechanics, the accessible phase space has volume proportional to 2 (two possible configurations).

After erasure, the particle is definitely in the left well. The accessible phase space has volume proportional to 1.

The phase space has shrunk by a factor of 2. But here is the problem: Liouville's theorem states that phase space volume is conserved in Hamiltonian dynamics. You cannot squeeze the system into a smaller region of phase space by any dynamical process.

How can it shrink?

The resolution: the particle's phase space shrinks, but the environment's phase space must expand to compensate. The system is coupled to a heat bath. When we force the particle into one well (regardless of where it started), we must dump entropy into the environment.

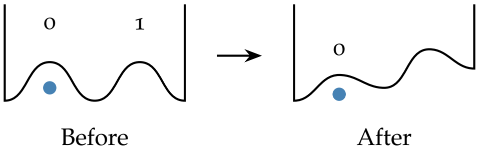


Figure 10.3: Erasure compresses phase space. Before: the particle could be in either well. After: it is definitely in the left well.

Liouville's theorem: phase space volume is conserved. If the system's phase space shrinks, something else must expand to compensate. That something is the environment.

The Derivation

Let us derive Landauer's limit carefully.

Initial state: A bit in state 0 or 1 with equal probability. The Shannon entropy is 1 bit.

In thermodynamic terms, the system entropy is:

$$S_{\text{initial}} = k_B \ln 2$$

Final state: The bit is in state 0 with certainty. The Shannon entropy is 0 bits.

$$S_{\text{final}} = k_B \ln 1 = 0$$

Change in system entropy:

$$\Delta S_{\text{system}} = S_{\text{final}} - S_{\text{initial}} = -k_B \ln 2$$

The second law: For any process, the total entropy change of system plus environment must be non-negative:

$$\Delta S_{\text{total}} = \Delta S_{\text{system}} + \Delta S_{\text{environment}} \geq 0$$

Therefore:

$$\Delta S_{\text{environment}} \geq k_B \ln 2$$

If the environment is a heat bath at temperature T, the minimum heat dissipated is:

$$Q = T \times \Delta S_{\text{environment}} \geq k_B T \ln 2$$

This is Landauer's principle: *erasing one bit of information costs at least* $k_B T \ln 2$ *of energy, dissipated as heat to the environment.*

Landauer's limit: erasing one bit costs at least $k_B T \ln 2$ of energy, dissipated as heat. This is thermodynamics, not engineering.

The Number

Let us compute the Landauer limit at room temperature.

At $T = 300$ K:

$$k_B T \ln 2 = (1.38 \times 10^{-23}\ \text{J/K})(300\ \text{K})(0.693) \approx 2.9 \times 10^{-21}\ \text{J}$$

In electron-volts, this is about 0.018 eV—18 milli-electron-volts.

This is extraordinarily small. For comparison, visible light photons carry about 2 eV—a hundred times more. The energy released when an ATP molecule is hydrolyzed is about 0.5 eV—thirty times more. A single bit of information at room temperature is energetically humble.

At room temperature, the Landauer limit is about 3×10^{-21} joules per bit. Current computers dissipate roughly a trillion times this amount.

Yet the limit is real. It is not a matter of better engineering. It is thermodynamics. You cannot do better.

Why Erasure Is Fundamental

Landauer's deeper point: erasure is the *only* logically irreversible operation that fundamentally requires energy dissipation. All other computational operations can, in principle, be done reversibly.

This includes:

- Copying data (reversible: knowing input and output, you can recover the original)
- Any invertible function (reversible by construction)
- Measurement (reversible correlation between system and memory)
- Logic gates like NOT (reversible: apply NOT again to recover input)

Only erasure destroys information. Only destroying information costs energy.

There is something profound here. Logically irreversible operations—where information is lost—are the ones that cost energy. Logic and thermodynamics are intertwined. The universe keeps track of information, and charges you for losing it.

10.4 Bennett's Resolution: Exorcising the Demon

The pieces were in place, but the connection to Maxwell's demon was not yet complete. It fell to Charles Bennett of IBM to close the loop in 1982.

The Complete Cycle

Previous analyses had examined single measurements or single erasures. Bennett asked: what happens when the demon operates *cyclically*?

To violate the second law, the demon must extract work cycle after cycle, indefinitely. Each cycle, it must:

Bennett's key insight: a demon that operates cyclically must return to its initial state—including its memory. This requires erasure.

1. Start with a blank memory
2. Observe a molecule and record its speed (memory now holds 1 bit)
3. Use this information to sort the molecule and extract work
4. End with... what?

The demon's memory now contains a record of the observation. To repeat the process, it must clear this memory. Otherwise, after N cycles, it holds N bits of data—the speeds of all molecules it has sorted.

But clearing the memory is erasure. And erasure, by Landauer's principle, costs at least $k_B T \ln 2$ per bit.

The Balance Sheet

Let us tally the demon's accounts for one complete cycle.

Work extracted: From Szilard's analysis, the demon extracts $+k_B T \ln 2$ by using its knowledge of the molecule's position.

Cost of observation: Zero. Measurement can be done reversibly.

Cost of erasure: The demon must erase 1 bit of memory, costing at least $k_B T \ln 2$.

Net energy gain:

$$\Delta E = k_B T \ln 2 - k_B T \ln 2 = 0$$

The demon cannot profit. The work it extracts is exactly paid back when it clears its memory. The second law is saved.

The demon's net energy extraction is zero. The work gained from sorting equals the energy lost to erasure. The second law survives.

Can the Demon Avoid Erasing?

You might try to save the demon with clever accounting.

Option 1: Keep all memories forever. The demon never erases—it just accumulates data. But then its memory grows without bound. After N cycles, it holds N bits. Eventually, it runs out of storage. This is not cyclic operation; it is a one-time extraction that eventually stops.

Option 2: Use an infinite memory tape. Extend the memory indefinitely. Now the demon can run forever. But the filled tape has lower entropy than the blank tape—it contains N bits of information. If you account for the tape as part of the system, total entropy has still increased. The demon has not violated the second law; it has merely deferred the accounting.

Option 3: Dump memories into the environment. Let the demon write its observations onto the thermal environment. But a system in thermal equilibrium is already at maximum entropy. Adding structured information to it requires work—you are lowering its entropy momentarily, which costs energy. We are back to paying $k_B T \ln 2$ per bit.

Every escape route leads back to Landauer's limit.

The Resolution in One Paragraph

Maxwell's demon can indeed observe molecules, sort them, and extract work. No law prevents this. But the demon must record what it observes—otherwise, how does it know whether to open the door? These records fill its memory. To operate cyclically, it must erase old records. Erasure, by Landauer's principle, requires dissipating $k_B T \ln 2$ per bit to the environment. This dissipation exactly compensates the work extracted. The second law is not violated; it is enforced through information.

The demon is defeated not by physics forbidding it to see or act, but by the thermodynamics of memory. Knowledge has physical consequences.

The demon is defeated not by being forbidden to see or act, but by the thermodynamics of thought. It can observe freely; it can sort freely; it can extract work. But it cannot forget for free. The universe charges for forgetting.

10.5 Reversible Computation

Landauer's principle raises a startling question: if erasure costs energy, and computers constantly overwrite data, how much energy must computation consume?

The Puzzle

Modern computers erase bits constantly. Every time a transistor switches, the old value is overwritten. Every logical AND or OR destroys information (knowing the output, you cannot always recover the inputs). By Landauer's principle, each such operation should cost at least $k_B T \ln 2$.

A modern processor executes roughly 10^{10} operations per second. At the Landauer limit:

Modern processors dissipate about a trillion times more energy per operation than the Landauer limit. There is room for improvement—but also fundamental floors.

$$P_{\text{min}} = 10^{10} \times 3 \times 10^{-21}\ \text{J} = 3 \times 10^{-11}\ \text{W} = 30\ \text{pW}$$

But modern processors consume about 100 watts—roughly 10^{12} times the Landauer limit. We are nowhere near the thermodynamic floor.

Does this mean we can improve by a factor of a trillion? Or is there something more fundamental limiting us?

Computing Without Erasing

In 1973, Bennett made a remarkable discovery: computation need not involve erasure at all.

Bennett showed that any computation can be performed reversibly, without erasing information. Reversible computation generates no fundamental heat.

The key idea: what if we compute while keeping track of everything?

An ordinary AND gate destroys information. $\text{AND}(1,1) = 1$, but so does... wait, no: $\text{AND}(0,1) = 0$ and $\text{AND}(1,0) = 0$. Given output 0, we cannot tell which inputs produced it. Two bits in, one bit out—information lost.

But we can make computation reversible by keeping the inputs alongside the output. Instead of computing $f(x)$, compute $F(x,0) = (x, f(x))$. The input x is preserved. Given the output $(x, f(x))$, we can uniquely recover the input $(x, 0)$.

The Toffoli Gate

Tommaso Toffoli showed in 1980 that any computation can be built from a single reversible gate, now called the Toffoli gate.

The Toffoli gate has three input bits (a, b, c) and three output bits

(a', b', c'):

$$a' = a$$
$$b' = b$$
$$c' = c \oplus (a \wedge b)$$

Here $\oplus$ is XOR (exclusive or) and $\wedge$ is AND.

The key property: the Toffoli gate is its own inverse. Apply it twice, and you return to the original input. Three bits in, three bits out—no information lost.

Yet it is universal: any Boolean function can be computed using only Toffoli gates, with appropriate ancilla bits.

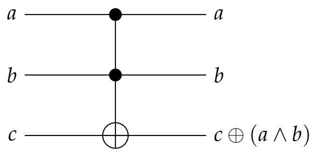


Figure 10.4: The Toffoli gate. The top two bits control whether the bottom bit is flipped. The operation is its own inverse.

The Garbage Problem

There is a catch. Reversible computation generates "garbage"—intermediate results that are not needed for the final output but must be kept to maintain reversibility.

Reversible computation generates garbage bits. These must be uncomputed before they can be reused, adding overhead but avoiding erasure.

Bennett's solution: compute forward, copy the answer to a safe location, then compute *backward* to uncompute the garbage.

The workflow:

1. Start with input x and blank workspace
2. Compute forward: produce output $f(x)$ and garbage $g(x)$
3. Copy output $f(x)$ to a fresh register (reversible)
4. Compute backward: uncompute the forward calculation, restoring the workspace to blank
5. Result: input x, output $f(x)$, blank workspace

We have computed $f(x)$ reversibly. The only irreversible step is the final copying of the answer to where we want it—and that is just one erasure for the whole computation, not one per gate.

The Cost of Reversible Computation

For a perfectly reversible computer, the energy cost comes only from:

- The final output (must either keep it or pay to erase it)
- Error correction (random errors must be erased to maintain coherence)
- The answer itself (if you want to reuse the output register, you pay $k_B T \ln 2$ per bit)

The computation itself—all the intermediate steps—can be done without dissipating energy. You pay only for what you ultimately throw away.

In principle, computation itself can be free. You pay only for the information you ultimately discard.

You might object: "But we always need to read out and use the answer!" True. But reading out costs only one bit-erasure per answer bit. The work of the computation itself can, in principle, be free.

Why Reversible Computing Matters

Today, reversible computing is mostly a theoretical curiosity. Building reversible circuits is hard. They run slowly (must operate near-adiabatically to avoid dissipation). Current technology is so far from Landauer's limit that the engineering gains from irreversible computing vastly outweigh the thermodynamic costs.

But this will not always be true. As transistors shrink toward atomic scales, the thermodynamic limits become visible. Quantum computers, which operate with unitary (hence reversible) gates, may approach these limits. Understanding reversible computation may guide the computer architectures of the future.

10.6 The Demon in the Laboratory

For fifty years after Landauer's paper, his principle remained theoretical—a beautiful argument, but untested. Measuring $k_B T \ln 2$ joules is not easy. At room temperature, this is 3×10^{-21} joules—about the energy of a molecule jiggling once.

Then, in 2012, the demon was finally caught in the laboratory.

Landauer's principle remained untested until 2012, when experiments with colloidal particles in optical traps confirmed the limit.

Measuring Single-Bit Erasure

Antoine Bérut and colleagues at the École Normale Supérieure de Lyon performed a landmark experiment. Their "bit" was a tiny silica bead, about 2 micrometers in diameter, suspended in water and held by an optical trap—a tightly focused laser beam that creates a potential well.

By adjusting the laser, they could create a double-well potential: the bead could sit in either the left well (state 0) or the right well (state 1). Erasure meant tilting the potential to force the bead into one well, regardless of where it started.

By tracking the bead's position with nanometer precision, they could measure the work done on it and the heat dissipated to the surrounding water.

Their key finding: when erasure was done slowly and gently, the heat dissipated approached $k_B T \ln 2$ from above. Fast erasure dissipated

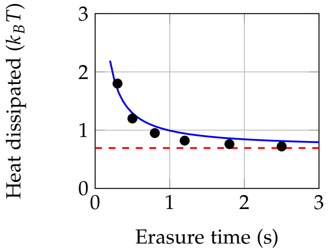


Figure 10.5: Heat dissipated versus erasure time. Fast erasure dissipates extra heat. Slow erasure approaches $k_B T \ln 2$.

more heat—extra friction from the rapid motion. But no matter how cleverly they designed the protocol, they could not go below $k_B T \ln 2$.

