From the Chain Rule to the Transformer

The mathematics of automatic differentiation and neural language models

C. Dawes

2026-11-23

Table of contents

Preface **1**
What the book assumes . . . 1
How the book is organised . . . 2
How the claims are checked . . . 3
Conventions . . . 3

I. Calculus in several variables **4**

1. Functions of several variables **5**
1.1. Changing several inputs at once . . . 5
1.2. Vectors and lengths in $\mathbb{R}^n$ . . . 8
1.3. Linear maps and matrices . . . 9
1.4. Limits and continuity . . . 10
1.5. Partial derivatives . . . 12
1.6. The derivative as a linear map . . . 13
1.7. The gradient . . . 17
1.8. Three derivatives used throughout the book . . . 19
1.9. Notes and further reading . . . 20
1.10. Exercises . . . 20

2. The chain rule **22**
2.1. The chain rule . . . 22
2.2. Worked examples . . . 27
2.3. Derivatives as sums over paths . . . 29
2.4. The product and quotient rules . . . 34

2.5. Gradients and level sets 36
2.6. Second derivatives and the Hessian 37
2.7. Taylor's theorem 40
2.8. Critical points and the second-derivative test 43
2.9. Notes and further reading 50
2.10. Exercises 51

II. Automatic differentiation **53**

3. Computational graphs and forward mode **54**
3.1. Evaluation traces 54
3.2. Topological orders 58
3.3. Forward mode 61
3.4. Dual numbers 65
3.5. Forward mode in code 68
3.6. Symbolic differentiation and expression swell 71
3.7. Finite differences 73
3.8. Notes and further reading 74
3.9. Exercises 74

4. Reverse mode and backpropagation **77**
4.1. Adjoints 77
4.2. The reverse sweep 79
4.3. Forward and reverse mode compute the same numbers . . . 82
4.4. The cost of a gradient 84
4.5. Vector–Jacobian products and the shape of the Jacobian . . 86
4.6. Memory and checkpointing 87
4.7. A reverse-mode engine on a tape 89
4.8. Linked nodes and depth-first search 98
4.9. History 103
4.10. Notes and further reading 103
4.11. Exercises 103

III. Learning as optimisation **107**

5. Gradient descent **108**
5.1. The iteration . . . 109
5.2. Smoothness . . . 110
5.3. The descent lemma . . . 114
5.4. Functions that are not convex . . . 115
5.5. Convex functions . . . 117
5.6. Strong convexity . . . 121
5.7. Quadratic functions and the condition number . . . 124
5.8. Gradient flow . . . 127
5.9. Notes and further reading . . . 131
5.10. Exercises . . . 131

6. Stochastic and adaptive methods **134**
6.1. Probability on a finite set . . . 134
6.2. Minibatch gradients . . . 139
6.3. Stochastic gradient descent . . . 141
6.4. Momentum . . . 144
6.5. Exponential moving averages . . . 148
6.6. Adam . . . 151
6.7. Learning-rate schedules . . . 154
6.8. Notes and further reading . . . 156
6.9. Exercises . . . 157

IV. Neural networks **159**

7. Neural networks **160**
7.1. Neurons and layers . . . 160
7.2. Backpropagation . . . 165
7.3. What automatic differentiation computes at the corner . . . 169
7.4. Universal approximation in one variable . . . 172
7.5. Initialisation and the size of activations . . . 176

7.6. Notes and further reading . . . 181
7.7. Exercises . . . 181

8. Probability and loss functions **184**
8.1. Distributions on a finite set . . . 184
8.2. Likelihood . . . 186
8.3. Entropy and divergence . . . 187
8.4. Maximum likelihood for a categorical distribution . . . 191
8.5. The softmax function . . . 192
8.6. Cross-entropy loss for a softmax model . . . 196
8.7. Where squared error comes from . . . 198
8.8. Notes and further reading . . . 199
8.9. Exercises . . . 200

V. Language models **202**

9. Language models **203**
9.1. Tokens and vocabularies . . . 203
9.2. The probability of a sequence . . . 205
9.3. Bigram models . . . 208
9.4. Loss and perplexity . . . 213
9.5. Embeddings and a neural bigram model . . . 215
9.6. Longer contexts . . . 221
9.7. Notes and further reading . . . 222
9.8. Exercises . . . 223

10. Attention **225**
10.1. Attention as a lookup . . . 225
10.2. Why the scores are scaled . . . 227
10.3. Self-attention and the causal mask . . . 231
10.4. Multi-head attention . . . 235
10.5. Order and position . . . 237
10.6. How gradients pass through attention . . . 240

