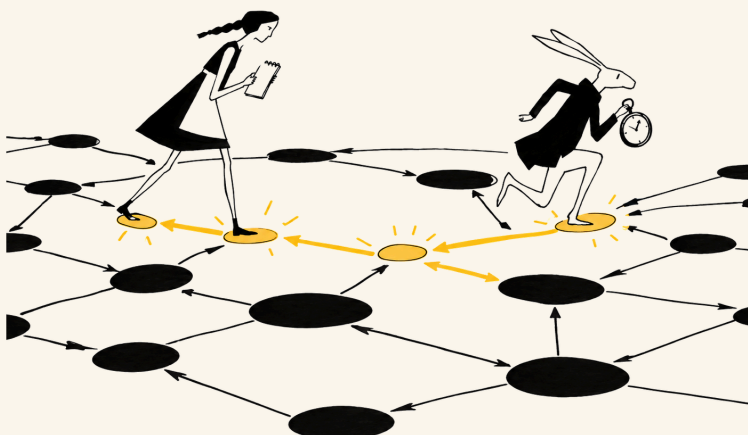


Alice's Adventures in a Differentiable Wonderland



Simone Scardapane · 2024

Illustrated audio lectures

Lecturized

Alice's Adventures in a Differentiable Wonderland

Illustrated audio lectures

Simone Scardapane · 2024

Retold as illustrated lectures by Lecturized
lecturized.com

From the original: Alice's Adventures in a Differentiable Wonderland, Volume I: A Tour
of the Land

Contents

One Simple Recipe, Scaled Up

Arrows That Recognise Each Other

How Costs Grow, and Which Way Is Uphill

Walking Downhill in the Fog

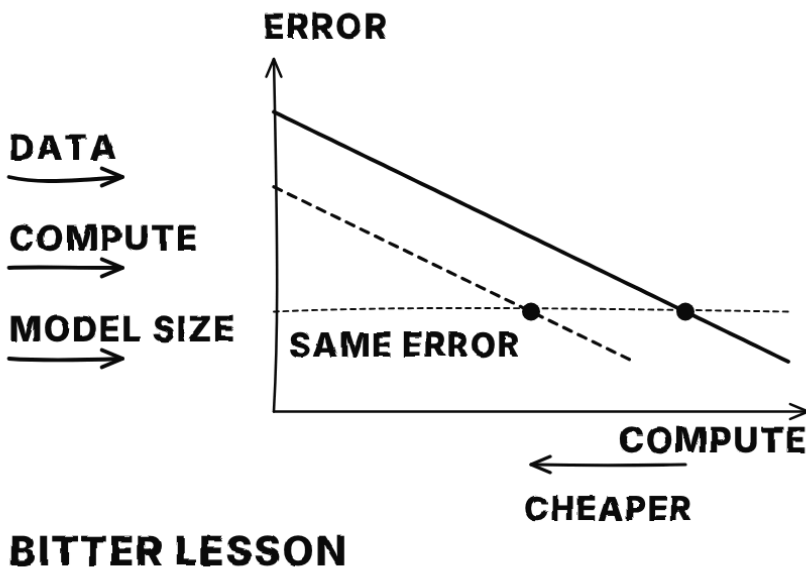
Momentum, and Where the Examples Come From

A Score to Lower, and a Test You Have Not Seen

Many Possible Models, and the Simplest One

One Simple Recipe, Scaled Up

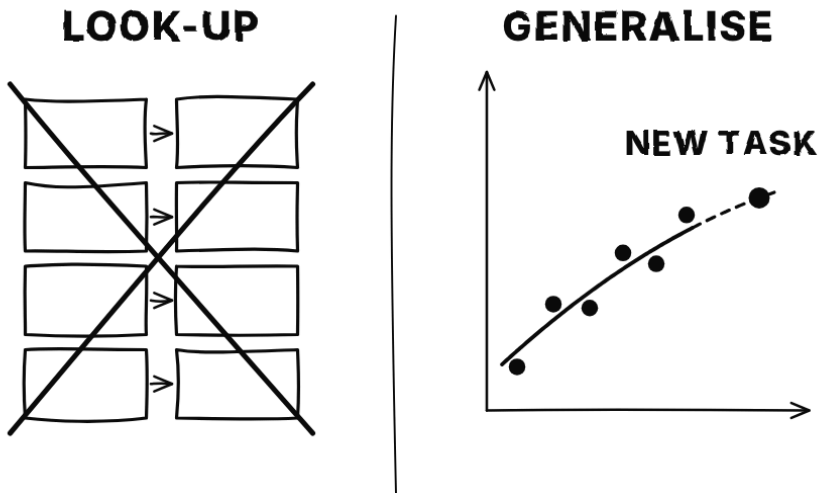
Open the chat assistant on your phone and ask it something it has surely never seen word for word, say, a limerick about your neighbour's cat and a tax form. You probably expect a clumsy patchwork of sentences copied from somewhere. What comes back is new, fitting, and often funny. Something inside that program has learned far more than a list of stored answers, and the book opens by asking how such a thing is built. Its answer is surprising: from a very small set of principles and parts, used over and over, and then made enormous.



Scaling Makes Error Fall

The book calls this the power of scaling. Researchers found that for almost any task, adding more data, more computing power and a bigger model together makes accuracy rise in a predictable way. Said the other way round, the computing needed to reach a given accuracy keeps shrinking year after year. The scientist Richard Sutton called this the bitter lesson: simple, general tools plus ever more computation beat clever special tricks.

Here is the puzzle that makes it almost magical. These networks are so large that they could simply memorise everything they are shown and become a giant look-up table. Instead, trained models generalise: they handle tasks that were never spelled out in their training data, and the bigger the data, the less they lean on pure memorising. That is why your assistant can write the cat limerick at all.

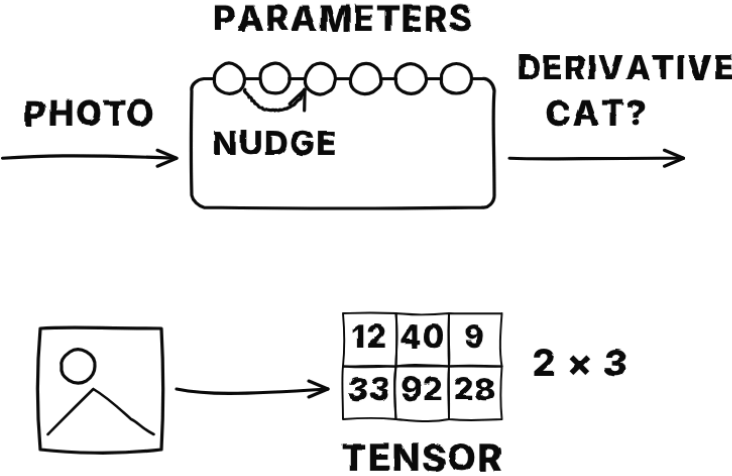


Memorise or Generalise

The book also notes how our way of using these models has changed. Huge pre-trained models, called foundation models, can be steered simply by prompting them with text or pictures, while their insides stay a black box. Only a few teams in the world can build the largest ones, much as only a few companies can build huge business software. Yet with ordinary hardware and some ingenuity you can still fine-tune, merge, shrink or test such models, and for that you need to look under the hood.

Looking under the hood starts with one idea: learning is a search. You write a program with a vast number of adjustable knobs, called parameters, and you turn them until the program behaves well. A photo app that tells cats from dogs is trained exactly like this: show it

thousands of labelled pictures and nudge its knobs after every mistake. To search among millions or billions of knobs, the book restricts itself to parts whose effect can be measured with derivatives, which tell you what happens if a knob is nudged slightly. That is why it prefers the name differentiable models to neural networks, and why it builds everything from automatic ways of computing derivatives and stepping downhill.



Learning Is a Search

Differentiable models cannot work directly with letters or whole numbers, which jump rather than vary smoothly, so all data is turned into large arrays of numbers that the book calls tensors. An image, a sound or a sentence all become such arrays, and every part of a model is classed by what shape of array it takes in and gives out. The rest of

the book then visits convolutions for images, sound and text, attention and transformers, graphs, and recurrent models.

None of these ideas is new. Training simple models by gradient descent goes back to the nineteenth century, and in 1958 the psychologist Frank Rosenblatt already drew newspaper hype with his perceptrons. What changed was data and power: the image contests of 2012 to 2017 and the arrival of transformers in 2017 showed how well these old ideas work at scale. Left out here are the book's plan chapter by chapter and its notation for tensors and their shapes; they are in the sections «About this book» and «Notation and symbols».