The Landauer limit is real.

Information to Energy Conversion

A year earlier, Shoichi Toyabe and colleagues in Tokyo had demonstrated the reverse process: converting information into energy.

Their setup: a colloidal particle on a staircase-like potential, able to climb against gravity. A feedback system observed the particle's position and adjusted the potential accordingly. When the particle fluctuated upward, the system would raise a barrier behind it, preventing it from falling back.

Toyabe's experiment demonstrated information-to-energy conversion. By observing the particle and exploiting thermal fluctuations, they extracted work against gravity.

Step by step, driven by thermal fluctuations and locked in by feedback, the particle climbed the staircase. It gained potential energy. Where did this energy come from? From the heat bath, guided by information.

The particle extracted work from thermal equilibrium—exactly as Szilard's engine does. The maximum energy extracted was $k_B T$ times the information gained by measurement.

Maxwell's demon, operating in a real physical system.

What the Experiments Confirm

These experiments confirm four things:

1. Erasure dissipates heat
2. The minimum dissipation is $k_B T \ln 2$ per bit
3. Information can be converted to work at the same rate
4. The second law holds when information is properly accounted

These experiments show that information is physical—as real as energy, as measurable as temperature. The demon's defeat is laboratory fact.

The experiments do not merely verify a formula. They show that information is physical—as real as energy, as measurable as temperature. The demon's defeat is not a mathematical trick; it is laboratory fact.

10.7 Implications for the Future of Computing

What does all this mean for computers?

The Current State

Modern processors consume about 100 watts while executing 10^{10} operations per second. That is roughly 10^{-8} joules per operation.

The Landauer limit at room temperature is about 3×10^{-21} joules per bit erasure.

The ratio:

$$\frac{10^{-8}}{3 \times 10^{-21}} \approx 3 \times 10^{12}$$

Current computers dissipate roughly a trillion times more energy per operation than the fundamental limit. Where does this energy go?

Current computers dissipate about a trillion times the Landauer limit. Most of this is engineering overhead, not fundamental physics.

Most of it is engineering overhead: charging and discharging transistor gates (capacitive energy), current leaking through "off" transistors, resistance in wires. These are not fundamental physics—they are artifacts of how we build computers.

The End of Dennard Scaling

For decades, computing benefited from Dennard scaling: as transistors shrank, voltage dropped proportionally, so power per transistor dropped even as speed increased. Every generation of chips was faster and cooler.

Around 2006, this stopped. Voltage could not drop further without transistors becoming unreliable—thermal noise would cause errors. Power density stopped falling. The multicore era began: spread computation across space rather than speed it up in time.

As devices continue to shrink, we approach not just engineering limits but physical ones. The Landauer limit, once academic, becomes visible on the horizon.

Dennard scaling ended around 2006. Further improvements require new approaches. Reversible and quantum computing may eventually approach fundamental limits.

Approaching the Limit

Several technologies might approach the Landauer limit:

Adiabatic computing: Charge capacitors slowly, recovering the energy on discharge. This reduces dissipation below the capacitive limit, though at the cost of speed.

Reversible logic: As discussed, computation can in principle avoid erasure entirely. This requires new architectures and is slower, but there is no fundamental barrier.

Quantum computing: Quantum gates are unitary, hence reversible. The computation itself need not dissipate energy. Errors and final measurement still cost, but these are per-answer, not per-operation.

None of these is practical today. But the gap between current practice and fundamental limits—a factor of 10^{12}—is enormous. There is room to explore.

The Ultimate Laptop

How fast could a computer possibly compute?

Seth Lloyd of MIT estimated the ultimate physical limits in 2000. Take one kilogram of matter. How many operations can it perform?

Energy limits: Using $E = mc^2$, one kilogram has 9×10^{16} joules of rest mass energy. At the Landauer limit, this allows about 3×10^{37} bit erasures before the matter is exhausted.

The ultimate physical computer: about 10^{40} operations per kilogram-second. We are far from this limit, but it exists.

Speed limits: Quantum mechanics limits how fast a system can evolve between distinguishable states (the Margolus-Levitin theorem). For a system with energy E, the maximum rate is about $E/\hbar$ operations per second.

The result: about 10^{40} operations per kilogram-second as an ultimate limit.

This is a staggering number, but it is finite. Even the ultimate computer is constrained by information thermodynamics.

10.8 The Century-Long Debate

Let us pause to appreciate the history of this puzzle.

1867: Maxwell introduces the demon in a letter to Peter Tait. He calls it "a very observant and neat-fingered being." The name "demon" came from William Thomson (Lord Kelvin).

1871: Maxwell publishes the thought experiment in *Theory of Heat*. The demon becomes famous—and controversial.

1905–1930: Multiple physicists attempt to resolve the paradox. Most focus on the mechanics of observation.

1929: Leo Szilard publishes his engine, connecting information to thermodynamics. He proposes that measurement costs energy.

The resolution of Maxwell's demon took 115 years: from Maxwell's letter in 1867 to Bennett's resolution in 1982.

1951: Léon Brillouin analyzes measurement, arguing that acquiring information costs at least $k_B T \ln 2$. His analysis has gaps.

1961: Rolf Landauer publishes "Irreversibility and Heat Generation in the Computing Process." He shows that erasure—not measurement—is the irreversible step. But he does not explicitly resolve the demon.

1973: Charles Bennett demonstrates universal reversible computation. Computation need not dissipate energy.

1982: Bennett finally exorcises the demon. The complete cycle requires erasure, which pays back the extracted work.

2010: Toyabe et al. demonstrate information-to-energy conversion experimentally.

2012: Bérut et al. measure the Landauer limit directly.

One hundred fifteen years from Maxwell's letter to Bennett's resolution. The delay came not from lack of cleverness but from asking the wrong question. Each generation focused on measurement, when the key was memory.

10.9 Information and Reality

What does the demon teach us?

Information Is Physical

Before Landauer, information seemed abstract—patterns, symbols, ideas. After Landauer, information is concrete. It takes space (in memory). It costs energy (to erase). It interacts with thermodynamics.

This is not metaphor. A bit of information at temperature T has thermodynamic entropy $k_B \ln 2$. Erasing it requires dissipating energy $k_B T \ln 2$. These are measurable quantities with physical dimensions.

Information is not merely abstract. It is physical—embodied in matter, subject to thermodynamics, constrained by physics.

Irreversibility Is About Information

The second law says entropy increases. But what is entropy increase, really?

We now have an answer: entropy increase is information loss. When we say a process is irreversible, we mean we cannot recover the initial state from the final state. Information about the past has been erased.

The arrow of time—why entropy increases toward the future—connects to information. The past is what we can remember; the future is what we cannot yet know. Memory requires physical records; records require stability; erasing records costs energy. The thermodynamics of memory may be the thermodynamics of time itself.

Computation Is Physical

Thinking—whether by brain or by machine—transforms information. That transformation has physical constraints. Not engineering constraints that might be overcome with cleverness, but fundamental constraints rooted in thermodynamics.

Thinking—whether by brain or machine—is a physical process. The thermodynamics of information constrains the thermodynamics of thought.

Your brain, running on about 20 watts, processes information at some rate. That processing dissipates heat. Some of that heat is fundamental—the cost of erasing memories, making decisions, forgetting details. The thermodynamics of thought is real.

The Universe as Computer?

John Wheeler proposed the slogan "it from bit"—that physical reality emerges from information. On this view, the universe is not just described by information; it *is* information. The laws of physics are laws about how information transforms.

We do not know if this is true. But we do know that information and physics are intertwined at the deepest level. The demon, meant as a paradox about heat, turned out to be a window into the nature of thought and reality.

10.10 From Demons to Inference

Let us take stock.

We began with a creature that threatened thermodynamics. For a century, the best physicists could not defeat it. The resolution came from an unexpected direction: the thermodynamics of memory.

Maxwell's demon is defeated by the cost of forgetting. This connects physics, computation, and information at the deepest level.

The demon can observe molecules freely. It can sort them freely. It can extract work. But it cannot forget for free. Erasing one bit of memory requires dissipating $k_B T \ln 2$ of energy. This is Landauer's principle, verified in the laboratory, as fundamental as any law of physics.

Reversible computation shows that erasure is the only irreducible cost. Computation itself can, in principle, be free. Only forgetting costs.

The implications ripple outward: to the future of computing, to the nature of time, to the meaning of information itself.

But there is another direction to explore. Information theory reaches into physics through Landauer's principle. Does it reach equally into statistics and reasoning? When we observe evidence and update our beliefs, we are performing a computation. Our prior becomes our posterior. Information flows from observation to conclusion.

Chapter 11 turns from physics to inference. Shannon's entropy appears naturally in Bayesian reasoning—the expected information gain is mutual information.

It turns out that Shannon's entropy appears naturally in Bayesian inference. The expected information gain from an observation is precisely the mutual information between observation and hypothesis. The demon gave us physics. Now let us turn to thought.

The demon was supposed to threaten thermodynamics. Instead, it revealed how deeply information is woven into the physical world. Let us see how deeply it is woven into the process of reasoning itself.

Historical note. Maxwell never intended his demon to be taken literally. In his original letter to Tait, he wrote: "if we conceive a being whose faculties are so sharpened that he can follow every molecule in its course, such a being, whose attributes are still as essentially finite as our own, would be able to do what is at present impossible to us." He was probing the foundations of the second law, asking whether it was truly a law of nature or merely a statement about human limitations. The answer—that it is both—took a century to clarify. The demon taught us that physics and knowledge are not separate domains. What we know affects what we can do. And what we forget has consequences.

11
Information and Inference

Suppose you are investigating a weighted die. Someone has tampered with it, but you do not know how. You get to roll it ten times. After those ten rolls, you know more than you did before. But how much more?

The question "how much did I learn?" has a precise answer. It is given by mutual information—the same quantity that governs communication channels.

This is not a vague question. We have been developing, through eleven chapters now, the machinery to answer it precisely. We have seen entropy measure uncertainty, mutual information quantify what one thing tells us about another, and the second law of thermodynamics enforce the cost of forgetting. Now we turn these tools to the problem of learning itself.

The surprise—though by now perhaps it should not surprise us—is that Shannon's theory, built for telephone engineers, turns out to be exactly what we need. The amount of information that data carries about an unknown quantity is mutual information. The most honest way to express ignorance is maximum entropy. The penalty for wrong beliefs is measured by KL divergence. Information theory and inference are not merely related; they are the same subject viewed from different angles.

This chapter forges the connection. We shall see that Bayesian inference, that framework for updating beliefs in light of evidence, speaks the language of entropy and mutual information. The tools we built to understand compression and communication illuminate reasoning itself.

11.1 The Detective's Problem

Let us be concrete before we are abstract.

A Biased Coin

Consider the simplest case: a coin that may be biased. Someone hands you a coin and says it lands heads with some probability p, but they will not tell you what p is. You flip it three times and get heads, heads, heads.

What have you learned?

Before flipping, if you knew nothing about the coin, you might have said p could be anything from 0 to 1. After seeing three heads, you suspect p is probably large. A fair coin would produce three heads in a row one time in eight; a coin with $p = 0.9$ would do it about 73% of the time. The evidence points toward a biased coin.

Before the flips, p could be anything. After three heads, you strongly suspect p is large—but you cannot be certain. This is the essence of inference.

But you cannot be certain. Even a fair coin sometimes produces three heads. The data has moved your beliefs, but it has not settled the matter. This is inference: using evidence to update what we believe, without ever achieving perfect certainty.

Prior, Likelihood, Posterior

To make this precise, we need three ingredients.

The prior: Before seeing any data, what do you believe about p? If you know nothing, you might assign equal probability to all values—a uniform distribution on $[0, 1]$. This represents maximal ignorance.

The likelihood: Given a particular value of p, what is the probability of seeing the data you saw? If $p = 0.7$, the probability of three heads is $0.7^3 = 0.343$. If $p = 0.5$, it is $0.5^3 = 0.125$. The likelihood function summarizes what the evidence says about different possible values.

The posterior: After seeing the data, what should you believe? Bayes' theorem tells us:

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

For our three heads with a uniform prior:

$$P(p \mid \text{HHH}) \propto p^3 \times 1 = p^3$$

The normalizing constant is found by integration: $\int_0^1 p^3 dp = 1/4$. So the posterior density is $4p^3$—a distribution that concentrates near $p = 1$.