10.7. Cost 242
10.8. Notes and further reading 243
10.9. Exercises 243

11. A small transformer **245**
11.1. RMS normalisation 245
11.2. Residual connections 252
11.3. The complete model 255
11.4. Training 261
11.5. Sampling and temperature 265
11.6. Caching keys and values 267
11.7. What the small model leaves out 269
11.8. Notes and further reading 270
11.9. Exercises 270

References **273**

Appendices **281**

A. Floating-point arithmetic **281**
A.1. Binary floating-point numbers 281
A.2. Cancellation 284
A.3. The error of finite differences 285
A.4. Softmax and log-sum-exp 288
A.5. Notes and further reading 291
A.6. Exercises 291

B. Solutions to the exercises **293**
B.1. Chapter 1 293
B.2. Chapter 2 295
B.3. Chapter 3 298
B.4. Chapter 4 301
B.5. Chapter 5 305

B.6. Chapter 6 308
B.7. Chapter 7 311
B.8. Chapter 8 314
B.9. Chapter 9 316
B.10. Chapter 10 319
B.11. Chapter 11 321
B.12. Appendix A 324

Preface

A modern language model is a function with a great many adjustable numbers in it. Training it means changing those numbers, a little at a time, so that the function's output gets closer to the text it learns from. Each change needs the same piece of information: how the error responds to a small change in every one of the numbers. That information is a gradient, and the method that computes it efficiently is reverse-mode automatic differentiation.

This book develops the mathematics behind that process, from the derivative of a function of several variables to a complete, small transformer language model. It proves what it can and says plainly where it relies on a result proved elsewhere. Code appears where it shows an idea more clearly than prose, but the book is about the mathematics, and no chapter depends on running a program.

What the book assumes

The book assumes a first university course in calculus, vectors and matrices. In particular it assumes:

- differentiation and integration of functions of one variable, including the chain rule, the product rule and the mean value theorem;
- vectors in two and three dimensions, and the dot product;
- matrices, their products, and the eigenvalues and eigenvectors of small matrices;

- linear transformations of the plane;
- first-order differential equations and recurrence relations;
- elementary counting and probability;
- direct proof, proof by contradiction and proof by induction.

Everything beyond that is developed in the text. The first part of the book extends calculus to functions of several variables, which the later parts use constantly.

How the book is organised

The book has five parts.

1. **Calculus in several variables** defines the derivative of a function from $\mathbb{R}^n$ to $\mathbb{R}^m$ as a linear map and proves the chain rule.
2. **Automatic differentiation** turns the chain rule into algorithms. Forward mode computes derivatives alongside values. Reverse mode computes a whole gradient for a small multiple of the cost of evaluating the function.
3. **Learning as optimisation** studies gradient descent and its stochastic and adaptive variants, including Adam.
4. **Neural networks** introduces networks as compositions of simple functions, and loss functions as measures of how well a probability model fits data.
5. **Language models** builds up to attention and assembles a small transformer of the kind used in modern language models.

Each chapter ends with exercises. Solutions are in the appendix.

How the claims are checked

Every computed example and exercise solution in this book is checked by computer algebra (SymPy and Maxima) each time the book is built, and a failed check stops the build. Selected theorems are also proved in the Lean proof assistant. These checks stay out of the text unless they help explain something, but their sources are part of the book's repository.

Conventions

Vectors are columns and are printed in bold, as in $\mathbf{x} = (x_1, \dots, x_n)^\top$. The norm $\|\mathbf{x}\|$ is always the Euclidean length. Theorems, definitions and examples are numbered within each chapter, and the symbol $\square$ marks the end of a proof.

Part I.

Calculus in several variables

1. Functions of several variables

The small language model in the last chapter of this book has 4192 adjustable numbers, its parameters. Training it means making one number that measures the model's error, its loss, as small as possible (Chapter 5 states these terms precisely), and that means answering one question thousands of times: if every parameter changes a little, how much does the loss change? For a function of one variable the derivative answers that question. Near a point a,

$$f(a+h) \approx f(a) + f'(a)\,h,$$

so a small change h in the input changes the output by about $f'(a)\,h$. The change h is called the *increment* of the input, often written Δx, and the resulting change $\Delta f = f(a+h) - f(a)$ is the increment of the function. The approximation says $\Delta f \approx f'(a)\,\Delta x$ when Δx is small. This chapter extends that approximation to functions with many inputs and many outputs. Before the definitions, a worked example shows what the extension looks like and why matrices appear in it.