Remember

Almost everything in modern AI is built from a few simple principles and parts, made powerful by scaling data, computing and model size together. Large models do not just memorise; they generalise to tasks they were never shown, which is the magic the book wants to explain. Learning is a search: a program with a huge number of adjustable parameters is tuned until it behaves well. The search works because every part is differentiable, so derivatives tell how to nudge each parameter, which is why the book speaks of differentiable models. All data, from photos to sentences, is handled as large arrays of numbers called tensors.

Questions

1. According to scaling laws, what usually happens when data, computing power and model size are all increased together?
 - a. Accuracy rises in a predictable way, for almost any task
 - b. Accuracy rises only for image tasks, not for language
 - c. The model starts to memorise and gets worse on new tasks
 - d. Nothing changes unless the model's design is rethought by hand
2. Why is it surprising that large trained models handle tasks they were never shown?
 - a. Because they are too small to hold any examples at all
 - b. Because they are large enough to simply memorise their training data like a look-up table, yet they generalise instead
 - c. Because they were trained only on numbers, never on text or pictures
 - d. Because nobody can prompt them once they are trained
3. Why does the book build its models only from differentiable parts?
 - a. Because differentiable parts are cheaper to store in memory
 - b. Because only differentiable parts can be prompted with text
 - c. Because derivatives tell how to nudge each of the millions of parameters, which makes the search for good settings practical
 - d. Because differentiable parts never need any data to be trained
4. Why must a sentence or a photo be turned into large arrays of numbers before a differentiable model can use it?

- a. Because letters and whole numbers jump instead of varying smoothly, so derivatives cannot work with them directly
- b. Because arrays of numbers take less space than letters
- c. Because the book's notation forbids letters
- d. Because only photos can be stored as letters

Answers

1. **a)** Researchers found that adding more data, more computing power and a bigger model together makes accuracy rise in a predictable way for almost any task.
2. **b)** The networks are so large that they could memorise everything they see and become a giant look-up table; instead they generalise, and lean less on memorising as the data grows.
3. **c)** Learning is a search over a huge number of parameters, and derivatives say what happens when a knob is nudged slightly, so they guide the search automatically.
4. **a)** Differentiable models cannot work directly with letters or whole numbers, which jump rather than vary smoothly, so all data becomes arrays of numbers called tensors.

Practice

1. Think of a photo app that learns to tell cats from dogs. Explain in your own words what it means to say that learning is a search, and what the parameters are in this example.

Hint: Picture the app as a machine with a huge panel of knobs, and ask what training does to them.

Answer: The app is a program with a vast number of adjustable knobs, its parameters. Training shows it labelled pictures and turns the knobs a

little after each mistake, searching among all possible knob settings for one that makes the app sort cats from dogs well.

2. The ideas behind today's AI, such as gradient descent and the perceptron, are old. Explain, as the part does, why they only became so powerful in the last fifteen years or so.

Hint: Ask what was missing in 1958 that is plentiful today.

Answer: The methods were known long ago, but there was not enough data or computing power to show how well they work. Large image contests from 2012 and transformers from 2017 showed that, given enough data and parameters, these old ideas perform remarkably well.

3. Many people now use a foundation model only by prompting it, as a black box. Explain why someone might still want to understand what happens inside it.

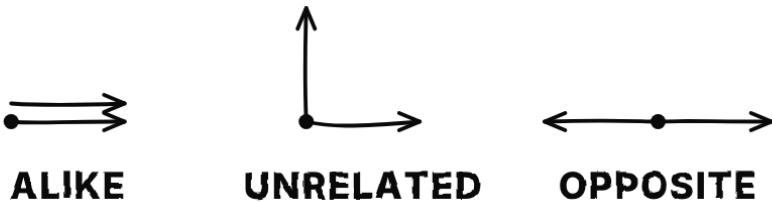
Hint: Think of what you could do with an open program beyond simply running it.

Answer: Prompting is enough to use the model, but adapting it needs more: fine-tuning it for your own task, merging models, shrinking them for weak hardware or testing how robust they are all require looking under the hood at how the model handles data inside.

Arrows That Recognise Each Other

Take two pencils and lay them on the table, both starting from the same point. Point them the same way, then at a right angle, then in opposite directions, and ask each time how alike the two directions are. You can feel the answer without any arithmetic: same way means very alike, right angle means unrelated, opposite means as unlike as possible. That simple feeling of alike or unlike turns out to be the main tool of the chapter on mathematics, and modern AI uses it millions of times a second.

DOT PRODUCT

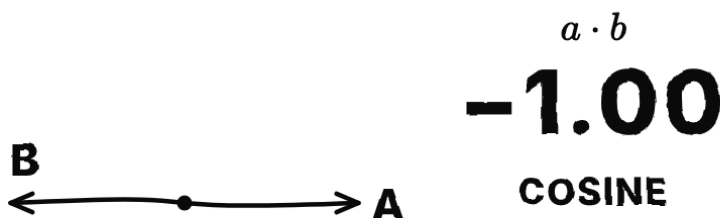


Same, Right Angle, Opposite

Before the mathematics, the book says where it comes from and what it is not. It grew out of a course on neural networks that the author teaches in a master's programme in data science at Sapienza University of Rome, and it aims to add its own viewpoint to other good books on the subject. It stresses that theory and code go hand in hand, and that the boom in neural networks since about 2012 owes much to software libraries such as TensorFlow, PyTorch and JAX. It is not a programming manual, it does not try to cover everything, and it says little about hardware or the craft of making huge models work in practice. It is also released under a free licence that allows anyone to remix and build on it, which is exactly what makes a course like this one possible.

The chapter on mathematics then sets out the language for everything that follows. As we saw, all data lives in tensors, arrays of numbers described by their shape, and they suit the graphics chips that run many simple sums side by side. A single row of numbers is a vector, which you can picture as an arrow from the origin, like your pencils. Two arrows can be stretched and added to give a third, and every arrow has a length.

$$a \cdot b = a_1b_1 + a_2b_2$$



The Dot Product as a Dial

The key operation is the dot product: multiply two vectors number by number and add up the results, which gives a single number. That number is large when the arrows point the same way, zero when they are at a right angle, and negative when they point in opposite directions, just as your pencils showed. For arrows of length one it measures exactly how alike their directions are, which is why it is called cosine similarity. The book draws the lesson that matters most: the dot product is largest when the second arrow matches the first one. In the chapters ahead, one arrow will be the input and the other the model's adjustable parameters, so the model responds most strongly when the input resonates with what its parameters have learned, a kind of template matching. When your photo app finds all

the pictures that look like your dog, a comparison of this kind is what measures how alike two pictures are.

Try it yourself

Take two pencils and hold them at the same starting point. Turn one of them slowly from pointing along the other to pointing the opposite way. You should feel the match drop from complete, through nothing at the right angle, to a complete mismatch at the end, which is the same journey the dot product makes.

A table of numbers, a matrix, can be read as a stack of such vectors, one per row, and the book calls this a batch of data. Multiplying matrices is then nothing more than computing many dot products at once, every row of one against every column of the other, in a single operation. When a chat assistant processes a whole batch of words at once, it is doing exactly this kind of mass comparison. There is also a simpler, number-by-number product used for masking and scaling, and functions such as the exponential are usually applied to every number of a table separately, which is different from the special matrix versions found in textbooks. The book warns of a common pitfall in code: libraries quietly stretch arrays of mismatched shapes to fit, and two vectors can turn into a whole table by accident.

BATCH

1	2	0
0	1	3

 $\times$

1	0
1	1
0	2

 $=$

3	2
1	7

--	--	--

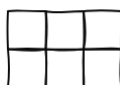
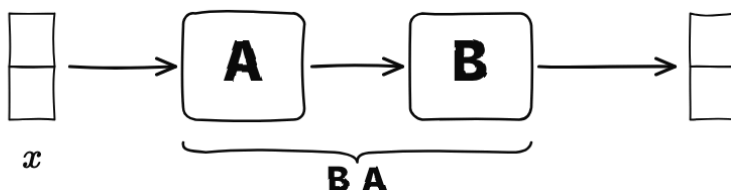
 $+$

 $=$

BROADCAST!

Many Dot Products at Once

Finally the book explains why matrix multiplication is defined the way it is. Every function that respects adding and stretching, called linear, can be written as a matrix, so multiplying a matrix by a vector means applying the function, and multiplying two matrices means applying one function after the other. Left out here are the exact formulas for lengths, dot products, matrix products and indexing; they are in the sections «Common vector operations» and «Common matrix operations».