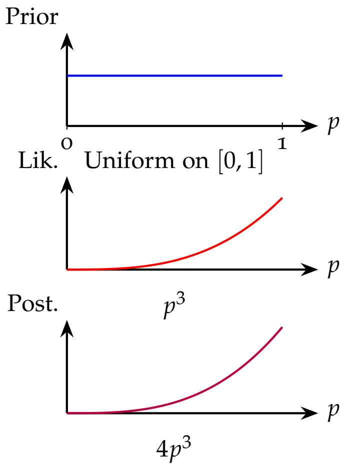


Figure 11.1: Bayesian updating. The prior (uniform) combines with the likelihood (p^3 for three heads) to produce a posterior concentrated near $p = 1$.

The Philosophical Objection

You might say: "But I do not have a prior! I want to know the true value of p, not to play games with distributions."

This is a common objection, and it deserves a serious answer.

The truth is that all inference involves assumptions. Classical statistics asks: "What would happen if I repeated this experiment many

The prior makes assumptions explicit. Classical statistics also makes assumptions—it simply hides them in the choice of test or estimator.

times?" But that requires assuming the experimental conditions remain fixed—itself an assumption. The Bayesian framework makes assumptions explicit through the prior. The prior is not a weakness; it is an honest statement of what you knew before seeing the data.

If two people have different priors, they will reach different posteriors from the same data. This is not a bug but a feature. A coin expert who knows that most manufactured coins are nearly fair will update differently from someone who suspects tampering. Both are reasoning correctly from their different starting points.

When the data is abundant, the prior becomes irrelevant. The likelihood dominates. In the limit of infinite data, all reasonable priors converge to the truth. This is reassuring: inference does not depend on arbitrary choices when evidence is plentiful.

11.2 Mutual Information as Expected Information Gain

Now we can ask our central question: how much did the data tell us?

Information Gain for a Single Outcome

Before seeing any flips, your uncertainty about p was captured by the entropy of the prior:

$$H(\text{prior}) = H(\text{Uniform}[0,1])$$

For a continuous uniform distribution on $[0,1]$, this is technically $\log 1 = 0$ (differential entropy can be zero or negative, but that is a technicality we need not dwell on). What matters is the *change* in entropy.

Information gain is the reduction in uncertainty. Entropy before minus entropy after. This difference is well-defined even when individual entropies are problematic.

After seeing three heads, your uncertainty is captured by the entropy of the posterior:

$$H(\text{posterior}) = H(4p^3 \text{ on } [0,1])$$

The information gained from this specific outcome is:

$$\text{Information gain} = H(\text{prior}) - H(\text{posterior})$$

Let us compute. The entropy of the Beta$(4,1)$ distribution (which is $4p^3$) is:

$$\begin{aligned} H(\text{Beta}(4,1)) &= \log B(4,1) - (4-1)\psi(4) - (1-1)\psi(1) + (4+1-2)\psi(5) \\ &\approx -0.217 \text{ (in nats)} \end{aligned}$$

The uniform prior Beta$(1,1)$ has entropy 0. So the information gain is about 0.217 nats, or roughly 0.31 bits.

Expected Information Gain

But here is the puzzle: we did not know in advance that we would see three heads. We might have seen two heads and a tail, or no heads at all. Each outcome would produce a different posterior, with a different entropy.

The natural question is: *on average*, how much information do we expect to gain?

We do not know what data we will observe. So we ask: averaging over all possible outcomes, how much do we expect to learn?

Let Θ represent the unknown parameter (the coin bias p), and let D represent the data (the sequence of flips). Before observing, our uncertainty about Θ is $H(\Theta)$. After observing $D = d$, our uncertainty is $H(\Theta \mid D = d)$.

The expected posterior entropy is:

$$H(\Theta \mid D) = \sum_d P(D = d)\, H(\Theta \mid D = d)$$

And the expected information gain is:

$$\text{Expected information gain} = H(\Theta) - H(\Theta \mid D)$$

But wait. This is exactly the definition of mutual information!

$$I(\Theta; D) = H(\Theta) - H(\Theta \mid D)$$

The expected information gain from observing D about the unknown Θ is the mutual information $I(\Theta; D)$.

The Unification

This is worth pausing over. Mutual information, which we introduced to analyze communication channels, turns out to be exactly what measures learning.

Mutual information answers both "How much can we communicate?" and "How much can we learn?" It is the fundamental quantity of information.

In Chapter 3, we showed that mutual information measures how much observing the channel output tells you about the input. Now we see it measures how much observing data tells you about unknown parameters. The mathematics is identical; only the interpretation changes.

The properties of mutual information now make intuitive sense for inference:

- $I(\Theta; D) \geq 0$: Data never makes us more uncertain on average.
- $I(\Theta; D) = 0$ if and only if Θ and D are independent: uninformative data tells us nothing.
- $I(\Theta; D) = H(\Theta)$ when D determines Θ exactly: we learn everything.
- $I(\Theta; D) = I(D; \Theta)$: the information that data carries about parameters equals the information that parameters carry about data.

A Numerical Example

Let us see this concretely for coin flips.

Suppose the prior on p is uniform on $[0, 1]$. We flip the coin once and observe heads or tails. How much information do we gain on average?

The calculation requires integrating over the prior and the binomial likelihood. The result: one flip provides about 0.28 bits of information about p.

Flips	$I(p; \text{data})$
1	0.28 bits
2	0.49 bits
5	0.92 bits
10	1.42 bits
50	2.42 bits
100	2.92 bits

Table 11.1: Mutual information between coin bias p (uniform prior) and observed flips. Each flip adds information, but with diminishing returns.

Why so little? Because a single flip tells you only whether the coin came up heads or tails—one bit of data. But that one bit is not entirely about p; much of it is noise. Even knowing p exactly, you cannot predict the flip with certainty.

As the number of flips increases, the mutual information grows, but with diminishing returns. The first flip is worth more than the tenth; the tenth is worth more than the hundredth. Eventually, you know p almost perfectly, and additional flips add little.

You might object: "The information depends on the prior. With a different prior, I would get a different answer."

Yes—and this is correct! If you already know $p \approx 0.5$ (a concentrated prior), a single flip tells you less than if you were completely ignorant (a uniform prior). The information gain depends on what you knew before. This is not a defect but a feature: information is always relative to a starting point.

11.3 *The Maximum Entropy Principle*

We have seen that mutual information measures how much we learn. But there is a complementary question: if we must state beliefs without data, how should we choose?

The Problem of Priors

Consider: you know that a die produces outcomes with some average of 4.2 (rather than 3.5 for a fair die). What probabilities should you assign to each face?

Infinitely many distributions have mean 4.2. You could assign all probability to face 6 and face 1 in just the right proportions. You could concentrate everything on faces 4 and 5. You could spread probability across all six faces. Which is correct?

Many probability distributions have the same mean. How do we choose among them? Maximum entropy says: pick the one that assumes the least.

The problem is that any specific choice assumes more than you know. If you put all probability on two faces, you are assuming the other faces never come up. But you were not told that.

Maximize Entropy Subject to Constraints

The maximum entropy principle says: among all distributions satisfying your constraints, choose the one with maximum entropy.

Why? Because any other distribution assumes more information than you have. Entropy measures uncertainty; maximizing it means assuming as little as possible beyond what you know.

For the die with mean 4.2:

$$\text{Maximize } H(p_1, \ldots, p_6) = -\sum_{k=1}^{6} p_k \log p_k$$

subject to:

$$\sum_{k=1}^{6} p_k = 1 \quad \text{(normalization)}$$

$$\sum_{k=1}^{6} k \cdot p_k = 4.2 \quad \text{(mean constraint)}$$

Using Lagrange multipliers, the solution is:

$$p_k = \frac{e^{\lambda k}}{Z} \quad \text{where } Z = \sum_{j=1}^{6} e^{\lambda j}$$

The parameter λ is chosen so that $\sum k \cdot p_k = 4.2$.

The result is an exponential distribution on the faces. Higher-numbered faces are more likely (to achieve the high mean), but all faces have nonzero probability. We have not assumed any face is impossible—because we were not told any face is impossible.

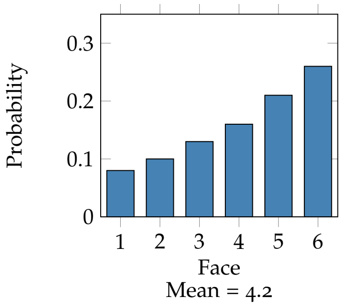


Figure 11.2: Maximum entropy distribution for a die with mean 4.2. Higher faces are more likely, but all faces have nonzero probability.

Recovering Familiar Distributions

The maximum entropy principle derives many familiar distributions:

- Known mean only (positive variable): exponential distribution
- Known mean and variance: Gaussian distribution
- No constraints (finite outcomes): uniform distribution
- Known geometric mean (positive variable): power law distribution

Let us derive the exponential case. Suppose $X > 0$ and we know only that $E[X] = \mu$. Maximize:

$$H = -\int_0^\infty p(x) \log p(x)\, dx$$

subject to $\int_0^\infty p(x)\, dx = 1$ and $\int_0^\infty x\, p(x)\, dx = \mu$.

The variational calculation gives:

$$p(x) = \frac{1}{\mu} e^{-x/\mu}$$

This is the exponential distribution with mean μ—the most famous distribution in queuing theory, reliability engineering, and radioactive decay.

The exponential distribution is the maximum entropy distribution when you know only the mean. The Gaussian emerges when you know mean and variance.

Connection to Thermodynamics

We have seen this before! In Chapter 9, we derived the Boltzmann distribution by maximizing entropy subject to fixed energy. That was thermodynamic equilibrium: the state that assumes nothing beyond the total energy.

The maximum entropy principle unifies statistical mechanics and inference. Both ask: given constraints, what is the most honest probability distribution? The Boltzmann distribution is not merely a physical fact; it is a logical consequence of maximum entropy reasoning. Thermodynamics is inference.

Statistical mechanics is a special case of maximum entropy inference. The Boltzmann distribution maximizes entropy subject to fixed average energy.

E.T. Jaynes, who developed this connection most fully, called it "probability theory as extended logic." Given what you know, there is a unique most honest way to quantify uncertainty. That way is maximum entropy.

A Philosophical Aside

You might ask: is maximum entropy objective or subjective?

In one sense, it is objective. Given a set of constraints, the maximum entropy distribution is unique. Different people with the same constraints must reach the same distribution.

In another sense, it is subjective. The choice of constraints is yours. If you decide that only the mean matters, you get the exponential. If you also constrain the variance, you get the Gaussian. The constraints encode your modeling choices.

This is the right level of subjectivity. Given what you know, the inference is determined. But what you know—or what you choose to model—is up to you. Maximum entropy separates the subjective (constraint choice) from the objective (inference given constraints).

11.4 KL Divergence: The Price of Wrong Beliefs

Maximum entropy tells us how to encode ignorance. But what if our beliefs are wrong? How do we measure the cost?

Definition and Interpretation

The Kullback-Leibler divergence between distributions P and Q is:

$$D_{\mathrm{KL}}(P\|Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

or, for continuous distributions:

$$D_{\mathrm{KL}}(P\|Q) = \int p(x) \log \frac{p(x)}{q(x)}\, dx$$

KL divergence measures the cost of believing Q when the truth is P. It has units of bits (or nats) and is always non-negative.

What does this measure? Consider encoding data from P using a code optimized for Q. The average code length is $H(P) + D_{\mathrm{KL}}(P\|Q)$, whereas the optimal code length is just $H(P)$. The KL divergence is the coding penalty for using the wrong distribution.

Equivalently, if P is true and you believe Q, then $D_{\mathrm{KL}}(P\|Q)$ is the expected extra surprise per observation. You think events have probability $Q(x)$, but they actually have probability $P(x)$. When Q underestimates the true probability, you are more surprised than necessary.

Key Properties

KL divergence has several important properties:

1. $D_{\mathrm{KL}}(P\|Q) \geq 0$, with equality if and only if $P = Q$
2. It is not symmetric: $D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$ in general
3. It is not a distance (does not satisfy the triangle inequality)

The asymmetry is meaningful. Believing Q when P is true differs from believing P when Q is true. Suppose P assigns positive probability to an event that Q says is impossible ($Q(x) = 0$). Then $D_{\mathrm{KL}}(P\|Q) = \infty$: your model assigns zero probability to something that actually happens. But $D_{\mathrm{KL}}(Q\|P)$ might be finite: your model just overpredicts some events.

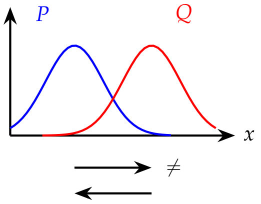


Figure 11.3: KL divergence is asymmetric: $D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$.

KL Divergence and Bayesian Updating

There is a beautiful connection between KL divergence and Bayesian inference.

Consider the expected KL divergence from the posterior to the prior:

$$\mathbb{E}_D[D_{\mathrm{KL}}(\text{posterior}\|\text{prior})]$$

This measures, on average, how much the posterior differs from the prior. It turns out that this expected divergence equals the mutual information between parameters and data:

$$I(\Theta; D) = \mathbb{E}_D[D_{\mathrm{KL}}(P(\Theta|D)\|P(\Theta))]$$

Bayesian updating can be seen as moving through probability space. The prior is your starting point; the posterior is where you end up. The average distance traveled (in KL divergence) equals the information gained.