1.1. Changing several inputs at once

Take $f(x, y) = x^2 y$ at the point $(1, 2)$, where $f(1, 2) = 2$. First change one input at a time, holding the other fixed.

- Change x alone. With $y = 2$ held fixed, f is the function $2x^2$ of one variable, whose derivative at $x = 1$ is 4. So an increment h_1 in x changes f by about $4h_1$: for $h_1 = 0.01$, $f(1.01, 2) = 2.0402$, an increment of 0.0402.

- Change y alone. With $x = 1$ held fixed, f is the function y, whose derivative is 1. So an increment h_2 in y changes f by $1 \cdot h_2$: for $h_2 = 0.01$, $f(1, 2.01) = 2.01$, an increment of 0.01.

These two rates, 4 and 1, are the *partial derivatives* of f at $(1, 2)$, written $\partial f/\partial x$ and $\partial f/\partial y$: each is an ordinary derivative, taken with the other input frozen.

Now change both inputs together, with increments $h_1 = h_2 = 0.01$. The two separate effects predict an increment of about $0.04 + 0.01 = 0.05$ in f, and indeed $f(1.01, 2.01) = 2.050401$, an increment of 0.050401. The separate effects add up. Expanding shows why:

$$f(1 + h_1, 2 + h_2) - f(1, 2) = \underbrace{4h_1 + h_2}_{\text{first order}} + \underbrace{2h_1^2 + 2h_1h_2 + h_1^2h_2}_{\text{products of increments}}.$$

The leftover terms are products of two or more increments. When the increments are small these products are much smaller still: halving both increments halves the first-order part but roughly quarters the leftover. So for small increments

$$f(1 + h_1, 2 + h_2) \approx f(1, 2) + 4h_1 + 1h_2. \tag{1.1}$$

The increment of f on the right is a fixed combination $4h_1 + 1h_2$ of the increments of the inputs. A function of the increments of this form, a sum of fixed multiples of them, is what Section 1.3 calls a *linear map*, and it is the several-variable version of "multiply the increment by $f'(a)$". That linear map is the derivative of f at $(1, 2)$.

A model usually has several outputs, so add a second one, $g(x, y) = x + 3y$, and treat the pair as one function $F(x, y) = (x^2y,\ x + 3y)$ with values in $\mathbb{R}^2$. Each output gets its own approximation, and the two approximations fit into one matrix equation:

$$F(1 + h_1, 2 + h_2) \approx F(1, 2) + \underbrace{\begin{pmatrix} 4 & 1 \\ 1 & 3 \end{pmatrix}}_{J} \begin{pmatrix} h_1 \\ h_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 7 \end{pmatrix} + \begin{pmatrix} 4h_1 + h_2 \\ h_1 + 3h_2 \end{pmatrix}.$$

The matrix is a table of the partial derivatives, laid out so that matrix multiplication does the adding up:

- **row** i lists how output i responds to each input: row 1 is $(\partial f/\partial x,\ \partial f/\partial y) = (4, 1)$, and row 2 is $(\partial g/\partial x,\ \partial g/\partial y) = (1, 3)$;
- **column** j lists how every output responds to input j alone: an increment h_1 in x alone multiplies column 1 by h_1, and changes F by about $(4h_1, h_1)$;
- **entry** (i, j) is the rate at which output i changes per unit increment of input j, with the other inputs frozen.

Multiplying J by the vector of increments takes each column, scales it by its increment, and adds the results. That is exactly the rule "separate effects add up". With the increment $(0.01, 0.01)$ it predicts the change $(0.05, 0.04)$; the true change is $(0.050401, 0.04)$.

This one example contains the whole chapter. Close to a point, the increment of a well-behaved function is approximately a linear map of the increment of its input, and that linear map is the derivative. A linear map between spaces of vectors is always given by a matrix (Theorem 1.2), and for the derivative the entries of that matrix are the partial derivatives (Theorem 1.5). When there is a single output the matrix has a single row, and standing it upright as a column vector gives the *gradient*, which points in the direction in which the output increases fastest (Theorem 1.8). Gradients are what the rest of the book computes and uses.

Two points need care before these statements can be made precise. "Approximately" must mean that the error is small compared with the increment, as it was above, where the leftover terms were products of increments. And partial derivatives alone, which look along one input at a time, can exist even when changing several inputs together does not behave like this at all (Example 1.3). The sections that follow set up the language of vectors and linear maps, and then the definitions that make "approximately linear" exact.