STACK OF ROWS

Matrices as Linear Functions

Remember

The book grew from a university course and treats theory and code as partners, with frameworks such as PyTorch and JAX in mind, while leaving hardware and engineering aside. Data lives in tensors; a vector is an arrow, and a matrix is a stack of vectors, often a batch of examples. The dot product turns two arrows into one number that says how alike they are, and it is largest when they match, which is how a model recognises patterns it has learned. Matrix multiplication is many dot products at once, and it is also the way of chaining linear functions one after another.

Questions

1. Two arrows of length one point at a right angle to each other. What does their dot product say?
 - a. That they are as alike as possible
 - b. That they are unrelated: the dot product is zero
 - c. That they are as unlike as possible
 - d. That one of them must be longer than the other
2. Why does a model respond most strongly when an input resembles what its parameters have learned?
 - a. Because the dot product between input and parameters is largest when the two match
 - b. Because parameters are always longer arrows than inputs
 - c. Because the model copies the input into its parameters
 - d. Because matrices can only be multiplied by matching matrices
3. What does multiplying two matrices actually compute?
 - a. The lengths of all the rows, one after another
 - b. Each number of one table times the number in the same place of the other
 - c. Many dot products at once, every row of the first against every column of the second
 - d. The exponential of every number in the table
4. In the book's view of linear functions as matrices, what does multiplying two matrices correspond to?

- a. Adding the two functions together
- b. Applying one function after the other
- c. Undoing the first function with the second
- d. Applying both functions to every number separately

Answers

1. **b)** The dot product is large when arrows point the same way, zero at a right angle, and negative when they point in opposite directions.
2. **a)** The dot product is largest when the second arrow matches the first, so when the input resonates with the parameters the response peaks: template matching.
3. **c)** Matrix multiplication is nothing more than many dot products done in one operation, every row of one table against every column of the other.
4. **b)** A matrix times a vector applies the function, and a matrix times a matrix applies one function after the other.

Worked examples

A dot product by hand Find the dot product of the two vectors 0.1, 0, -0.3 and -4.0, 0.05, 0.1, the example the book itself works.

Multiply the numbers in the same places: 0.1 times -4.0 gives -0.4.

Next pair: 0 times 0.05 gives 0.

Last pair: -0.3 times 0.1 gives -0.03.

Add the three results: $-0.4 + 0 - 0.03 = -0.43$, a single number; it is negative, so the arrows point more apart than together.

Answer: -0.43

Practice

1. Compute the dot product of the vectors 1, 2 and 3, 4. Then compute it for 1, 0 and 0, 1, and say what the second answer tells you about the two arrows.

Hint: Multiply the numbers in the same places and add the results.

Answer: $1 \times 3 + 2 \times 4 = 11$. For 1, 0 and 0, 1 the result is 0, so the two arrows are at a right angle: unrelated directions.

2. The book warns that libraries quietly stretch arrays of mismatched shapes. In its example, a column of 4 numbers is added to a plain list of 4 numbers. What shape comes out, and why is that a trap?

Hint: Think of what happens if each number of the column is combined with each number of the list.

Answer: The result is a 4 by 4 table instead of a list of 4 numbers, because the library stretches both arrays to fit. Nothing crashes, so the mistake can go unnoticed.

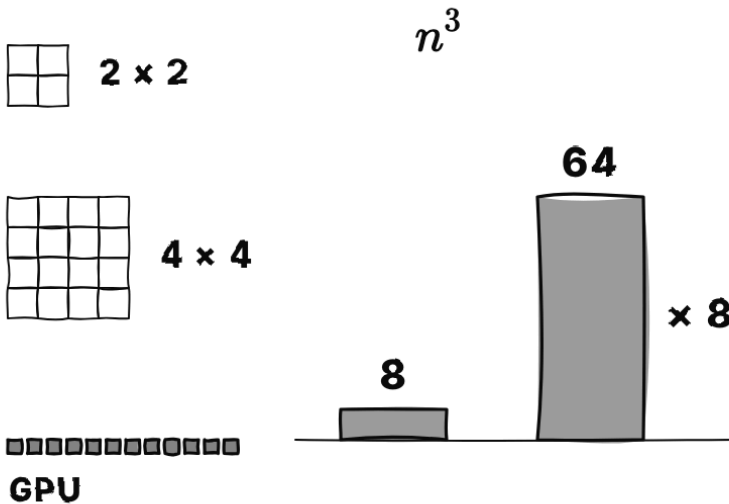
3. Explain in your own words why a photo app could use dot products to find pictures that look like your dog.

Hint: Think of each picture turned into an arrow of numbers.

Answer: If each picture is turned into an arrow of numbers, pictures that look alike give arrows pointing in similar directions. Their dot product is then large, so the app can rank the pictures by how large the dot product with your dog's arrow is.

How Costs Grow, and Which Way Is Uphill

Multiply two small tables of numbers by hand, two rows by two columns each: you need eight multiplications. Now double the size to four by four, and you might expect sixteen. Count them and you find sixty-four, eight times as many, because doubling every side of the job multiplies the work three times over. That surprise is the book's way into a question every AI engineer asks: how does the cost of an operation grow as the data grows?

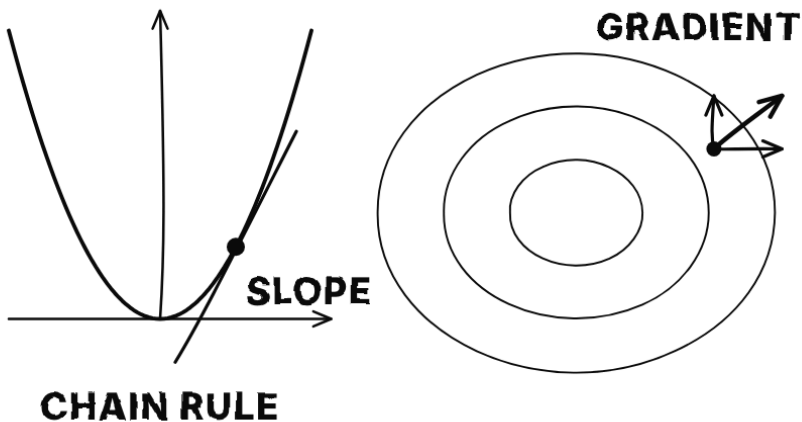


Double the Size, Eight Times the Work

The book separates time, the number of steps, from space, the memory needed, and describes their growth with a shorthand computer scientists use, which keeps only what dominates when sizes get large. By that measure, plain matrix multiplication grows with the cube of the size, which is the eightfold jump you just counted. Cleverer methods with slower growth exist, yet they are rarely used, because they are hard to spread across the thousands of small workers in a graphics chip. Having many workers only divides the time by a fixed amount, and moving data around can cost more than the arithmetic itself, so in practice you measure real code rather than trust the formula alone. This is why training the model behind a chat assistant is planned around memory and chips as much as around mathematics.

Beyond tables, the book needs only two extensions: doing the same matrix product for a whole batch of tables at once, and a dot product over arrays of any shape. It also shows an optional shorthand called `einsum`, where a short string of letters names which axes are combined and which are summed, so the same line works in different libraries.

Then comes the heart of the chapter, the derivative. For a function of one number, the derivative is the slope of the tangent line at a point: it tells you whether the function rises or falls if you step a little to the right. A few rules let you find derivatives of sums, products and, most importantly, of one function applied after another, where the slopes simply multiply; this chain rule will carry much of the book.

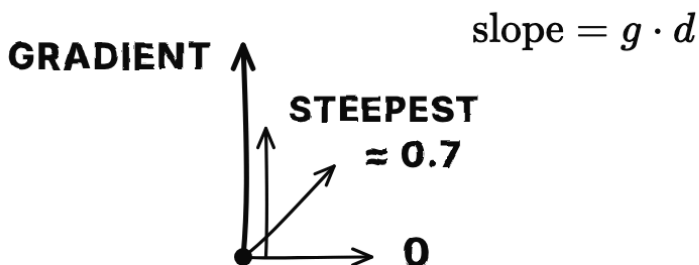


Slope, Then Gradient

A function of many numbers is like a landscape, and there is no single slope, because you can step in endless directions. Stepping along one axis at a time gives one slope per number, called a partial derivative, and stacking them all gives the gradient. The book's key result is that the slope in any other direction is just the dot product of the gradient with that direction, so knowing the gradient means knowing every slope at once. When a photo app is trained, the gradient tells, for each of its millions of parameters, how the error changes if that parameter is nudged.