The expected KL divergence from posterior to prior equals the mutual information. Learning can be measured as movement in probability space.

Maximum Entropy as Minimum KL

Here is another unifying perspective. The maximum entropy distribution subject to constraints can be characterized as the distribution that minimizes KL divergence from the uniform distribution (or from a specified reference), subject to the constraints.

If your "base" distribution is uniform and you want to incorporate a constraint like $E[X] = \mu$, the maximum entropy solution is the distribution closest to uniform (in KL sense) that satisfies the constraint.

This perspective unifies maximum entropy with Bayesian updating. Both are about moving the minimum distance in KL divergence from a starting point to satisfy new information. The starting point for maximum entropy is typically uniform (maximum ignorance); the starting point for updating is the prior.

11.5 Fisher Information and the Limits of Precision

We have discussed Shannon information—entropy and mutual information. But there is another quantity called "information" in statistics, developed by R.A. Fisher in the 1920s, that measures something different.

The Estimation Problem

Suppose data comes from a distribution $p(x|\theta)$, where θ is an unknown parameter. You want to estimate θ from the data.

How precise can your estimate be? This is a different question from "how much information does data carry about θ?" Here we ask about the best possible estimator, in terms of variance.

Fisher information measures how precisely a parameter can be estimated. It is the curvature of the likelihood function at the true parameter value.

Fisher Information Defined

The Fisher information is:

$$\mathcal{I}(\theta) = \mathbb{E}\left[\left(\frac{\partial}{\partial\theta}\log p(X|\theta)\right)^2\right]$$

This measures how sharply peaked the log-likelihood is around the true parameter. If the likelihood changes rapidly with θ, small changes in data give strong evidence about θ—high Fisher information. If the likelihood is flat, data distinguishes poorly between nearby parameter values—low Fisher information.

An equivalent formula, when the regularity conditions hold:

$$\mathcal{I}(\theta) = -\mathbb{E}\left[\frac{\partial^2}{\partial\theta^2}\log p(X|\theta)\right]$$

Fisher information is the expected curvature of the log-likelihood.

The Cramer-Rao Bound

Fisher information leads to a fundamental limit on estimation. For any unbiased estimator $\hat{\theta}$:

$$\text{Var}(\hat{\theta}) \geq \frac{1}{\mathcal{I}(\theta)}$$

This is the Cramer-Rao bound. No matter how clever your estimation procedure, you cannot beat this variance. The bound is achieved by certain "efficient" estimators, including the maximum likelihood estimator under good conditions.

The Cramer-Rao bound: no unbiased estimator can have variance less than $1/\mathcal{I}(\theta)$. This is fundamental, not a limitation of our cleverness.

Let us see this for coin flips. If you flip a coin n times to estimate the bias p:

$$\mathcal{I}(p) = \frac{n}{p(1-p)}$$

The Cramer-Rao bound gives:

$$\text{Var}(\hat{p}) \geq \frac{p(1-p)}{n}$$

The maximum likelihood estimator $\hat{p} = k/n$ (the fraction of heads) achieves this bound asymptotically. If $n = 100$ and the true $p = 0.3$:

$$\text{Minimum standard deviation} = \sqrt{\frac{0.3 \times 0.7}{100}} \approx 0.046$$

You cannot do better than about 4.6% standard deviation with 100 flips from a 30% coin.

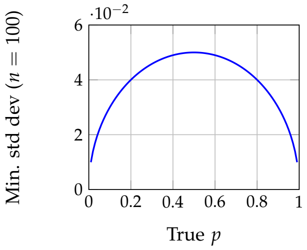


Figure 11.4: Cramer-Rao lower bound on standard deviation for estimating coin bias with 100 flips. Estimation is hardest near $p = 0.5$.

Connection to Shannon Information

Fisher information and Shannon information both use the word "information," but they measure different things. Are they related?

Yes, through KL divergence. The KL divergence between $p(x|\theta)$ and $p(x|\theta + d\theta)$ expands, for small $d\theta$:

$$D_{\text{KL}}(p(\cdot|\theta)\|p(\cdot|\theta + d\theta)) \approx \frac{1}{2}\mathcal{I}(\theta)\,(d\theta)^2$$

Fisher information is the local curvature of KL divergence in parameter space. It measures how quickly distributions become distinguishable as parameters change.

Fisher information is the local curvature of KL divergence. It measures how distinguishable nearby parameter values are.

This connects two conceptions of information. Shannon entropy measures uncertainty about outcomes. Fisher information measures sensitivity to parameters. KL divergence bridges them: it measures distinguishability of distributions, and Fisher information is its second derivative.

Two Fishers, One Framework

A historical note, which I cannot resist. R.A. Fisher, who developed Fisher information in the 1920s, and Shannon, who developed Shannon information in the 1940s, worked in different fields (statistics and communication) with different motivations. The connection between their "informations" was clarified later.

The pun writes itself: two Fishers, one framework. But the deeper point is that information theory unifies concepts that arose independently. The mathematics of uncertainty is the same whether we are estimating parameters, decoding messages, or understanding thermodynamics.

11.6 Information in Hypothesis Testing

Let us apply these ideas to a concrete problem: distinguishing between two hypotheses.

The Setup

Suppose you are testing whether a coin is fair (H_0: $p = 0.5$) or biased (H_1: $p = 0.6$). You flip it n times and must decide which hypothesis is true.

Hypothesis testing asks: can we distinguish two explanations? KL divergence measures their distinguishability.

There are two kinds of errors:

- Type I error (false positive): deciding H_1 when H_0 is true
- Type II error (false negative): deciding H_0 when H_1 is true

With limited data, you cannot avoid both errors. There is a tradeoff.

KL Divergence as Distinguishability

How distinguishable are the two hypotheses? KL divergence provides the answer.

Under H_0, each flip is Bernoulli(0.5). Under H_1, it is Bernoulli(0.6).

The KL divergence per flip is:

$$\begin{aligned} D_{\text{KL}}(H_0 \| H_1) &= 0.5 \log \frac{0.5}{0.6} + 0.5 \log \frac{0.5}{0.4} \\ &= 0.5 \times (-0.263) + 0.5 \times (0.322) \\ &\approx 0.029 \text{ nats} \approx 0.042 \text{ bits} \end{aligned}$$

This is small—the hypotheses are hard to distinguish. After n flips, the total KL divergence is about $0.042n$ bits.

For $n = 100$, we have about 4.2 bits of distinguishing power. The error probability decays roughly as $2^{-4.2} \approx 0.054$—about 5%.

Flips	KL (bits)	Min. error
10	0.42	0.67
50	2.1	0.23
100	4.2	0.05
200	8.4	0.003

Table 11.2: Distinguishing $p = 0.5$ from $p = 0.6$. As flips increase, the KL divergence grows and error probability drops exponentially.

The Neyman-Pearson Lemma

The optimal test—the one that minimizes Type II error for a given Type I error—is the likelihood ratio test. You compute:

$$\Lambda = \frac{P(\text{data}|H_1)}{P(\text{data}|H_0)}$$

and decide H_1 if Λ exceeds a threshold.

From an information-theoretic viewpoint, the likelihood ratio captures all the information relevant for distinguishing the hypotheses. The sufficient statistic (in this case, the number of heads) contains everything the data says about which hypothesis is true. Processing the data further cannot help.

This is the data processing inequality applied to hypothesis testing. You cannot distinguish hypotheses better by throwing away information.

11.7 Historical Development

Let us pause to appreciate how these ideas came together.

Fisher's Foundations (1920s)

R.A. Fisher developed maximum likelihood estimation, sufficient statistics, and Fisher information in a series of papers in the 1920s. He was concerned with agricultural experiments: how to design trials and analyze data efficiently.

Fisher's framework was frequentist: probabilities describe long-run frequencies, and parameters are fixed (not random). He did not connect his work to entropy or communication.

Fisher developed his information concept in the 1920s for agricultural statistics. Shannon's entropy came later, for communication. Jaynes unified them.

Shannon's Revolution (1948)

Shannon's 1948 paper established information theory as a mathematical discipline. He defined entropy, mutual information, and channel capacity. He proved the fundamental theorems of source and channel coding.

Shannon's focus was communication: how to transmit messages efficiently and reliably. He did not explicitly develop applications to inference, though the tools were there.

Jaynes and the Maximum Entropy Revolution (1950s–1980s)

E.T. Jaynes, a physicist, made the crucial synthesis. In his 1957 papers "Information Theory and Statistical Mechanics," he showed that the Boltzmann distribution emerges from maximum entropy reasoning. Thermodynamics is not a physical peculiarity but a consequence of consistent inference.

Jaynes spent the rest of his career advocating "probability theory as extended logic." His book *Probability Theory: The Logic of Science*, published posthumously in 2003, remains the definitive statement of this view.

Jaynes spent decades advocating that probability is logic. Maximum entropy follows from consistency requirements. His view is now mainstream in Bayesian statistics.

The Bayesian and information-theoretic perspectives, once seen as separate, are now understood as aspects of one framework. Information theory provides the quantities; Bayesian inference provides the updating rules; maximum entropy provides the priors. They fit together.

Cox's Theorem (1946, 1961)

Underlying this synthesis is a remarkable result by R.T. Cox. He asked: what mathematical rules must "degrees of belief" satisfy to be logically consistent?

Starting from simple requirements (like: if belief in A and belief in B given A are both high, belief in A and B should be high), Cox derived that any consistent system of beliefs must satisfy the rules of probability theory. This is not an assumption but a theorem.

Cox showed that probability theory is the unique consistent system for degrees of belief. Any other system leads to logical contradictions.

If Cox is right, then Bayesian inference is not one option among many. It is the only consistent way to quantify uncertainty and update beliefs. Maximum entropy follows as a corollary: given constraints, the unique consistent prior is the maximum entropy one.

11.8 From Information to Learning

Let us take stock of what we have established.

The Unified Picture

Mutual information measures expected information gain. When you observe data and update your beliefs, the average reduction in uncertainty equals $I(\Theta; D)$, the mutual information between parameters and data.

Maximum entropy provides principled priors. When you must state beliefs without data, the most honest distribution is the one with maximum entropy subject to your constraints. Any other choice smuggles in assumptions.

KL divergence measures the cost of wrong beliefs. If you believe Q when the truth is P, you pay $D_{\mathrm{KL}}(P\|Q)$ bits of coding inefficiency per observation. Wrong beliefs have a price.

Fisher information limits precision. No estimator can have variance below $1/\mathcal{I}(\theta)$. There are fundamental limits on what we can learn, not just practical ones.

The unified view: mutual information measures learning, maximum entropy encodes ignorance, KL divergence penalizes errors, Fisher information limits precision.

The Deeper Lesson

Information is not merely something we transmit through wires. It is the currency of inference—the measure of how much we know and how much we can learn.

This is why the same quantity, entropy, appears in communication theory and in statistical mechanics and in Bayesian inference. It is not three things with the same name; it is one thing viewed from three angles.

The demon of Chapter 10 taught us that information is physical. Now we see that information is logical: the measure of what we know, and the guide to honest reasoning.

Entropy appears everywhere because uncertainty is everywhere. Quantifying uncertainty is the fundamental problem of science, engineering, and thought.

An Open Question

We have treated parameters as fixed and data as random. But what if we flip this perspective? What if we ask: how much data is needed to learn a given model?

This leads to a remarkable principle that we will develop in the next chapter: the minimum description length principle. It says that the best model is the one that compresses the data most. Compression and learning are two faces of the same coin.

Think about it: a good model predicts the data. A good prediction means short coding (you assign high probability to what happens). Short coding is good compression. So a model that predicts well is a model that compresses well.

Good prediction is good compression. A model that predicts data well assigns it high probability, which means short code lengths. This connects learning to coding.

This is not a metaphor. It is a mathematical identity. The Bayesian posterior predictive distribution is exactly the distribution that minimizes expected code length for new data. Learning and compression are the same thing.

In Chapter 12, we develop this idea fully. We will see how Occam's razor—the preference for simple explanations—emerges from information theory. We will see that every good learner is, at heart, a compressor. And we will see fundamental limits on what can be learned from data.

But that is for next time. For now, we have forged the link between Shannon's theory and the problem of inference. The tools built for telephone engineers illuminate reasoning itself.

Historical note. The unification of information theory and inference took decades. Shannon's 1948 paper focused on communication, mentioning inference only in passing. Fisher, working in statistics, never connected his "information" to Shannon's. Jaynes, starting in 1957, spent thirty years advocating the connection, often against resistance. By the 1990s, the Bayesian interpretation was mainstream. Today, information theory and statistical inference are recognized as aspects of one framework. The story illustrates how ideas can take generations to synthesize, even when the mathematics is clear. The connections were always there; we had to learn to see them.