1.2. Vectors and lengths in $\mathbb{R}^n$

Definition 1.1 (The space $\mathbb{R}^n$)**.** For a positive integer n, $\mathbb{R}^n$ is the set of columns $\mathbf{x} = (x_1, \dots, x_n)^\top$ of n real numbers. Vectors are added and multiplied by real numbers component by component. The *standard basis vector* $\mathbf{e}_j$ has 1 in position j and 0 everywhere else.

Every vector is a combination of the standard basis vectors, $\mathbf{x} = x_1\mathbf{e}_1 + \cdots + x_n\mathbf{e}_n$. When n is 2 or 3 these are the vectors of the plane and of space, written as columns.

Definition 1.2 (Dot product and norm)**.** The *dot product* of $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^n$ is

$$\mathbf{x} \cdot \mathbf{y} = x_1y_1 + \cdots + x_ny_n = \mathbf{x}^\top\mathbf{y},$$

and the *norm* of $\mathbf{x}$ is $\|\mathbf{x}\| = \sqrt{\mathbf{x} \cdot \mathbf{x}}$. A vector of norm 1 is a *unit vector*.

The dot product is symmetric and linear in each argument. The norm satisfies $\|\mathbf{x}\| \geq 0$, with equality only when $\mathbf{x} = \mathbf{0}$, and $\|c\mathbf{x}\| = |c|\,\|\mathbf{x}\|$ for every real c. In the plane and in space, $\|\mathbf{x}\|$ is the length of $\mathbf{x}$, and $\mathbf{x} \cdot \mathbf{y}$ equals $\|\mathbf{x}\|\,\|\mathbf{y}\|\cos\theta$ where θ is the angle between the vectors. In higher dimensions there is no picture to measure an angle on. The next theorem supplies what a picture would.

Theorem 1.1 (Cauchy–Schwarz inequality)**.** *For all* $\mathbf{x}$ *and* $\mathbf{y}$ *in* $\mathbb{R}^n$*,*

$$|\mathbf{x} \cdot \mathbf{y}| \leq \|\mathbf{x}\|\,\|\mathbf{y}\|.$$

Equality holds if and only if one of the vectors is a scalar multiple of the other.

Proof. If $\mathbf{y} = \mathbf{0}$, both sides are 0, and $\mathbf{y} = 0\mathbf{x}$. Suppose $\mathbf{y} \neq \mathbf{0}$. For every real t,

$$q(t) = \|\mathbf{x} - t\mathbf{y}\|^2 = \|\mathbf{x}\|^2 - 2t\,\mathbf{x} \cdot \mathbf{y} + t^2\|\mathbf{y}\|^2 \geq 0.$$

This quadratic in t has positive leading coefficient and never takes a negative value, so its discriminant is not positive: $4(\mathbf{x}\cdot\mathbf{y})^2 - 4\|\mathbf{x}\|^2\|\mathbf{y}\|^2 \le 0$. Taking square roots gives the inequality.

If equality holds, the discriminant is 0, so q has a real root t_0. Then $\|\mathbf{x}-t_0\mathbf{y}\| = 0$, and so $\mathbf{x} = t_0\mathbf{y}$. Conversely, if $\mathbf{x} = c\mathbf{y}$ then $|\mathbf{x}\cdot\mathbf{y}| = |c|\,\|\mathbf{y}\|^2 = \|\mathbf{x}\|\,\|\mathbf{y}\|$, and the case $\mathbf{y} = c\mathbf{x}$ is the same with the roles exchanged. □

So the ratio $\mathbf{x}\cdot\mathbf{y}/(\|\mathbf{x}\|\,\|\mathbf{y}\|)$ always lies between -1 and 1, and we can define the angle between two non-zero vectors in any dimension as the θ in $[0,\pi]$ whose cosine is that ratio.

Corollary 1.1 (Triangle inequality)**.** *For all* $\mathbf{x}$ *and* $\mathbf{y}$ *in* $\mathbb{R}^n$, $\|\mathbf{x}+\mathbf{y}\| \le \|\mathbf{x}\| + \|\mathbf{y}\|$.