Try it yourself

Stand on a sloping lawn or a ramp and take one small step in several directions: straight up, sideways, diagonally. You should feel the steepest climb in one direction, no climb at all at right angles to it, and partial climbs in between. The gradient is the arrow of that steepest direction, and every other step's slope follows from how closely it lines up with that arrow.



n^3 CHAIN RULE

Every Slope from One Arrow

Left out here are the formal definition of the growth shorthand, the einsum syntax and the exact rules of differentiation; they are in the

sections «Computational complexity», «Einstein's notation» and «Derivatives of scalar functions».

Remember

The cost of an operation is judged by how it grows with size; plain matrix multiplication grows with the cube of the size, so doubling the tables makes eight times the work. Real speed also depends on the hardware, on parallel work and on moving data, which is why code is measured, not only analysed. A derivative is the slope of a function at a point, and the chain rule multiplies slopes when one function follows another. The gradient collects the slope along every axis, and the slope in any direction is its dot product with the gradient, so the gradient alone says how a function changes in every direction.

Questions

1. Why does doubling the size of two square tables make plain matrix multiplication about eight times slower?
 - a. Because the work grows with the cube of the size, and two cubed is eight
 - b. Because each number has to be copied eight times
 - c. Because the graphics chip slows down for larger tables
 - d. Because the result table becomes eight times as wide

2. Why are the cleverer multiplication methods with slower growth rarely used in practice?
 - a. Because they give wrong answers for large tables
 - b. Because they are hard to spread across the many small workers of a graphics chip
 - c. Because they need tables of a special shape
 - d. Because nobody has written them down yet
3. A function of many numbers has a slope in endless directions. What is enough to know all of them?
 - a. The slope along the first axis only
 - b. The value of the function at many random points
 - c. The gradient: the slope in any direction is its dot product with that direction
 - d. The second derivative of the function
4. On a sloping lawn, in which direction does a small step give no climb at all?
 - a. Straight along the steepest direction
 - b. At a right angle to the steepest direction
 - c. Exactly opposite to the steepest direction
 - d. Diagonally between up and sideways

Answers

1. **a)** Plain matrix multiplication grows with the cube of the size, so doubling every side multiplies the work by two three times over: eight.
2. **b)** Methods with slower growth exist, but they are hard to spread across the thousands of small workers in a graphics chip, so they lose in real use.

- 3. c)** The slope in any direction is the dot product of the gradient with that direction, so knowing the gradient means knowing every slope at once.
- 4. b)** The gradient points along the steepest climb, and a step at a right angle to it lines up with it not at all, so its slope is zero.

Worked examples

Counting the work of a matrix product A table with 3 rows and 4 columns is multiplied by a table with 4 rows and 5 columns. How many single multiplications does the plain method need?

The result has 3 rows and 5 columns, so 15 numbers must be computed.

Each of them is a dot product of a row of length 4 with a column of length 4, which takes 4 multiplications.

So the total is 15 times 4, which is 60, the product of the three sizes 3, 4 and 5.

Answer: 60 multiplications

Practice

1. The book's code example multiplies a batch of 4 tables, each 5 by 2, by a batch of 4 tables, each 2 by 3. What is the shape of the result?

Hint: Each of the 4 pairs is multiplied separately, like ordinary tables.

Answer: A batch of 4 tables, each 5 by 3: in every pair, 5 by 2 times 2 by 3 gives 5 by 3.

2. The book plots the function $x^2 - 1.5x$. Its slope at a point x is $2x - 1.5$. Is the function rising or falling at $x = 1$, and at $x = 0$?

Hint: A positive slope means rising to the right, a negative slope falling.

Answer: At $x = 1$ the slope is $2 - 1.5 = 0.5$, positive, so the function is rising; at $x = 0$ the slope is -1.5 , negative, so it is falling.

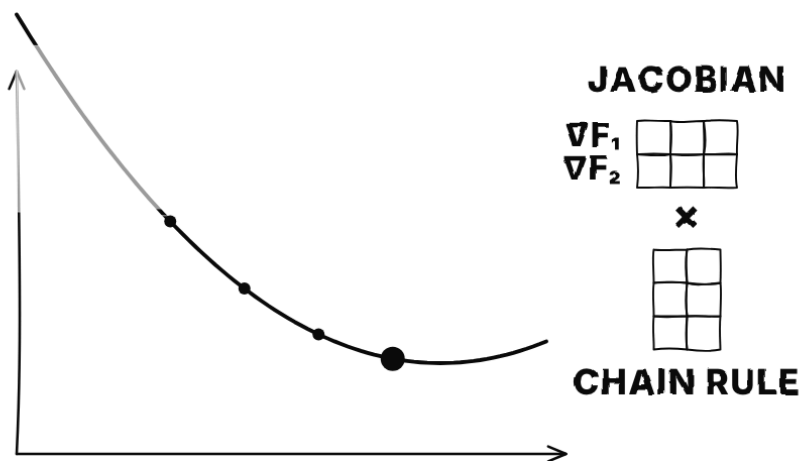
3. At some point a function of two numbers has the gradient $2, -1$. What is the slope if you step in the direction $1, 1$?

Hint: Take the dot product of the gradient with the direction.

Answer: $2 \times 1 + (-1) \times 1 = 1$, so the function rises with slope 1 in that direction.

Walking Downhill in the Fog

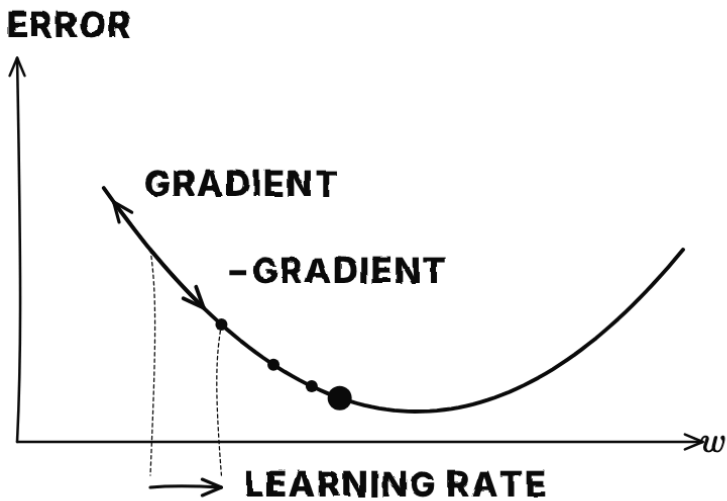
Imagine standing on a hillside in thick fog, unable to see more than your own feet, and wanting to reach the lowest point of the valley. You cannot see the valley, but you can feel which way the ground tilts under your shoes. So you take a small step in the direction where it falls most steeply, feel again, and step again. Surprisingly often, this blind recipe gets you down, and it is exactly how almost every AI model is trained.



Feeling the Slope in the Fog

First the book widens the slope idea once more. When a function gives back several numbers instead of one, each output has its own gradient, and stacking them gives a table called the Jacobian. The chain rule carries over: for one function applied after another, their Jacobians are simply multiplied, as tables are. Like the slope of a single curve, the Jacobian is the best straight-line guess of how the function behaves just around a point. The book adds a warning for later: for the simplest layer of a model, these tables can be huge and full of repeated numbers, so in practice they are never stored in full, a trick the chapter on automatic differentiation explains.

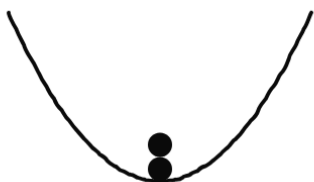
Now the fog. The model's parameters are your position, and a number measuring how badly the model performs is your height; training means finding the position where that number is lowest. Except in rare cases there is no formula that jumps straight to the answer, so you move step by step, each step having a length and a direction. In machine learning the length is called the learning rate. Any direction that goes down counts as a descent direction, and because the slope in a direction is its dot product with the gradient, the steepest way down is straight opposite to the gradient. Stepping again and again against the gradient, with small enough steps, is gradient descent. When a chat assistant is trained, every one of its countless training steps is one such small step downhill for all its parameters at once.



Step Against the Gradient

Where does the walk end? It stops where the ground is flat, a stationary point, but flat ground can be a valley floor, a hilltop, or a saddle like a mountain pass that falls one way and rises the other. In general gradient descent only promises to reach some flat point, and which one depends on where you start. For a bowl-shaped landscape, called convex, every slope leads to the same lowest point, so you reach the global minimum from anywhere. Finding the lowest point of a bumpy landscape for certain would mean starting from every possible place, which is hopeless.