12
Minimum Description Length

Suppose I show you ten data points, scattered roughly along a line but with some noise. You want to fit a curve. A straight line captures the trend, leaving small residuals. A degree-nine polynomial passes through every point exactly, leaving no residuals at all.

The polynomial achieves zero error. The line has residuals. By the standard of "minimize error," the polynomial wins. Yet something feels deeply wrong about this.

Which is the better fit?

If your goal is to minimize error on these ten points, the polynomial wins. It achieves perfection. Yet anyone with scientific intuition recoils. The polynomial is not capturing the underlying pattern; it is memorizing the noise. The line, with its imperfect fit, somehow seems truer.

But why? What makes us prefer the simpler model despite its larger errors?

The usual answer is Occam's razor: prefer the simpler explanation. But this is philosophy, not mathematics. It tells us what to prefer without telling us why, or how much. Is a model with three parameters twice as good as one with six? Half as good? How do we compare the cost of complexity against the benefit of fit?

This chapter provides the answer. The Minimum Description Length principle—MDL for short—transforms Occam's razor from a vague heuristic into a precise mathematical statement. The best model, MDL says, is the one that *compresses the data most*. Simplicity is not aesthetic prejudice; it is information-theoretic necessity.

12.1 The Two-Part Code

Let us think carefully about what it means to describe data.

Description as Communication

Imagine you need to transmit those ten data points to a colleague. The colleague knows nothing about your experiment; you must tell them everything.

One approach: send the raw coordinates. Each point (x_i, y_i) requires some number of bits. If x and y are each specified to 10-bit precision, you need 20 bits per point, or 200 bits total.

Transmission is description. A code that sends data efficiently must capture its structure. This is the heart of MDL.

But suppose you first send a model, then send only the residuals—the deviations from the model's predictions. If the model captures the pattern, the residuals are small. Small numbers need fewer bits.

With the linear model $y = 2.1 + 0.8x$:

1. Send the model: two parameters (slope and intercept), maybe 20 bits each, so 40 bits total
2. Send the residuals: if they are small (say, within ± 0.5), each might need only 4 bits instead of 10, saving 60 bits
3. Total: 40 + 40 = 80 bits, versus 200 for the raw data

The model has earned its keep. By spending 40 bits to describe the pattern, we saved more than 40 bits on the residuals.

The Fundamental Tradeoff

Now consider the degree-nine polynomial. It passes through every point, so the residuals are zero. But describing nine coefficients (plus the constant) requires perhaps 200 bits. The total message is: 200 bits for the model, 0 bits for the residuals, giving 200 bits overall.

Simple models are cheap to describe but may leave large residuals. Complex models are expensive to describe but can reduce residuals to nothing. MDL balances these costs.

We have gained nothing! The polynomial is so complex that describing it costs as much as just listing the data. Its perfect fit is an illusion—it compresses nothing.

Here is the tradeoff:

- Simple models: cheap to describe, but may leave expensive residuals
- Complex models: expensive to describe, but may leave cheap (or zero) residuals

The MDL principle says: minimize the total. Find the sweet spot where the sum of model description length plus residual description length is smallest.

Making It Precise

Let us write this mathematically. Given data D and a model M:

$$L(D, M) = L(M) + L(D \mid M)$$

where $L(M)$ is the description length of the model and $L(D \mid M)$ is the description length of the data given the model.

$L(M) + L(D \mid M)$ is the total message length: describe the model, then describe how the data deviates from it. Minimize this sum.

The MDL principle: among all candidate models, choose the one that minimizes $L(D, M)$.

This is not merely a heuristic. It is a theorem about compression. The model that minimizes total description length is, by definition, the model that compresses the data most. And compression, as we have seen throughout this book, is prediction. A model that compresses well must be capturing genuine structure.

12.2 What Goes Into Description Length?

We have been vague about what "description length" means. Let us be precise.

Describing the Model Class

First, we must say what kind of model we are using. Are we fitting a line? A polynomial? A neural network? This takes some bits.

If we have K model classes under consideration, specifying which one requires $\log K$ bits. This is usually a small contribution.

Describing the Parameters

Next, we must specify the parameters. A linear model has two parameters (slope and intercept). A degree-d polynomial has $d + 1$ parameters. A neural network might have millions.

Parameters must be specified to some precision. More precision means longer descriptions. The right precision is part of what MDL optimizes.

But parameters are real numbers, and real numbers have infinitely many digits. How do we describe them finitely?

The answer is discretization. We specify each parameter to k bits of precision. This allows 2^k possible values per parameter, spread over whatever range we consider reasonable.

If a model has p parameters, each specified to k bits:

$$L(\text{parameters}) = p \cdot k$$

More parameters means longer descriptions. Higher precision means longer descriptions. Both contribute to model complexity.

Describing the Residuals

Finally, we describe how the data deviates from the model's predictions.

If we assume the residuals $r_i = y_i - \hat{y}_i$ are drawn from some distribution—say, Gaussian with variance σ^2—then the description length for each residual is approximately $-\log P(r_i)$.

For Gaussian residuals:

$$L(D \mid M) = \sum_{i=1}^{n} (-\log P(r_i)) = \frac{n}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2 \ln 2} \sum_{i=1}^{n} r_i^2$$

The key term is the sum of squared residuals. Larger residuals mean longer descriptions. A model that fits poorly is expensive to "correct" because its errors must be transmitted explicitly.

Large residuals cost more bits. A model that fits poorly produces large r_i, inflating $L(D \mid M)$.

A Numerical Example

Let us make this concrete with polynomial regression.

We generate $n = 20$ data points from the true model $y = 2 + 3x + \epsilon$, where ϵ is Gaussian noise with $\sigma = 1$. We fit polynomials of degree $d = 0, 1, 2, \ldots$ and compute the description lengths.

I have assumed 8 bits per parameter and computed $L(D \mid M)$ from the sum of squared residuals assuming $\sigma = 1$ is known.

d	Params	$\sum r_i^2$	$L(M)$	Total
0	1	185	8	228
1	2	21	16	68
2	3	20	24	75
3	4	19	32	80
5	6	17	48	92
10	11	12	88	124

Table 12.1: Description lengths for polynomial fits. The linear model ($d = 1$) achieves the minimum total, correctly identifying the true model order.

The degree-0 model (just a constant) gives huge residuals—it cannot capture the linear trend. The degree-1 model (the true model) fits well with only two parameters. Higher-degree models reduce residuals slightly, but the additional parameters cost more than the reduction saves.

MDL correctly identifies degree 1 as optimal. It finds the true model.

What If We Have More Data?

Now suppose we repeat the experiment with $n = 200$ points instead of 20.

The model description length $L(M)$ stays the same—it depends only on the number of parameters, not the amount of data.

But $L(D \mid M)$ scales with n. With ten times more data, the residual term is roughly ten times larger in absolute size.

With more data, the model complexity penalty becomes relatively smaller. More data justifies more complex models.

What happens? The ratio $L(M)/L(D \mid M)$ shrinks. The complexity penalty becomes relatively less important. With abundant data, we can afford more complex models.

This is exactly right. With 20 points, we should be conservative; the data cannot justify much complexity. With 200 points, we can reliably distinguish a quadratic from a line. With 2000 points, we might detect subtle cubic terms.

MDL automatically adjusts its conservatism to the data size. It is appropriately humble with limited data and appropriately ambitious with abundant data.

12.3 The Bayesian Connection

Here is a remarkable fact: MDL and Bayesian model selection are two faces of the same coin.

Bayes' Theorem for Models

Given data D and candidate models $M_1, M_2, \ldots$, Bayesian inference computes:

$$P(M \mid D) \propto P(D \mid M) \cdot P(M)$$

The best model maximizes this posterior probability. Taking negative logarithms:

$$-\log P(M \mid D) = -\log P(D \mid M) - \log P(M) + \text{const}$$

Minimizing the negative log posterior is equivalent to maximizing the posterior.

The Correspondence

Compare the expressions:

$$\begin{aligned} \text{MDL:} &\quad L(M) + L(D \mid M) \\ \text{Bayesian:} &\quad -\log P(M) - \log P(D \mid M) \end{aligned}$$

MDL description lengths correspond exactly to Bayesian negative log probabilities. The model prior *is* a code. The likelihood *is* a code.

These are the same if:

- $L(M) = -\log P(M)$: model description length equals negative log prior
- $L(D \mid M) = -\log P(D \mid M)$: residual description length equals negative log likelihood

The first equation says that assigning a code of length L to model M is equivalent to assigning it a prior probability $P(M) = 2^{-L}$. Short codes correspond to high priors. A model that is easy to describe is a model we consider plausible before seeing the data.

The second equation is just the source coding theorem applied to residuals. If residuals come from distribution $P(r \mid M)$, the optimal code length is $-\log P(r \mid M)$.

Where Do Priors Come From?

Bayesian inference is sometimes criticized for requiring "subjective" priors. Where do these priors come from? Who decides?

MDL provides a different perspective. The prior is not a subjective belief; it is a coding scheme. The question "What prior should I use?" becomes "What code should I use?"

The prior is a code. The code is a prior. These are not metaphors but mathematical identities. Choosing one is choosing the other.

And coding schemes are not arbitrary. A good code assigns short lengths to models we expect to encounter frequently and long lengths to bizarre ones. A prior that assigns equal probability to all possible neural network weight configurations is like a code that wastes bits on nonsense—inefficient and wrong.

In practice, priors and codes both encode our beliefs about which models are reasonable. The MDL perspective makes this explicit: your prior is revealed by how efficiently you can describe each model. If a model is easy to describe, you implicitly believe it is plausible.

A Philosophical Aside

You might say: "This is circular. The prior determines the code, and the code determines the prior. We have explained nothing."

There is something to this objection. But consider: what alternative is there? Any method of choosing among models must somehow penalize complexity. Whether we call it a "prior" or a "code" or a "regularization penalty," we must say how much complexity costs.

MDL makes the cost explicit in bits. Bayesian inference makes it explicit in log-probabilities. These are the same thing measured in different units. The advantage of both frameworks is that they force us to state our assumptions. A Bayesian must write down a prior; an MDL user must specify an encoding. Neither can hide behind vague appeals to "simplicity."

12.4 Normalized Maximum Likelihood

We have been discretizing parameters to compute description lengths. But this involves arbitrary choices: How many bits? What range? Different choices give different answers.

Can we do better? Can we define description length without arbitrary discretization?

The Problem with Two-Part Codes

In a two-part code, we first encode the model, then the data. But encoding continuous parameters requires discretization, and the resulting description lengths depend on our discretization choices.

Two-part codes require choosing how to encode parameters. The Normalized Maximum Likelihood distribution eliminates this choice.

This is not just a technical annoyance. It means that MDL, as we have presented it, gives different answers depending on how we encode parameters. The principle seems less principled than we might hope.

The NML Distribution

The solution is the Normalized Maximum Likelihood (NML) distribution. For a model class with parameter θ, define:

$$P_{\text{NML}}(x) = \frac{P(x \mid \hat{\theta}(x))}{\sum_y P(y \mid \hat{\theta}(y))}$$

where $\hat{\theta}(x)$ is the maximum likelihood estimate of θ given data x.

In words: for each possible dataset y, compute how well it can be fit by the best parameter value for that dataset. The NML probability of x is how well x can be fit, normalized by the total “fittability” of all possible datasets.

Why This Works

The denominator—call it C—measures the complexity of the model class. It sums up how well the class can fit every possible dataset. A flexible model class that can fit many different patterns has large C. A rigid class that fits only specific patterns has small C.

The parametric complexity C measures how flexible the model class is. Flexible classes can fit anything, so any particular fit is less impressive.

The description length under NML is:

$$L_{\text{NML}}(x) = -\log P(x \mid \hat{\theta}(x)) + \log C$$

The first term is the negative log likelihood at the best parameter value—how well the data fits. The second term is the complexity penalty—how flexible the model class is.

This is beautiful. We no longer need to discretize parameters. The complexity penalty emerges automatically from the model class itself. It measures, in a principled way, how much we should distrust a good fit.

An Example: Coin Flips

Consider n coin flips with k heads. The model class is Bernoulli(θ), with $\theta \in [0, 1]$.

The maximum likelihood estimate is $\hat{\theta} = k/n$. The likelihood at this estimate is:

$$P(x \mid \hat{\theta}) = \binom{n}{k} \left(\frac{k}{n}\right)^k \left(\frac{n-k}{n}\right)^{n-k}$$

The parametric complexity sums this over all possible values of k:

$$C = \sum_{k=0}^{n} \binom{n}{k} \left(\frac{k}{n}\right)^k \left(\frac{n-k}{n}\right)^{n-k}$$

For small n, we can compute this exactly.

The complexity grows as $\frac{1}{2}\log n$. With more data, the Bernoulli model becomes “more complex” in the sense that it can fit more precisely—but the growth is slow, only logarithmic.

n	$\log C$ (bits)
1	1.00
2	1.50
5	2.16
10	2.66
100	4.16

Table 12.2: Parametric complexity for the Bernoulli model. It grows as $\frac{1}{2}\log n + O(1)$.