Proof. By Theorem 1.1,

$$\|\mathbf{x}+\mathbf{y}\|^2 = \|\mathbf{x}\|^2 + 2\,\mathbf{x}\cdot\mathbf{y} + \|\mathbf{y}\|^2 \le \|\mathbf{x}\|^2 + 2\|\mathbf{x}\|\,\|\mathbf{y}\| + \|\mathbf{y}\|^2 = (\|\mathbf{x}\| + \|\mathbf{y}\|)^2,$$

and both sides of the result are non-negative. □

1.3. Linear maps and matrices

Linear transformations of the plane send lines through the origin to lines through the origin, and each one is given by a 2×2 matrix. The same holds in any dimension.

Definition 1.3 (Linear map)**.** A function $L\colon \mathbb{R}^n \to \mathbb{R}^m$ is *linear* if $L(\mathbf{x}+\mathbf{y}) = L(\mathbf{x}) + L(\mathbf{y})$ and $L(c\mathbf{x}) = cL(\mathbf{x})$ for all vectors $\mathbf{x}$, $\mathbf{y}$ and all real numbers c.

Theorem 1.2. *For every linear map* $L\colon \mathbb{R}^n \to \mathbb{R}^m$ *there is exactly one* $m\times n$ *matrix* A *with* $L(\mathbf{x}) = A\mathbf{x}$ *for all* $\mathbf{x}$*. The* j*th column of* A *is* $L(\mathbf{e}_j)$.

Proof. Let A be the matrix whose jth column is $L(\mathbf{e}_j)$. By linearity,

$$L(\mathbf{x}) = L\Big(\sum_j x_j \mathbf{e}_j\Big) = \sum_j x_j L(\mathbf{e}_j) = A\mathbf{x},$$

because $A\mathbf{x}$ is the combination of the columns of A with weights $x_1, \dots, x_n$. If B is another matrix with $B\mathbf{x} = L(\mathbf{x})$ for all $\mathbf{x}$, then column j of B is $B\mathbf{e}_j = L(\mathbf{e}_j)$, which is column j of A. □

From now on we often use the same letter for a linear map and its matrix. Composing linear maps corresponds to multiplying their matrices, exactly as for transformations of the plane: $L(M(\mathbf{x})) = A(B\mathbf{x}) = (AB)\mathbf{x}$.

Later proofs need a bound on how far a linear map can stretch a vector.

Lemma 1.1. *For an $m \times n$ matrix $A = (a_{ij})$, let $\|A\|_F = \big(\sum_{i,j} a_{ij}^2\big)^{1/2}$. Then $\|A\mathbf{x}\| \le \|A\|_F \, \|\mathbf{x}\|$ for every $\mathbf{x}$ in $\mathbb{R}^n$.*

Proof. Let $\mathbf{r}_i$ be row i of A, written as a column. Component i of $A\mathbf{x}$ is $\mathbf{r}_i \cdot \mathbf{x}$, and by Theorem 1.1 $(\mathbf{r}_i \cdot \mathbf{x})^2 \le \|\mathbf{r}_i\|^2 \|\mathbf{x}\|^2$. Summing over i gives $\|A\mathbf{x}\|^2 \le \big(\sum_i \|\mathbf{r}_i\|^2\big)\|\mathbf{x}\|^2 = \|A\|_F^2 \, \|\mathbf{x}\|^2$. □

The number $\|A\|_F$ is the *Frobenius norm* of A. It is rarely the smallest constant that works, but any constant is enough for what follows.

1.4. Limits and continuity

For a function of one variable, $f(x) \to L$ as $x \to a$ means that $|f(x) - L|$ can be made as small as we like by taking x close enough to a. In $\mathbb{R}^n$ a point can approach $\mathbf{a}$ from infinitely many directions, and the definition has to cover all of them at once. It does so by measuring closeness with the norm.

Definition 1.4 (Limit and continuity)**.** Let f take values in $\mathbb{R}^m$ and be defined at every $\mathbf{x}$ with $0 < \|\mathbf{x} - \mathbf{a}\| < r$, for some $r > 0$. Then $f(\mathbf{x}) \to \mathbf{L}$ as $\mathbf{x} \to \mathbf{a}$ if for every $\varepsilon > 0$ there is a $\delta > 0$ such that

$$\|f(\mathbf{x}) - \mathbf{L}\| < \varepsilon \quad \text{whenever} \quad 0 < \|\mathbf{x} - \mathbf{a}\| < \delta.$$

The function f is *continuous* at $\mathbf{a}$ if it is defined at $\mathbf{a}$ and $f(\mathbf{x}) \to f(\mathbf{a})$ as $\mathbf{x} \to \mathbf{a}$.