CONVEX



EGG CARTON



Bowl versus Egg Carton

Try it yourself

Drop a marble into a round salad bowl from any side: it always ends at the same bottom point. Now drop it onto an egg carton: it settles in whichever hollow is nearest to where it landed. The bowl is a convex problem; the egg carton is what training real models looks like.

The book ends on the twist: every useful neural network is an egg carton, not a bowl, unlike some older learning methods that kept their problems convex. Yet, started from a sensible place, gradient descent on these networks lands in points that work very well in practice. Left

out here are the formal definitions of the Jacobian, of minima and of convexity, and the proof that the steepest descent is opposite the gradient; they are in the sections «Jacobians», «Gradient descent» and «Convergence of gradient descent».

Remember

For a function with many outputs, the gradients stack into a Jacobian, and the Jacobians of chained functions multiply like tables. Training means finding parameters where a measure of error is lowest, by small repeated steps. The steepest way down is opposite to the gradient, and walking that way with a small learning rate is gradient descent. On a bowl-shaped, convex problem it reaches the single lowest point from anywhere; on the bumpy landscapes of neural networks it only reaches some flat point, which in practice works surprisingly well.

Questions

1. Why is the steepest way down a landscape exactly opposite to the gradient?
 - a. Because the gradient always points to the global minimum
 - b. Because the slope in a direction is its dot product with the gradient, which is most negative when the direction is opposite to it
 - c. Because the learning rate reverses the gradient automatically
 - d. Because the gradient is zero everywhere except at the bottom

2. Gradient descent stops on flat ground. Why does that not guarantee the lowest point?
 - a. Because flat ground can also be a hilltop or a saddle, and which flat point is reached depends on the start
 - b. Because gradient descent never stops on flat ground
 - c. Because the learning rate grows near the bottom
 - d. Because flat ground only exists in convex problems
3. What is special about a convex, bowl-shaped problem?
 - a. It has many hollows, so the start decides the answer
 - b. Gradient descent cannot be used on it
 - c. Every slope leads to the same lowest point, so the global minimum is reached from anywhere
 - d. Its gradient is the same everywhere
4. Neural networks are not convex. What does the book say about training them anyway?
 - a. It never works without a closed-form formula
 - b. Started from a sensible place, gradient descent lands in points that work very well in practice
 - c. It only works if the network is turned into a convex problem first
 - d. It always reaches the global minimum after enough steps

Answers

1. **b)** The slope in a direction is the dot product of that direction with the gradient, and this is lowest when the two point in opposite directions.

2. **a)** Flat points can be valley floors, hilltops or saddles, and in general gradient descent only reaches some flat point, depending on where it starts.
3. **c)** For a bowl-shaped, convex landscape every slope leads to the same lowest point, so gradient descent reaches the global minimum from any start.
4. **b)** Every useful neural network is an egg carton, not a bowl, yet from a sensible start gradient descent lands in points that perform very well.

Worked examples

A straight-line guess near a point The book takes the curve x squared minus $1.5x$, whose slope is $2x$ minus 1.5 , and replaces it near $x = 0.5$ by its tangent line. Compare the curve and the line at $x = 0.51$.

At $x = 0.5$ the curve's value is $0.25 - 0.75 = -0.5$, and its slope is $1 - 1.5 = -0.5$.

The tangent line at 0.51 is the value plus slope times the small step: $-0.5 + (-0.5 \times 0.01) = -0.505$.

The curve itself at 0.51 gives $0.2601 - 0.765 = -0.5049$.

The two differ by only one ten-thousandth: very close to the point, the straight line is an excellent guess.

Answer: Curve -0.5049 , line -0.5050

Practice

1. Run two steps of gradient descent on the function x squared, whose slope is $2x$, starting at $x = 3$ with a learning rate of 0.1 .

Hint: Each step: new x equals old x minus learning rate times slope.

Answer: Step 1: slope 6 , so $x = 3 - 0.6 = 2.4$. Step 2: slope 4.8 , so $x = 2.4 - 0.48 = 1.92$. Each step moves closer to the bottom at 0 .

2. A function takes 5 numbers in and gives 3 numbers out. What shape is its Jacobian, and what does each row hold?

Hint: Each output has its own gradient.

Answer: It is a table with 3 rows and 5 columns: each row is the gradient of one output with respect to all 5 inputs.

3. Using the marble picture, explain why the starting point matters when training a neural network but not when minimising a bowl-shaped function.

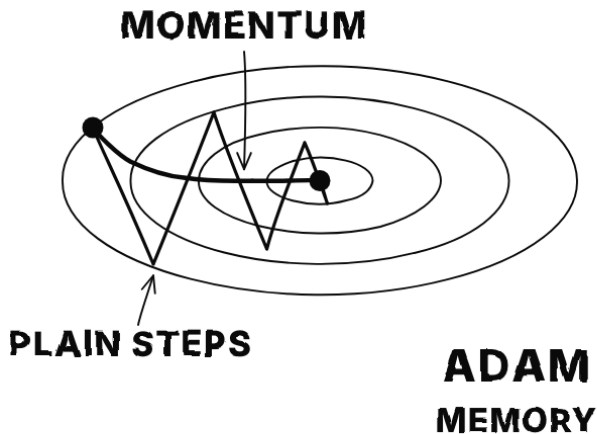
Hint: Compare the salad bowl with the egg carton.

Answer: In a bowl every slope leads to one bottom, so the marble ends there from anywhere. A neural network's error landscape is like an egg carton with many hollows and flat points, so gradient descent settles in whichever one is near its start.

Momentum, and Where the Examples Come From

Roll a ping-pong ball and then a heavy marble down a bumpy, rippled slope, such as a crumpled sheet of cardboard propped up at one end. The light ball jitters, catching on every ripple and changing direction at each bump. The heavy marble keeps much of its earlier motion, rides over the small ripples and reaches the bottom along a smoother path. That difference is the first idea of this stretch of the book, and the second is about the ground itself: the examples a model learns from.

SEEN FROM ABOVE



Jittery Steps versus Momentum

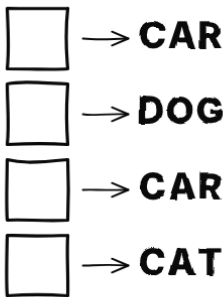
As we saw, gradient descent steps against the gradient, but the gradient only tells the steepest way down right where you stand, and for large models it can be very noisy. Momentum gives the walk some weight: each step keeps part of the previous step's direction, while older steps fade away. The path becomes smoother and training usually gets there faster. Another trick gives each parameter its own step size, depending on how large its gradients have been, and a popular method called Adam combines such ideas; it is the default choice in most frameworks and is fairly forgiving about its settings. The price is memory, because the method must remember earlier steps. The assistants and photo models you use were almost certainly trained with Adam or one of its close relatives.

The book then pauses for practice instead of exercises. It suggests getting fluent in NumPy, the basic library for arrays, and then in JAX and PyTorch, which look much like NumPy but also run on graphics chips and compute gradients for you. The first task is to write gradient descent by hand for a wavy landscape with many hills and hollows, with and without momentum, and watch the paths from different starting points. The book notes, with a smile, that ChatGPT was asked to invent that wavy function, and that nothing else in the book is machine-generated.

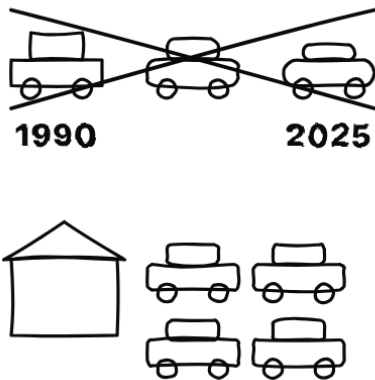
Chapter 3 then asks where learning starts: with a dataset. Some rules are impossible to write by hand, such as recognising objects in photos, but examples of the desired behaviour are easy to collect, so a dataset is simply many pairs of an input and the right output. The book stresses one assumption that quietly decides success: the examples must come from one stable source and be collected without bias. Photos of cars gathered over several decades mix models that no longer exist, and photos taken next to one brand's showroom

overrepresent that brand; a dataset from Italy may serve Rome well and Tokyo badly. Even once a model is running, the world drifts away from its data, and a recommender changes the very behaviour it learns from, so the check never ends.