The Minimax Property

NML has a remarkable optimality property. Among all possible codes for a model class, NML minimizes the worst-case regret—the maximum excess code length compared to knowing the true parameter.

This makes NML the "safest" choice. It never does much worse than the best possible, for any data and any true parameter. It is the coding analog of minimax decision theory.

12.5 Kolmogorov Complexity: The Ultimate Limit

We have been comparing models within a fixed class: lines versus polynomials, one degree versus another. But MDL's logic extends further. What if we compare across all possible descriptions?

Kolmogorov complexity asks: what is the shortest program that produces this data? It is the ultimate MDL—but uncomputable.

The Definition

The Kolmogorov complexity $K(x)$ of a string x is the length of the shortest computer program that produces x.

This captures a profound intuition. A string is "simple" if it can be generated by a short program. A string is "complex" if there is no shortcut—the only way to describe it is to list it out.

Examples

Consider a string of 1000 zeros. Its Kolmogorov complexity is roughly $\log 1000 \approx 10$ bits. A program like "print '0' 1000 times" needs only specify the character and the count.

Consider the first 1000 digits of π. The Kolmogorov complexity is again about 10 bits: "compute π to 1000 digits." Despite appearing random, π is generated by a simple algorithm.

Consider a truly random string of 1000 bits. Its Kolmogorov complexity is approximately 1000 bits. There is no pattern; no shortcut; the only description is the string itself.

`000...000` (repetitive)	$K \approx 10$
`3.14159...` (algorithmic)	$K \approx 10$
`0110100...` (random)	$K \approx 1000$

Figure 12.1: Three 1000-symbol strings. Repetitive and algorithmic strings have low complexity; random strings have high complexity.

MDL as an Approximation to Kolmogorov

Kolmogorov complexity is the ultimate MDL. The "model" is any computer program. The "residuals" are zero (the program produces the data exactly). The description length is just the program length.

Any specific MDL framework—polynomial regression, neural networks, whatever—provides an upper bound on $K(x)$. If you can describe x using 500 bits in your framework, then $K(x) \leq 500$ (plus a small constant for specifying the framework itself).

MDL with a particular model class is an approximation to the ideal of Kolmogorov complexity. We cannot compute $K(x)$ directly, but we can compute description lengths in tractable model classes.

The Catch

Kolmogorov complexity is uncomputable. There is no algorithm that, given a string x, outputs $K(x)$.

Kolmogorov complexity is uncomputable. No algorithm can determine the shortest program for an arbitrary string. MDL provides computable approximations.

Why? Because computing $K(x)$ would solve the halting problem. To find the shortest program, we would need to test all short programs to see if any produces x. But testing whether a program produces x requires running it—and we cannot know in advance whether it will halt.

This is not a practical limitation we might overcome with faster computers. It is a mathematical impossibility, as fundamental as the undecidability of the halting problem itself.

Why It Matters Anyway

If Kolmogorov complexity is uncomputable, why discuss it?

Because it provides a theoretical foundation. It tells us what MDL is trying to approximate. It shows that "simplicity" has an objective meaning independent of our choice of model class. It connects information theory to the foundations of computation.

And it offers a philosophical insight. There is a fact of the matter about how complex a string is. A string with $K(x) = 100$ bits is genuinely simpler than one with $K(x) = 1000$ bits, regardless of what model class we happen to use. MDL in practice approaches this objective truth; it does not create it.

12.6 A Worked Example: Detecting the True Model

Let us see MDL in action on a more substantive example.

The Setup

An experimentalist measures a response y at various inputs x. The true relationship is:

We generate data from a known model and see if MDL can recover it. This tests whether the principle works, not just in theory, but in practice.

$$y = 1.0 + 2.0x - 0.5x^2 + \epsilon$$

where $\epsilon \sim N(0, 0.5^2)$. The experimentalist collects 30 data points.

She considers models from degree 0 (constant) to degree 6 (sextic polynomial). Which should she choose?

The Calculation

For each degree d, she fits the polynomial by least squares, computes the sum of squared residuals, and calculates the description length.

I assume:

- Each parameter is encoded to 16 bits of precision

- The noise variance $\sigma^2 = 0.25$ is known
- The model class (polynomial) is fixed, so we need not encode it

Degree	Parameters	SSR	$L(M)$ (bits)	$L(D \mid M)$ (bits)
0	1	95.3	16	275
1	2	42.1	32	122
2	3	7.8	48	22
3	4	7.5	64	22
4	5	7.2	80	21
5	6	6.9	96	20
6	7	6.6	112	19

Table 12.3: MDL analysis for polynomial regression. SSR is sum of squared residuals.

The quadratic model achieves the minimum total description length. MDL correctly identifies the true model order.

Degree	Total L
0	291
1	154
2	70
3	86
4	101
5	116
6	131

Table 12.4: Total description lengths. Degree 2 is the clear minimum, correctly identifying the true model.

What Happened?

Going from degree 1 to degree 2, the sum of squared residuals drops dramatically (from 42 to 7.8). The extra parameter is worth it: it captures real structure.

Going from degree 2 to degree 3, the residuals barely change (from 7.8 to 7.5). The cubic term is fitting noise, not signal. The 16 bits spent describing it are not recovered in shorter residuals.

MDL balances fit against complexity automatically. It does not need to be told which improvements are "real"—the description lengths reveal this.

The Key Insight

When a model captures genuine structure, the reduction in residual description length exceeds the cost of describing the additional parameters. The total shrinks.

A coefficient that captures real structure reduces residuals by more than its description cost. A coefficient that fits noise does not pay for itself.

When a model fits noise, the reduction in residuals is small (noise, by definition, cannot be predicted). The parameter description costs more than it saves. The total grows.

This is why MDL works. It does not magically know which patterns are "real." It simply notices that real patterns compress the data, while noise does not.

12.7 MDL and Information Criteria

You may have encountered model selection criteria like AIC and BIC. These are essentially MDL in disguise.

The Akaike Information Criterion

AIC is defined as:

$$\mathrm{AIC} = -2\log L + 2k$$

where L is the maximum likelihood and k is the number of parameters.

AIC and BIC are MDL with specific parameter encoding assumptions. They differ in how heavily they penalize complexity.

The first term is twice the negative log likelihood—essentially the description length of the data given the model. The second term is a complexity penalty: 2 bits per parameter.

AIC penalizes complexity lightly. It is appropriate when the true model is rich and you have plenty of data to estimate it.

The Bayesian Information Criterion

BIC is defined as:

$$\mathrm{BIC} = -\log L + \frac{k}{2}\log n$$

where n is the sample size.

The complexity penalty now depends on the amount of data. With more data, each parameter "costs" more in the sense that we hold models to a higher standard.

BIC is consistent: as $n \to \infty$, it selects the true model with probability approaching 1. AIC is not consistent; it tends to overfit slightly even with infinite data.

Connection to MDL

Both criteria are approximations to MDL:

- AIC corresponds to a two-part code with a fixed 1-bit-per-parameter penalty
- BIC corresponds to NML for regular parametric models

The $\frac{k}{2}\log n$ term in BIC matches the parametric complexity we computed for NML. It is not an arbitrary choice but an asymptotic result from information theory.

12.8 Compression as Understanding

We have been developing MDL as a model selection principle. But there is a deeper lesson here, one that connects to the themes running through this entire book.

What Does It Mean to Understand?

Consider what it means to "understand" a phenomenon. You understand the tides when you can predict them—when, given today's conditions, you can say what tomorrow's tide will be. You understand a language when you can compress it—when, given the previous words, you can predict the next.

To understand data is to compress it. A theory that explains is a theory that predicts, and a theory that predicts is a code that compresses.

This is not metaphor. Prediction and compression are mathematically equivalent. If you can predict with probability p, you can encode in $-\log p$ bits. If you can compress to L bits, you were implicitly predicting with probabilities 2^{-L}.

A scientific theory is a compression scheme. Newton's laws compress the motions of planets into a few equations. Quantum mechanics compresses atomic spectra into wave functions. The theory that compresses most—that reduces the most data to the shortest description—is the theory that understands most.

The Limits of Understanding

But some data cannot be compressed. A truly random sequence is incompressible; its Kolmogorov complexity equals its length. No theory can predict it; no understanding can simplify it.

This is profound. It means that understanding has limits. There exist phenomena that no theory can explain, not because we are not clever enough, but because there is no pattern to find.

Random data is incompressible. This is not a failure of our methods but a fact about the data. Some things cannot be explained because there is nothing to explain.

When we encounter data that resists compression, we face a choice. Perhaps we have not found the right model class. Perhaps we need more data. Or perhaps the data is genuinely random—noise with no signal, chaos with no pattern.

MDL does not tell us which possibility holds. But it tells us when compression fails, and that is the first step toward understanding our own ignorance.

Simplicity and Truth

Why should the simplest model be truest? This is an old question, older than MDL, older than information theory.

One answer: simplicity is a proxy for probability. Among all the functions that pass through ten points, almost all are complex (high-degree polynomials, bizarre curves). Simple functions (lines, low-degree polynomials) are rare. If we draw a function at random, it is almost certainly complex. So if our data fits a simple function, that is surprising—and surprises are evidence.

Another answer: Occam's razor is not a fact about nature but a fact about inference. Given finite data, we cannot distinguish among all models. We must choose, and choosing the simplest is the only strategy that does not overcommit. Simplicity is not about truth; it is about humility.

Perhaps nature is simple. Perhaps simplicity is just humility. MDL works either way.

A third answer: nature really is simple. The laws of physics fit on a t-shirt. The universe runs on elegant mathematics. This might be a deep fact about reality, or it might be selection bias (we notice the parts we can understand), or it might be both.

MDL does not resolve these philosophical puzzles. But it provides a framework in which they can be precisely stated. Whatever simplicity means, MDL measures it in bits.

12.9 Looking Forward

We have transformed Occam's razor into a calculation. The best model is the one that compresses data most. Compression equals prediction. Prediction equals understanding.

This principle—that learning and compression are one—has profound implications for what can be learned from data. If a model cannot compress the data, it has learned nothing. If it compresses perfectly, it has understood completely. Between these extremes lies the practical problem of learning: how much can we compress, and how much data do we need?

MDL tells us which model is best. The next question: how much data do we need to learn reliably? Chapter 13 develops information-theoretic bounds on learning.

In Chapter 11, we saw that mutual information measures expected information gain. Now we see that MDL measures which model best captures that information. The two perspectives fit together: mutual information is the currency of learning, and MDL is the budget.

But we have not yet asked the hardest question. Given a learning problem, how much data is enough? What are the fundamental limits on what can be learned? These are not philosophical questions but mathematical ones, and information theory provides answers.

Chapter 13 develops these learning bounds. We will see that overfitting is not just a practical nuisance but an information-theoretic necessity: any learner that tries to extract more bits than the data contains will fail. We will connect MDL to PAC learning, to the information bottleneck, to the deep question of what machine learning can and cannot achieve.

The tools are now in place. We have entropy, mutual information, KL divergence, maximum entropy, and MDL. These are not separate ideas but facets of one framework. The next chapter puts them to work on the ultimate question: what can be learned?

Historical note. The Minimum Description Length principle was developed primarily by Jorma Rissanen, beginning in the 1970s. Rissanen, an engineer at IBM, was motivated by problems in data compression and automatic model selection. His key insight was that compression and inference are the same problem in different clothes. The connection to Bayesian inference was recognized early, and the equivalence between description lengths and negative log probabilities is now understood as fundamental. Kolmogorov complexity, which predates MDL, was developed independently by Solomonoff, Kolmogorov, and Chaitin in the 1960s. The incomputability of Kolmogorov complexity was established by Kolmogorov and remains one of the central results connecting information theory to computability theory. MDL can be seen as making Kolmogorov's ideas practical: we compute description lengths in tractable model classes rather than over all possible programs.

13
Information-Theoretic Limits on Learning

Suppose I give you a million data points. You train the most sophisticated algorithm known to science. You tune every hyperparameter. You achieve perfect accuracy on your training data.

Then I show you new data, and your predictions fail.

This is not a hypothetical. It happens constantly. The question is not whether it happens, but whether we could have known in advance.

Not because your algorithm was poorly chosen. Not because you made implementation errors. But because no algorithm, however clever, could have done better. The data simply did not contain enough information to solve your problem.

This is the situation we confront in this chapter: the fundamental limits on what can be learned from data. These limits are not about computational resources—give your algorithm unlimited time and memory, and it will still fail. They are not about our cleverness—Einstein himself could not transcend them. They are limits imposed by information theory, as fundamental as the speed of light or the second law of thermodynamics.

You might say, "But machine learning works! Deep networks achieve remarkable performance! Surely these limits cannot be too restrictive."

We shall see. The limits are real, but they are not where the naive observer might expect them. Understanding where the limits lie—and where they do not—is essential for knowing what we can and cannot ask of our learning systems.

13.1 The Communication Channel of Experience

Let us begin with a metaphor that will guide us through this chapter.