ONE SOURCE, NO BIAS PAIRS



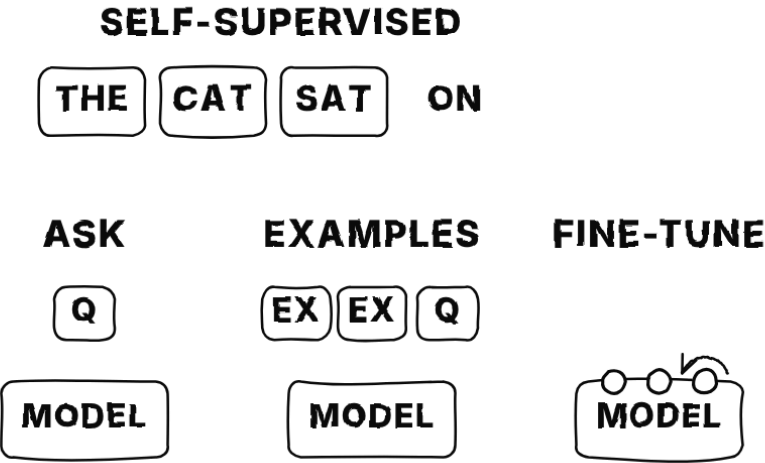
CHECK AGAIN ↻



A Dataset Must Be Fair

Not every task has right answers attached. Without them, in unsupervised learning, one groups similar things or searches for the closest matches, and a model already trained on another task helps, because its inner numbers place similar things close together. Self-supervised learning invents its own answers from raw data, for example by predicting how a piece of text continues, and this is how large language models are first trained. Such a foundation model can then be asked a task directly, shown a few examples in the prompt, or

fine-tuned by gradient descent on a small dataset, sometimes changing only a small share of its parameters. When you paste two sample emails into a chat assistant and ask for a third in the same style, you are using it with a few examples, without changing any of its parameters. Left out here are the momentum formula, the details of the coding exercise and the long list of other learning settings; they are in the sections «Accelerating gradient descent», «From theory to practice» and «Variants of supervised learning».



Without Labels, Then Three Uses

Remember

Momentum lets each step of gradient descent keep part of the previous direction, which smooths the noisy path and speeds training, at the cost of some memory; Adam, the common default, adds a step size for each parameter. A dataset is a collection of input and output pairs, used when the rule itself cannot be written down. It must come from one stable source and be collected without bias, and this has to be checked for as long as the model is in use. Without labelled answers, models can guess, search or learn from tasks they invent themselves, which is how large language models begin, and they can later be used directly, prompted with examples, or fine-tuned.

Questions

1. Why does momentum make the path of gradient descent smoother?
 - a. Because it makes every step longer
 - b. Because each step keeps part of the previous step's direction, so single noisy gradients matter less
 - c. Because it removes the gradient from the update
 - d. Because it only moves when the gradient is zero
2. What is the price of using momentum or Adam?
 - a. More memory, because earlier steps must be remembered
 - b. They need the second derivatives of the function

- c. They can only be used on convex problems
 - d. They need a new dataset at every step
3. A car-recognition dataset is built only from photos taken outside one brand's showroom. Which assumption does this break?
- a. That the dataset is small enough to fit in memory
 - b. That every photo has the same size
 - c. That the examples are collected without bias from the whole source the model will meet
 - d. That the model uses momentum
4. How are large language models first trained, according to the part?
- a. On a self-supervised task invented from raw text, such as predicting how a piece of text continues
 - b. Only on pairs of questions and answers written by hand
 - c. By grouping sentences into clusters without any task
 - d. By fine-tuning a few parameters on a small dataset

Answers

- 1. **b)** Momentum gives the walk some weight: each step keeps part of the previous direction while older steps fade, so the path is smoother.
- 2. **a)** The method must remember earlier steps, so it needs more memory.
- 3. **c)** The examples must come from one stable source and be collected without bias; photos near one showroom overrepresent that brand.
- 4. **a)** Self-supervised learning invents its own answers from raw data, for example by predicting how text continues, and this is how large language

models start.

Worked examples

How fast old steps fade With momentum, each step keeps a share of the previous one; the book's figure uses a share of 0.3. How much weight do the gradients from one, two and three steps back carry in the current step?

The newest gradient enters the step in full: weight 1.

The step before is kept with a share of 0.3: weight 0.3.

The one before that has been kept twice: 0.3 times 0.3, which is 0.09.

Older gradients keep shrinking by 0.3 each time, so the walk remembers only its recent direction.

Answer: 1, 0.3 and 0.09

Practice

1. A model recognising car models was trained on photos collected in Italy. Explain, as the book does, why it may work in Milan but fail in Tokyo, and why the check never ends.

Hint: Ask whether the examples come from the same source the model will meet.

Answer: The dataset reflects the cars common in Italy, so in Tokyo the model meets a different mix of cars it rarely saw. Even in Italy the cars on the road change over the years, so the data drifts away from what the model learned and its performance must keep being checked.

2. You give a chat assistant two sample emails in your prompt and ask for a third in the same style. Is this fine-tuning? Explain the difference.

Hint: Ask whether any parameter of the model changes.

Answer: No, it is few-shot prompting: the examples sit in the prompt and no parameter changes. Fine-tuning would run gradient descent on a small dataset and actually change the model's parameters.

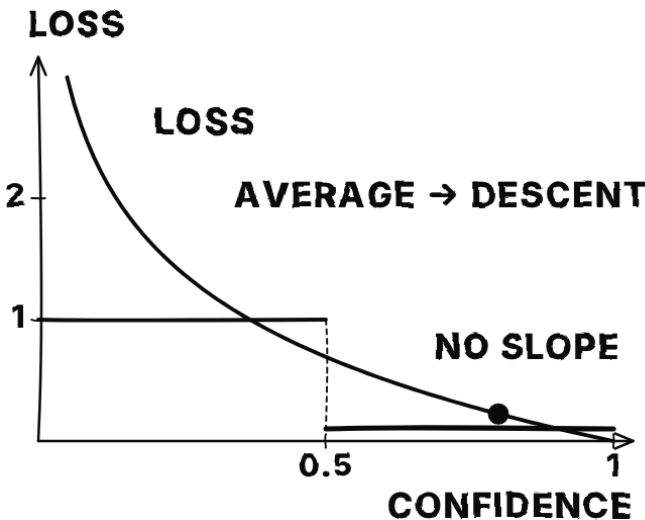
3. Why can a model that was trained to classify images help you search a photo collection for similar pictures, even though nobody labelled them?

Hint: Think of what its inner numbers do with similar images.

Answer: The trained model's inner numbers place similar images close together, so comparing those numbers finds the most similar pictures without any labels.

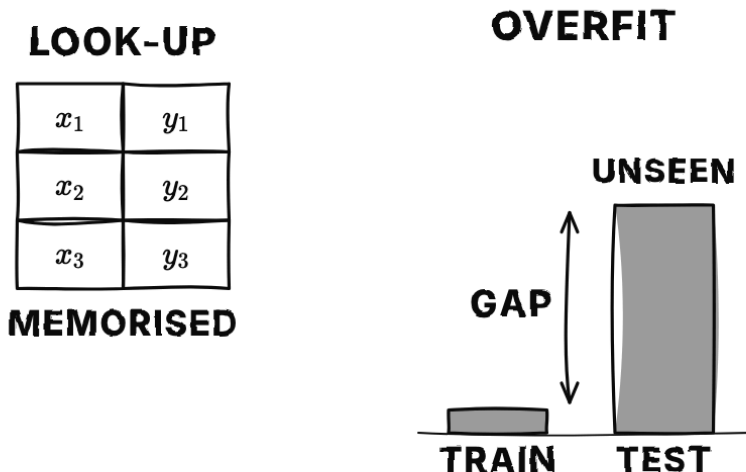
A Score to Lower, and a Test You Have Not Seen

Give a friend last year's driving-theory test with the answers, and let them study it as long as they like. Then hand them the very same test: they score perfectly. Now give them a fresh test with new questions, and their score may collapse, because they learned the answers, not the rules of the road. That small experiment holds the whole of this stretch of the book: how learning is turned into a number to make small, and why making it small on your own examples is not the real goal.



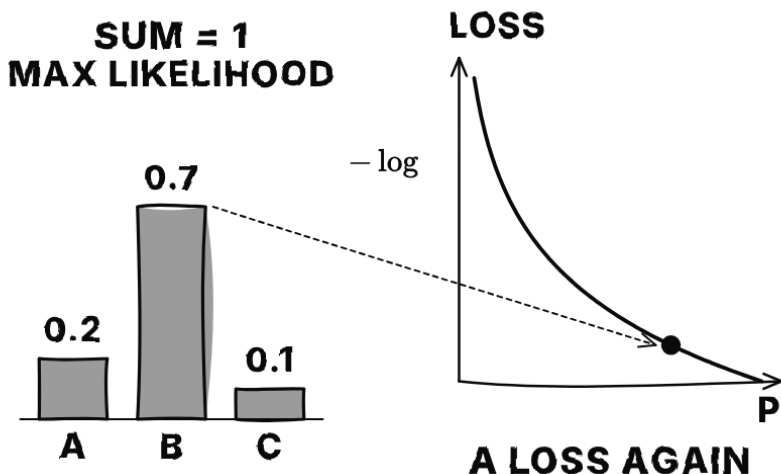
The number is the loss: it compares the model's prediction with the right answer and gives a single score, lower meaning better. It must be differentiable, because the plan is to average the loss over the whole dataset and lower that average by gradient descent on the parameters, a recipe the book calls empirical risk minimization. The loss is where you write down what you care about, and it must also be usable for optimisation. A plain right-or-wrong score fails here, because it stays flat almost everywhere, so its gradient is zero and gradient descent never moves; smoother scores that also reward being confidently right are used instead.

Then comes a trap. The average loss on the training examples can always be made perfect by a cheat: a look-up table that stores every example and answers with it, and gives some default for anything else. That table has memorised and learned nothing, exactly like your friend with last year's test. What we really want is a low loss on future inputs the model has never seen, called the expected risk, which can never be computed because we cannot collect every possible example. The difference between the two is the generalisation gap, and a model with a large gap is said to overfit. In practice one keeps a separate test set, never used in training, and compares the loss there with the training loss. This is why the makers of a photo app judge it on pictures it was never trained on, not on the ones it learned from.



Memorised versus Learned

How do you choose a good loss? The book offers a second, more principled view: let the model predict not one answer but a probability for every possible answer. For a choice among three classes the model gives three probabilities that add up to one, and for a number it can give a most likely value together with how uncertain it is. When a chat assistant writes, it does exactly this for every next word: a probability for each word it might use. Training then follows the principle of maximum likelihood: choose the model under which the dataset you actually observed is as probable as possible.



Probabilities, Then $-\log$

The two views meet neatly. Taking logarithms and flipping the sign turns maximising that probability into minimising a sum of negative log-probabilities, which is just an average loss again. So every loss used in practice can be seen as coming from a choice of probabilities, and the book asks you to keep both views in mind. Left out here are the formulas of the risks, the hinge loss and the Gaussian and categorical models; they are in the sections «3.2 Loss functions», «Expected risk and overfitting» and «Maximum likelihood».

Remember

A loss is a differentiable score of how wrong a prediction is, and training lowers its average over the dataset by gradient descent. Memorising the training examples can make that average perfect while being useless, so the real goal is a low loss on unseen data. A separate test set measures the gap between the two, and a large gap means overfitting. Letting the model predict probabilities and choosing the model that makes the observed data most probable, maximum likelihood, gives the same thing as minimising the negative log-probability, a loss.

Questions

1. Why is a plain right-or-wrong score useless for training by gradient descent?
 - a. Because it is too expensive to compute
 - b. Because it stays flat almost everywhere, so its gradient is zero and the parameters never move
 - c. Because it can only be used on test sets
 - d. Because it gives negative values for correct answers
2. A look-up table that stores every training example gets a perfect average loss on them. Why is it still useless?
 - a. Because it memorises instead of learning, so it says nothing useful about inputs it has not seen
 - b. Because look-up tables cannot store numbers

- c. Because its loss is not differentiable
 - d. Because it makes the dataset larger
3. How can you tell in practice whether a model overfits?
- a. By training it twice as long
 - b. By checking that its training loss is exactly zero
 - c. By comparing its loss on a separate test set, never used in training, with its training loss
 - d. By counting its parameters
4. What does the principle of maximum likelihood choose?
- a. The model with the fewest parameters
 - b. The model under which the dataset actually observed is as probable as possible
 - c. The model whose predictions are always the most likely class
 - d. The model with the largest test set

Answers

- 1. b)** A right-or-wrong score stays flat almost everywhere, so its gradient is zero and gradient descent never moves.
- 2. a)** The table has memorised and learned nothing; the real goal is a low loss on future inputs the model has never seen.
- 3. c)** One keeps a separate test set never used in training and compares the loss there with the training loss; a large gap means overfitting.
- 4. b)** Maximum likelihood chooses the model under which the observed dataset is as probable as possible.

Worked examples

The hinge loss at work For answers that are -1 or $+1$, the book mentions the hinge loss: the larger of 0 and 1 minus the answer times the prediction. Compute it when the right answer is $+1$ and the model predicts 0.3, then 2, and when the right answer is -1 and the model predicts 0.5.

Answer $+1$, prediction 0.3: $1 - 0.3 = 0.7$, larger than 0, so the loss is 0.7; right side, but not confident enough.

Answer $+1$, prediction 2: $1 - 2 = -1$, so the loss is 0; right and confident, nothing to fix.

Answer -1 , prediction 0.5: answer times prediction is -0.5 , so $1 + 0.5 = 1.5$; wrong side, larger loss.

Unlike a right-or-wrong score, this loss changes smoothly with the prediction, so its slope can guide training.

Answer: 0.7, 0 and 1.5

Practice

1. A model gives the right answers of three training examples the probabilities 0.9, 0.8 and 0.5. What is the likelihood of this small dataset, and what is the sum of the negative natural logarithms of the three probabilities?

Hint: The examples are independent, so their probabilities multiply; the logarithm turns the product into a sum.

Answer: The likelihood is $0.9 \times 0.8 \times 0.5 = 0.36$. The negative logs are about 0.105, 0.223 and 0.693, which add up to about 1.02, the same as minus the log of 0.36. Raising the likelihood means lowering this sum.

2. Using the driving-theory test example, explain the difference between the empirical risk and the expected risk.

Hint: Which test is the training set, and which is the real road?

Answer: The empirical risk is the score on last year's test, the examples you trained on; the expected risk is the score over all questions you could ever be asked. Memorising lowers the first without lowering the second, and the second is what matters.

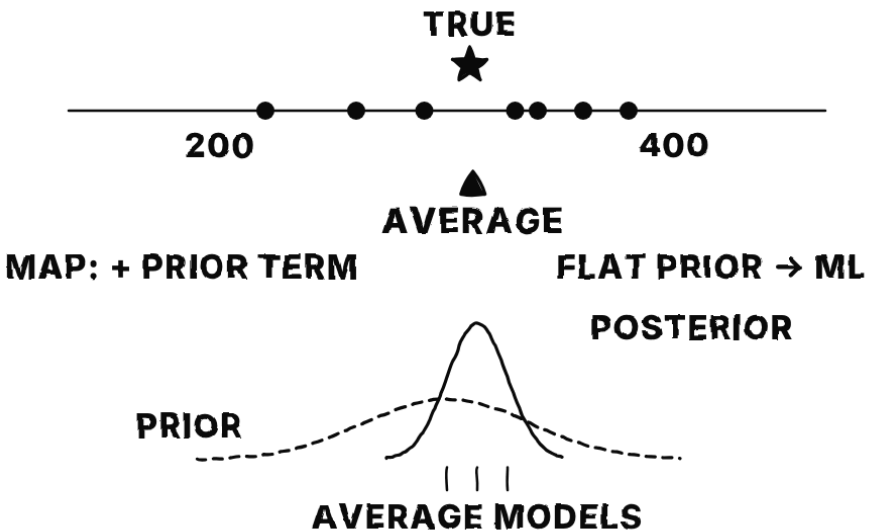
3. Why is it useful that a model can predict a most likely value together with how uncertain it is, instead of a single number?

Hint: Think of what extra information the second output gives.

Answer: The second output tells how confident the prediction is, so the model can express uncertainty, and choosing such a probability model can be easier than inventing a loss by hand; maximum likelihood then supplies the loss.

Many Possible Models, and the Simplest One

Fill a jar with sweets and ask several friends to guess how many there are. Each guess will be off, some too high and some too low. Yet the average of all the guesses is often closer than most single guesses, because their different mistakes partly cancel out. The book uses the same thought to go one step beyond maximum likelihood, and then it builds its very first model.