Learning from data is communication through a noisy channel. The "sender" is the true pattern. The "channel" is the sampling process. The "receiver" is our model.

Learning from data is a communication problem. There is a true pattern in the world—perhaps a function relating inputs to outputs, or a distribution governing future events. This pattern is the "sender." You observe samples from this pattern, corrupted by noise, selection bias, finite precision. The sampling process is the "channel." Your learning algorithm constructs a model from these samples. The algorithm is the "receiver."

Shannon's channel capacity theorem tells us that reliable communication is possible up to capacity—and impossible beyond it. No matter how clever your decoder, you cannot recover more signal than the channel transmitted.

The same logic applies to learning. No matter how clever your algorithm, you cannot learn more than your data contains. If the data carries only 50 bits of information about the true pattern, you cannot learn 100 bits worth of pattern. You will fill the gap with noise, mistaking the static for the message.

This is what overfitting *is*: decoding more than was transmitted. The extra "information" you extract is not information at all—it is artifacts of the particular sample you observed, patterns that will not appear in future samples because they never existed in the true distribution.

The Fundamental Inequality

Consider learning a function $f : X \to Y$ from n training examples. How many bits does f require to specify? If f is complex—say, a function that makes a different decision for each of 2^{100} possible inputs—it might require 100 bits or more.

But how many bits do the training examples contain about f? Each example (x_i, y_i) provides at most $\log |Y|$ bits (the entropy of the output). With n examples, you get at most $n \log |Y|$ bits.

If f requires 100 bits to specify and your training data contains only 50 bits about it, you face an impossible situation. The data cannot uniquely determine f. Many functions are consistent with your observations. Any algorithm must choose among them, and that choice must come from somewhere other than the data—from prior assumptions, regularization, architectural biases.

If specifying f requires more bits than the data provides, the data cannot determine f. Some of what you learn must come from assumptions, not evidence.

This is not a failure of any particular algorithm. It is a mathematical necessity. Information cannot be created from nothing.

The Role of Assumptions

You might object: "But algorithms *do* learn useful things from limited data! They must be doing something right."

They are. They bring assumptions to the table. A linear regression assumes the true function is linear—if correct, this assumption provides the missing bits. A neural network architecture embodies assumptions about what kinds of functions are likely. A prior distribution in Bayesian inference explicitly encodes beliefs about which hypotheses are plausible.

Every learning algorithm embodies assumptions. The question is not whether to assume, but what to assume. Information theory does not tell us what to assume; it tells us that we must.

These assumptions are not cheating. They are necessary. Without them, learning is impossible. The question is not whether to make assumptions, but which assumptions to make, and how much they help.

Information theory does not prescribe assumptions. It measures them. A prior that assigns probability p to hypothesis h provides $-\log p$ bits toward specifying h. The tighter your prior, the fewer bits you need from data. The looser your prior, the more you demand from observations.

13.2 The Information Bottleneck

Consider a concrete learning task. You have inputs X—perhaps images—and outputs Y—perhaps labels ("cat," "dog," "truck"). You want to learn a representation T of the inputs that is useful for predicting the outputs.

What should T look like?

Two Desiderata

You might want T to preserve everything about X. After all, who knows what might be relevant? But this leads to overfitting. If $T = X$, you have learned nothing—you are just passing raw inputs through, including all their noise and irrelevant details.

The information bottleneck formalizes a fundamental tradeoff: retain what matters for prediction, discard what does not.

You might want T to be as simple as possible. Simpler representations generalize better. But if T is too simple, it throws away information needed to predict Y.

The information bottleneck, introduced by Naftali Tishby and colleagues, formalizes this tradeoff:

1. T should retain as much information about Y as possible: $I(T;Y)$ should be large.

2. T should discard as much information about X as possible: $I(X;T)$ should be small.

These goals conflict. A representation that discards X-information may discard Y-relevant information too. The question is: how much can we compress X while retaining what matters for Y?

The Objective Function

The information bottleneck objective is:

$$\min_{T} \quad I(X;T) - \beta\, I(T;Y)$$

where $\beta > 0$ controls the tradeoff.

The parameter β controls how much we care about prediction versus compression. Different β values trace out the optimal tradeoff curve.

At $\beta = 0$, we care only about compression. The optimal T is a constant—maximum compression, no prediction.

As $\beta \to \infty$, we care only about prediction. The optimal T is X itself—no compression, maximum prediction (to the extent X determines Y).

For intermediate β, we obtain representations that balance compression against prediction. Varying β from 0 to infinity traces out a curve of achievable $(I(X;T), I(T;Y))$ pairs.

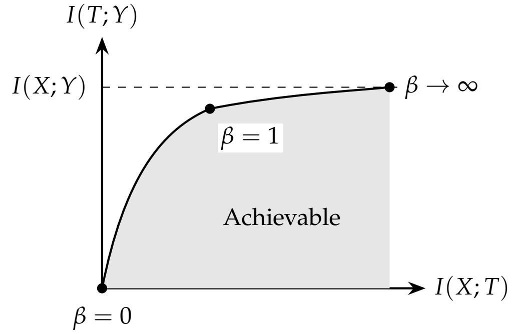


Figure 13.1: The information bottleneck curve. Points below the curve are achievable; points above are impossible.

The Information Curve

This curve is remarkable. It is not a property of any learning algorithm—it is a property of the joint distribution $P(X,Y)$ itself. No algorithm can achieve points above the curve. Every algorithm, if optimal, achieves points on the curve.

The curve tells us the fundamental tradeoff for this particular learning problem. Some problems have curves that rise steeply—a little compression costs a lot of prediction. Other problems have curves that stay flat—massive compression is possible with minimal loss. The curve reveals the structure of the problem.

What This Means for Generalization

A representation with small $I(X;T)$ and large $I(T;Y)$ will generalize well. It has extracted what matters for prediction and discarded what does not.

Good representations compress the input while preserving prediction. This is exactly what generalization requires.

Why? Consider what overfitting means in this language. An overfit model has large $I(X;T)$—it remembers too much about the training inputs, including their noise and idiosyncrasies. But this extra information does not help prediction on new data. It hurts.

The information bottleneck makes this precise. The goal is not to learn everything about X. The goal is to learn what X tells us about Y—and nothing more.

Practical Challenges

You might ask: how do we compute these mutual information quantities for real data?

Mutual information is notoriously difficult to estimate for high-dimensional data. This limits direct application of the bottleneck principle, but the principle itself remains valid.

The answer is sobering. For high-dimensional data (images, text, neural network activations), estimating mutual information is extremely difficult. Different estimation methods give substantially different answers. Published claims about "mutual information in neural networks" should be interpreted cautiously.

But the principle remains valid even when exact computation is impossible. The information bottleneck tells us what to aim for. Good representations should compress; they should retain what matters; they should discard what does not. Architectures that explicitly impose bottlenecks (like the narrow hidden layer in an autoencoder) are trying to enforce this principle, whether or not they optimize the exact objective.

13.3 PAC-Bayesian Bounds

The information bottleneck asks: what representations should we learn? We now ask a different question: how well can any learned hypothesis generalize?

The Setup

We observe n training examples $(x_1, y_1), \ldots, (x_n, y_n)$ drawn independently from some unknown distribution D. We learn a hypothesis h from some class H. We want to know the true error:

$$L(h) = \Pr_{(x,y)\sim D}[h(x) \neq y]$$

What we observe is the training error:

$$\hat{L}(h) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}[h(x_i) \neq y_i]$$

The true error is what we care about. The training error is what we observe. The gap between them is the generalization gap.

The generalization gap $L(h) - \hat{L}(h)$ measures how much worse we do on new data than on training data. We want this gap to be small.

Classical Bounds

Classical learning theory bounds the generalization gap in terms of the "complexity" of the hypothesis class H—its VC dimension or Rademacher complexity. These bounds apply uniformly to all hypotheses in the class.

But such bounds can be loose. They penalize complexity of the entire class, even if the algorithm selects a simple hypothesis from a complex class.

The PAC-Bayesian Approach

PAC-Bayesian bounds take a different approach. Instead of penalizing the class, they penalize the algorithm's dependence on the training data.

The setup: before seeing any data, we specify a prior distribution P over hypotheses. This represents what we believe before observing anything. After seeing data, we compute a posterior distribution Q. This represents what we believe after learning.

PAC-Bayes bounds measure how much the learning algorithm changed its beliefs after seeing data. Large changes are penalized; small changes generalize safely.

McAllester's PAC-Bayes theorem states: with high probability over the training sample,

$$\mathbb{E}_{h\sim Q}[L(h)] \leq \mathbb{E}_{h\sim Q}[\hat{L}(h)] + \sqrt{\frac{\mathrm{KL}(Q\|P) + \ln(2n/\delta)}{2n}}$$

Interpreting the Bound

The bound says: expected true error is at most expected training error plus a penalty term.

The penalty depends on $\mathrm{KL}(Q\|P)$—how much the posterior differs from the prior. If you change your beliefs dramatically after seeing data, the KL divergence is large, and the bound is loose. If you change only slightly, the bound is tight.

This is intuitive. If the training data convinced you of something radically different from your prior beliefs, one of two things happened. Either the data contained strong evidence (good), or you overreacted to noise (bad). The bound does not distinguish—it simply says that large changes carry risk.

The PAC-Bayes message: you can only generalize safely if you do not deviate too far from your prior beliefs. Wild changes in belief may fit training data but fail on new data.

The penalty also shrinks with n. More data justifies larger changes in belief. With abundant evidence, you can safely move far from your prior.

The Information-Theoretic View

The KL divergence $\mathrm{KL}(Q\|P)$ measures the information gained about the hypothesis from the training data:

$$\mathrm{KL}(Q\|P) \approx I(h;S)$$

where S denotes the training sample and h the learned hypothesis.

Generalization gap is bounded by mutual information between hypothesis and training data. You cannot generalize better than the information content allows.

This reveals the information-theoretic nature of PAC-Bayes bounds. The generalization gap is bounded by how much information the algorithm extracts from the training data. Algorithms that extract less information generalize better.

This explains why regularization helps: it constrains the algorithm, reducing the information it can extract. It explains why early stopping helps: cutting off learning before the algorithm fully fits the data limits information extraction. It explains why dropout helps: randomizing the training process reduces the dependence between hypothesis and data.

These techniques work not because of ad-hoc intuitions but because they reduce $I(h;S)$.

13.4 Generalization Through an Information Lens

Let us now step back and ask: why does overfitting happen at all?

Signal and Noise

Every training sample contains signal and noise. The signal is the true pattern that will recur in future data. The noise is the idiosyncratic variation of this particular sample.

Training data = signal + noise. Overfitting is learning the noise as if it were signal.

A model that fits the training data perfectly has captured both. It cannot distinguish signal from noise—both are present in the data, both influence the fit.

But on new data, only the signal recurs. The noise is different. The model's predictions based on the old noise fail to match the new noise.

This is overfitting. It is not a bug in our algorithms. It is an inevitable consequence of learning from finite samples.

The Information Perspective

In information-theoretic terms:

- $I(\text{model}; \text{signal})$ is what we want—information about the true pattern
- $I(\text{model}; \text{noise})$ is what kills generalization—information about this sample's idiosyncrasies

The model cannot distinguish signal from noise in the training data. All it sees is data. This is the tragedy of finite samples.

Fitting more tightly increases both. The model cannot tell which is which. All it sees is data.

The signal-to-noise ratio of the training data limits how much signal we can extract. If noise dominates, even perfect learning extracts more noise than signal. If signal dominates, aggressive fitting is safe.

The Bias-Variance Tradeoff

The classical decomposition of prediction error is:

$$\text{Error} = \text{Bias}^2 + \text{Variance} + \text{Noise}$$

Bias is systematic error from model limitations. Variance is error from sensitivity to the training sample. Noise is irreducible error from the problem itself.

Bias comes from model limitations. Variance comes from overfitting. The tradeoff between them is forced by information constraints.

The information-theoretic interpretation:

- High bias: the model class cannot capture the truth, so $I(\text{model}; \text{signal})$ is limited by model capacity
- High variance: the model depends sensitively on the training sample, so $I(\text{model}; \text{noise})$ is high
- Irreducible noise: $I(\text{data}; \text{truth})$ is limited by the problem itself

The tradeoff is forced by information constraints. Using a more flexible model class allows higher $I(\text{model}; \text{signal})$, but also higher $I(\text{model}; \text{noise})$. You cannot increase one without risking the other.

13.5 Sample Complexity

We now ask quantitatively: how many examples do we need to learn?

The Definition

The sample complexity of learning a concept to accuracy ϵ with confidence $1 - \delta$ is the minimum number of examples $n(\epsilon, \delta)$ required.

This depends on the concept being learned, the hypothesis class, and the learning algorithm. But information theory provides lower bounds that no algorithm can beat.

Sample complexity asks: how much data is enough? Information theory provides lower bounds that no algorithm can beat.