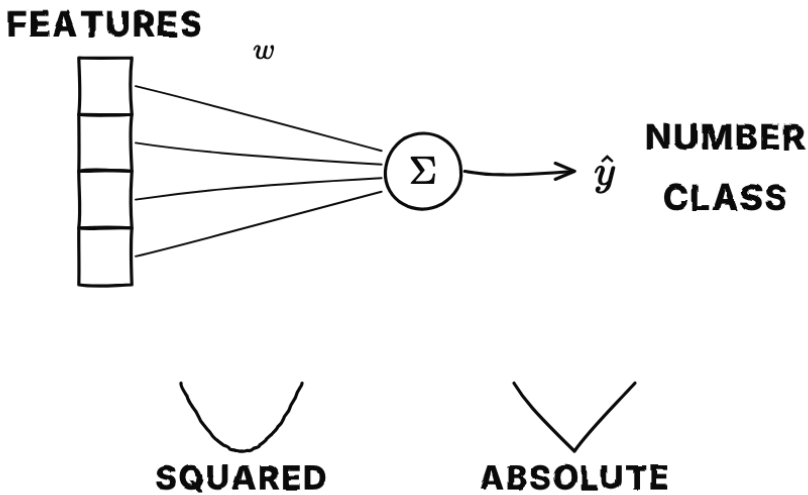
Averaging Many Guesses

Maximum likelihood picks a single best model, but often many different settings of the parameters explain the data about equally well. The Bayesian view keeps them all: before seeing any data you state a prior, a belief about which models are plausible, for example that models with smaller parameters are more stable and so preferred. After seeing the data, Bayes' theorem combines this prior with the likelihood into a posterior, an updated belief over all models. A prediction then averages what all the models say, each weighted by how plausible it now is, just like the friends' guesses. The catch is that the posterior can almost never be computed exactly, so in practice one uses approximations.

Two side results matter a lot. If you keep only the single most plausible model, you get the likelihood plus a term from the prior, and that extra term gently pulls the solution toward what the prior favours; this is called regularisation and it helps against overfitting. With a prior that favours nothing, it reduces exactly to maximum likelihood. And when new data arrive, yesterday's posterior simply becomes today's prior, which helps a model learn more without forgetting what it already knew.

Chapter 4 now builds the simplest block of all, the linear model, of which the book says every differentiable model is a clever variation. The input is a list of features, say the details of a bank's client, and the output is either a number, a regression task, or one of a few classes, a classification task. For predicting a number, the natural loss is the squared error, and the book shows it is not arbitrary: it is exactly what maximum likelihood gives if the errors are assumed to follow a bell-shaped curve. Because squaring lets a few badly wrong examples dominate, there are milder variants, such as the absolute error and a

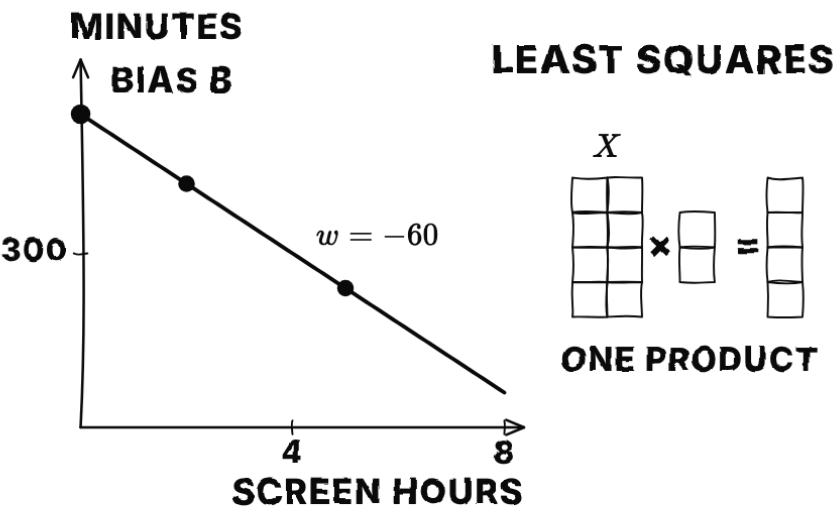
blend of the two, and the single sharp corner of the absolute error is no real obstacle for gradient descent.



The Simplest Block

The linear model itself gives each feature a weight, adds up all the weighted features, and adds a bias, the prediction when every feature is zero. With one feature it draws a straight line, with two a flat plane. A simple estimate of how long your phone's battery will last could be built just like this: so many minutes lost per hour of screen time, so many per hour of navigation, plus a fixed amount. Putting together the linear model, the squared error and the average over the dataset gives the least-squares problem. The book then stacks the whole dataset into one table, one example per row, so that every prediction comes from a single matrix product, which is exactly what graphics

chips do best; shuffling the rows only shuffles the predictions the same way. Left out here are Bayes' formula and its approximations, the formulas of the losses and the vectorised least-squares expression; they are in the sections «3.3 Bayesian learning», «The squared loss and variants» and «The least-squares model».



A Line with a Bias

Remember

The Bayesian view keeps a belief over all models, updates it with the data through Bayes' theorem, and predicts by averaging the models, though in practice this must be approximated. Keeping only the most plausible model adds a regularisation term to maximum likelihood, which pulls toward the prior and fights overfitting. The linear model weights each feature, adds them up with a bias, and is the basic block of all differentiable models. The squared error follows from maximum likelihood with bell-shaped errors; together with the linear model it gives least squares, computed for the whole dataset at once with matrix products.

Questions

1. How does the Bayesian view make a prediction, compared with maximum likelihood?
 - a. It uses only the single model with the lowest loss
 - b. It averages what all plausible models say, each weighted by how plausible it is after seeing the data
 - c. It ignores the data and uses only the prior
 - d. It picks a model at random
2. What does the prior add when you keep only the single most plausible model?

- a. A regularisation term that pulls the solution toward what the prior favours and helps against overfitting
 - b. A second dataset to train on
 - c. A larger learning rate
 - d. Nothing at all, in every case
3. Why is the squared error a natural loss for predicting numbers?
- a. Because it ignores large mistakes
 - b. Because it is the only differentiable loss
 - c. Because it is what maximum likelihood gives when the errors follow a bell-shaped curve
 - d. Because it has a sharp corner at zero
4. Why does the book stack the whole dataset into one table?
- a. So that every prediction comes from a single matrix product, which graphics chips do very fast
 - b. Because the order of the examples changes the predictions
 - c. Because linear models cannot handle single examples
 - d. To remove the bias from the model

Answers

- 1. b)** A prediction averages what all the models say, each weighted by its plausibility after the data, like averaging friends' guesses.
- 2. a)** Keeping only the most plausible model gives the likelihood plus a prior term, which pulls toward the prior: regularisation, useful against overfitting.
- 3. c)** The squared error is exactly what maximum likelihood gives if the errors are assumed to follow a bell-shaped curve.

4. a) With one example per row, all predictions come from a single matrix product, exactly what graphics chips do best.

Worked examples

Three losses, one outlier Compare the squared loss, the absolute loss and the blended Huber loss for a small error of 0.5 and a large error of 3. The Huber loss is half the squared error when the error is at most 1, and the absolute error minus one half otherwise.

Error 0.5: squared 0.25, absolute 0.5, Huber half of 0.25, which is 0.125.

Error 3: squared 9, absolute 3, Huber $3 - 0.5 = 2.5$.

The squared loss grows ninefold from the large error, so one badly labelled example can dominate the average; the absolute and Huber losses grow only in step with the error.

Answer: 0.25, 0.5, 0.125 for the small error; 9, 3, 2.5 for the large one

Practice

1. A linear model has the weights 2 and -1 and the bias 0.5. What does it predict for the features 3 and 4, and what is its squared error if the right answer is 3?

Hint: Weight each feature, add them up, then add the bias.

Answer: $2 \times 3 + (-1) \times 4 + 0.5 = 2.5$. The error is $2.5 - 3 = -0.5$, so the squared error is 0.25.

2. A dataset has 100 examples with 3 features each, stacked into one table. What shape has the table, and how many predictions does one matrix product with the 3 weights give?

Hint: One example per row.

Answer: The table has 100 rows and 3 columns; multiplying it by the 3 weights gives 100 predictions at once, to which the same bias is added.

3. Explain how the Bayesian view helps a model learn from new data without forgetting what it already knew.

Hint: What role does yesterday's posterior play today?

Answer: After the first data, the posterior holds what the model learned. When new data arrive, that posterior is used as the new prior and updated again, so the old knowledge is carried along instead of being overwritten.