Information-Theoretic Lower Bounds

Consider learning a target hypothesis h^* from a class H. The hypothesis is unknown; we must identify it from training data.

Each training example provides some number of bits about h^*. This depends on the problem: in a noiseless binary classification task, each example provides 1 bit. In a noisy task, each example provides less.

To uniquely identify h^* from H, we need at least $\log |H|$ bits of information. If each example provides at most c bits, we need at least $n \geq \log |H| / c$ examples.

This is a fundamental limit. No algorithm can learn faster. Clever algorithms can approach this limit; none can beat it.

You need at least $\log |H|$ bits of information to identify a hypothesis from class H. This is not a worst case—it is a lower bound.

Fano's Inequality

Fano's inequality makes this precise. If we observe data S and try to identify hypothesis h^*:

$$P(\text{error}) \geq \frac{H(h^* \mid S) - 1}{\log |H|}$$

If the conditional entropy $H(h^* \mid S)$ is large—if the data leaves substantial uncertainty about h^*—then the probability of error is bounded away from zero. No algorithm can do well.

Fano's inequality is the information-theoretic version of "you can't get blood from a stone." If the data does not distinguish among hypotheses, no algorithm can distinguish among them either.

Fano's inequality: if the data leaves too much uncertainty about the true hypothesis, no algorithm can identify it reliably.

A Worked Example

Consider learning an unknown binary string $b \in \{0, 1\}^k$. Each example is a noisy observation of one bit, correct with probability $p > 1/2$.

The target has k bits. Each example provides $1 - H_2(p)$ bits of information, where $H_2(p) = -p \log p - (1 - p) \log(1 - p)$ is the binary entropy.

p	Bits/example	Examples for $k = 100$
0.99	0.92	109
0.90	0.53	189
0.75	0.19	526
0.60	0.03	3333

Table 13.1: Sample complexity for learning a 100-bit string from noisy observations. As noise increases (p decreases toward 0.5), required samples grow dramatically.

To learn all k bits reliably, we need approximately:

$$n \geq \frac{k}{1 - H_2(p)}$$

examples. As $p \to 1/2$ (pure noise), the denominator approaches zero and the required samples explode to infinity. This is correct: random noise contains no information about the target.

13.6 Deep Learning and the Information Plane

Modern deep learning has achieved remarkable successes. Can information theory explain what is happening?

The Deep Network as a Markov Chain

A deep neural network processes data through layers:

$$X \to T_1 \to T_2 \to \cdots \to T_L \to \hat{Y}$$

Each layer T_ℓ depends only on the previous layer. This is a Markov chain. By the data processing inequality:

$$I(X; T_1) \geq I(X; T_2) \geq \cdots \geq I(X; T_L)$$

Each layer of a deep network forms a Markov chain. By the data processing inequality, information about X can only decrease as we go deeper.

Information about X can only decrease as we go deeper. Each layer compresses the representation.

If the network is doing its job, it should also maintain $I(T_\ell; Y)$—the information about the target should be preserved even as irrelevant information is discarded.

The Information Plane Hypothesis

Tishby and colleagues proposed visualizing learning in the "information plane": a 2D space with $I(X; T)$ on one axis and $I(T; Y)$ on the other.

The claim: neural networks during training first increase $I(X; T)$ (the fitting phase), then decrease it while maintaining $I(T; Y)$ (the compression phase). Learning proceeds in two distinct phases.

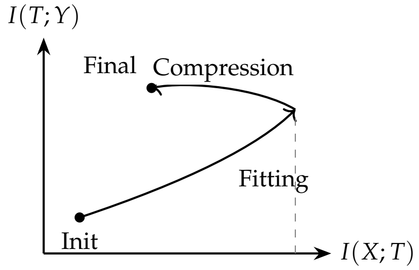


Figure 13.2: The information plane hypothesis: learning proceeds in two phases. First, fitting increases $I(X; T)$. Then compression decreases $I(X; T)$ while maintaining $I(T; Y)$.

The Controversy

This claim is controversial. Subsequent research has shown:

- The compression phase may be an artifact of how mutual information is estimated
- Networks with ReLU activations do not show the same pattern as networks with tanh

- The observed dynamics depend sensitively on architecture, data, and estimation method

What can we say with confidence? Less than the original claims, but still something important:

Whether deep networks actually follow the two-phase trajectory is disputed. The principle that good representations compress may be true even if the specific dynamics are not universal.

Layers do compress. Even if the dynamics are not a clean two-phase process, deeper layers generally have lower $I(X;T)$ than earlier layers. The network is discarding information.

Compression correlates with generalization. Networks that achieve better generalization tend to have representations with lower $I(X;T)$ for comparable $I(T;Y)$. Whether this is cause or effect is unclear.

The principle is sound even if the dynamics are debated. The information bottleneck tells us what good representations should look like, even if we cannot reliably measure whether any particular network achieves them.

Grokking

A recent observation adds to the puzzle. In some cases, neural networks achieve perfect training accuracy quickly, then—much later—suddenly achieve perfect test accuracy as well. This phenomenon is called "grokking."

Grokking: perfect training accuracy, then a long plateau, then suddenly perfect test accuracy. What happens during the plateau?

What happens during the plateau? The network has already fit the training data. Its predictions are not changing. Yet something is happening internally that eventually leads to generalization.

One hypothesis: the network is compressing its representation. The weights are rearranging to find a simpler solution that achieves the same training accuracy with less reliance on the specific training examples. When compression reaches a threshold, generalization emerges.

If true, this supports the view that compression causes generalization. Fitting is fast—gradient descent rapidly reduces training loss. Compression is slow—the network must find simpler representations that achieve the same predictions.

But this is speculative. We do not yet fully understand grokking.

13.7 *The Double Descent Puzzle*

Classical learning theory predicts that test error should follow a U-curve as model complexity increases: underfitting at low complexity, good fit in the middle, overfitting at high complexity.

Classical theory: error decreases, then increases with complexity. Reality: error decreases, increases, then decreases again. This is double descent.

Modern deep learning violates this prediction. As model complexity increases past a critical point—where the model can exactly fit the training data—test error starts *decreasing* again. This is "double descent."

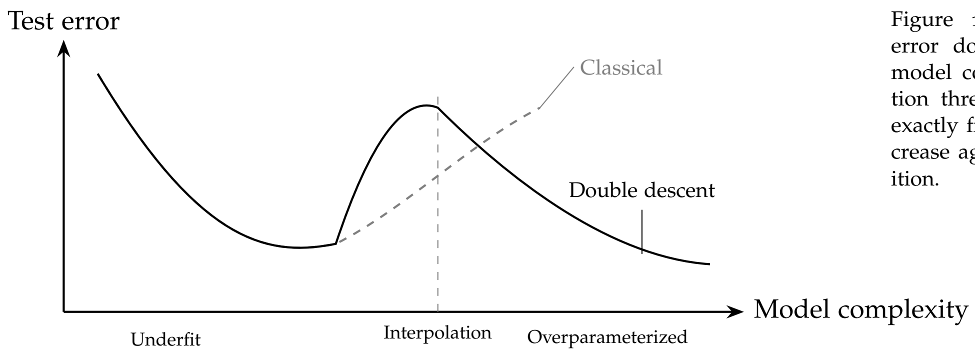


Figure 13.3: Double descent: test error does not simply increase with model complexity. Past the interpolation threshold—where the model can exactly fit training data—error may decrease again. This defies classical intuition.

How can overparameterized models generalize well? They have far more parameters than training examples. By classical reasoning, they should massively overfit.

The Information-Theoretic Resolution

The resolution is subtle. What matters is not parameter count but *effective information content*.

Parameter count is not the same as effective complexity. A neural network with millions of parameters may have far lower effective description length.

A neural network with 100 million parameters, trained with stochastic gradient descent and early stopping, does not encode 100 million independent pieces of information. The training dynamics constrain the solution. The implicit regularization of SGD biases toward simple solutions. The architecture limits what kinds of functions can be represented efficiently.

The *effective description length* of a well-generalizing network is much shorter than its parameter count suggests. MDL tells us that generalization depends on description length, not parameter count. Double descent becomes less mysterious: past the interpolation threshold, the effective complexity may actually *decrease* as parameter count increases.

The Lottery Ticket Hypothesis

This connects to the "lottery ticket hypothesis": large neural networks contain small subnetworks that can achieve the same test accuracy when trained in isolation.

The lottery ticket hypothesis: inside a large network is a small network that does all the work. The large network helps find it; the small network is what generalizes.

The large network's purpose is not to encode a complex function. It is to provide enough "tickets"—enough random initializations—that one of them happens to be a good small network. Training finds and amplifies this subnetwork.

If true, this explains why large networks generalize: they are effectively small. The extra parameters are scaffolding for the search process, not load-bearing parts of the final solution.

13.8 What Cannot Be Learned

We close by asking what is fundamentally unlearnable.

Random Functions

If the target function is random—if outputs have no systematic relationship to inputs—then no algorithm can learn it.

This is obvious, but worth stating precisely. The training data provides no information about future outputs. Every example is independent. The mutual information $I(\text{training data}; \text{test outputs})$ is zero.

No algorithm, however clever, can extract information that does not exist.

Random functions are unlearnable in principle. This is not a limitation of our methods; it is a fact about the problem.

Noisy Labels

If training labels are corrupted with probability ϵ, you cannot achieve better than ϵ error on those examples.

The noise destroys information. If 30% of your labels are wrong, you are learning from a dataset that is 30% lies. No algorithm can distinguish the lies from the truth.

This does not mean learning is hopeless. If the noise is random (not adversarial), and if the true pattern is strong, you can still learn a useful model. But you cannot achieve perfect accuracy. The noise sets a floor.

Noisy labels set a floor on achievable error. This floor cannot be lowered by algorithmic cleverness.

Insufficient Features

If the features do not contain information about the target, no algorithm can predict the target.

This sounds obvious, but it is often forgotten. If you try to predict stock prices from weather data, and prices are not influenced by weather, you will fail—not because your algorithm is bad, but because the problem is impossible.

Information theory quantifies this. The mutual information $I(X; Y)$ between features X and target Y bounds predictive performance. If this mutual information is low, even the best algorithm achieves low accuracy.

If the features contain no information about the target, learning is impossible. No algorithm can create information from nothing.

What This Does Not Mean

These limits do not imply pessimism about machine learning. They tell us what is impossible; they also tell us what is possible.

Humans face the same limits. We cannot learn from data that contains no information. We cannot achieve perfect accuracy with noisy labels. We are bounded by the same information-theoretic constraints.

Humans face the same limits. Information theory does not say learning is impossible; it says what is possible and what is not.

Yet humans achieve remarkable intelligence. Within the limits, extraordinary things are possible. Information theory does not forbid artificial intelligence; it clarifies what artificial intelligence can and cannot do.

13.9 *Looking Forward*

We have traced information theory from Shannon's original question—how to communicate reliably—through compression, physics, inference, and now learning. The same mathematical structures appear in wildly different contexts.

Overfitting is not a flaw in our algorithms. It is an information-theoretic necessity. Finite data contains finite information; we cannot extract more than is present.

The deep insight of this chapter: overfitting is not a bug in our algorithms. It is the inevitable consequence of trying to learn from finite data. When we memorize noise, we are not making a mistake—we are encountering a fundamental limit on what data can tell us.

The limits are real, but they are not discouraging. They tell us what to aim for. A system that approaches the information-theoretic limits is doing as well as possible. We are not there yet—there is room to improve.

And there is something beautiful in the unity we have discovered. Compression and prediction are the same thing. Learning and communication are the same thing. The model that compresses data most is the model that understands it best.

We began asking how to communicate. We end asking what can be known. Information theory answers both: what can be communicated reliably, and what can be learned reliably, up to limits set by the information the channel carries.

We began this book with Shannon's question: how do we communicate reliably over noisy channels? We end with the same question in a different form: how do we learn reliably from noisy experience? Information theory gives the same answer to both: we can, but only within limits set by the information the channel carries.

In our final chapter, we step back to survey the entire landscape. We return to the question we began with—"What is information?"—now armed with everything we have learned. We ask what information theory tells us about physics, computation, and knowledge itself. And we point toward frontiers where the story continues.

Historical note. The information bottleneck was introduced by Naftali Tishby, Fernando Pereira, and William Bialek in 1999, though it builds on earlier work by Tishby and others. Tishby (1952–2021) spent two decades developing connections between information theory and learning. His 2015 work with Ravid Shwartz-Ziv proposing the "information plane" interpretation of deep learning generated both excitement and controversy. PAC learning was introduced by Leslie Valiant in 1984, work that contributed to his Turing Award. The PAC-Bayesian extension, connecting learning theory to information measures, developed through the 1990s and 2000s, with key contributions from McAllester, Langford, Shamir, and others. The double descent phenomenon was documented systematically by Belkin and colleagues in 2019, though hints of it appeared earlier. The lottery ticket hypothesis was proposed by Frankle and Carlin in 2019. These developments show that the connection between information theory and machine learning is not a historical curiosity but an active research frontier.