The rules for limits of one variable carry over with the same proofs: sums, products and quotients (with non-zero denominator) of continuous functions are continuous, and so are compositions. Each coordinate function $\mathbf{x} \mapsto x_j$ is continuous, because $|x_j - a_j| \le \|\mathbf{x} - \mathbf{a}\|$. So every polynomial in $x_1, \dots, x_n$ is continuous, and so is any expression built from polynomials with exp, sin, cos and log, wherever it is defined. A function with values in $\mathbb{R}^m$ is continuous if and only if each of its components is, because each component is at most the norm in absolute value, and the norm is at most the sum of the components' absolute values.

What is new is that a limit must not depend on the direction of approach.

Example 1.1. Let $f(x, y) = xy/(x^2 + y^2)$ for $(x, y) \neq (0, 0)$, and $f(0, 0) = 0$. Along the line $y = mx$, for $x \neq 0$,

$$f(x, mx) = \frac{mx^2}{x^2 + m^2 x^2} = \frac{m}{1 + m^2}.$$

So f is constant along each line through the origin, but the constant depends on the line: it is 0 along the x-axis and $1/2$ along the line $y = x$. Every disc around the origin contains points of both lines, so no number L is within $1/4$ of both values, and f has no limit at the origin. In particular f is not continuous there.

1.5. Partial derivatives

Definition 1.5 (Partial derivative)**.** Let f be a real-valued function defined on a disc around $\mathbf{a}$ in $\mathbb{R}^n$. The *partial derivative* of f with respect to x_j at $\mathbf{a}$ is

$$\frac{\partial f}{\partial x_j}(\mathbf{a}) = \lim_{t \to 0} \frac{f(\mathbf{a} + t\mathbf{e}_j) - f(\mathbf{a})}{t},$$

if this limit exists.

This is the ordinary derivative of the function of one variable $t \mapsto f(\mathbf{a}+t\mathbf{e}_j)$. In practice we hold every variable except x_j fixed and differentiate with respect to x_j, using the familiar rules.

Example 1.2. Let $f(x, y) = x^2 y + \sin(xy)$. Treating y as a constant, $\partial f/\partial x = 2xy + y\cos(xy)$, and treating x as a constant, $\partial f/\partial y = x^2 + x\cos(xy)$. At $(1, 0)$ these are 0 and 2.

A partial derivative looks along one coordinate direction only, and the next example shows how much that misses.

Example 1.3. Return to the function f of Example 1.1. On the x-axis, $f(t, 0) = 0$ for every t, so $\partial f/\partial x(0, 0) = 0$, and in the same way $\partial f/\partial y(0, 0) = 0$. Both partial derivatives exist at the origin, yet f is not even continuous there.

So the existence of partial derivatives is too weak to serve as the definition of differentiability. Whatever "differentiable" means, it should imply continuity, as it does for functions of one variable.

1.6. The derivative as a linear map

A function f of one variable is differentiable at a, with derivative $f'(a)$, when

$$\frac{f(a+h) - f(a) - f'(a)h}{h} \to 0 \quad \text{as } h \to 0.$$

That is, $f(a+h) \approx f(a) + f'(a)h$ with an error that is small compared with h. The map $h \mapsto f'(a)h$ is a linear map from $\mathbb{R}$ to $\mathbb{R}$, and this is the form of the definition that generalises.

A set $U \subseteq \mathbb{R}^n$ is *open* if around each of its points it contains a disc of some positive radius. Functions in the rest of the chapter are defined on open sets, so a point of U can be approached from every direction without leaving U.

Definition 1.6 (Differentiable function)**.** Let $U \subseteq \mathbb{R}^n$ be open and $f\colon U \to \mathbb{R}^m$. The function f is *differentiable* at $\mathbf{a} \in U$ if there is a linear map $L\colon \mathbb{R}^n \to \mathbb{R}^m$ such that

$$\lim_{\mathbf{h}\to\mathbf{0}} \frac{\|f(\mathbf{a}+\mathbf{h}) - f(\mathbf{a}) - L\mathbf{h}\|}{\|\mathbf{h}\|} = 0. \tag{1.2}$$

The map L is the *derivative* of f at $\mathbf{a}$, written $Df(\mathbf{a})$. The function is *differentiable on* U if it is differentiable at every point of U.

If we write $\mathbf{r}(\mathbf{h}) = f(\mathbf{a}+\mathbf{h}) - f(\mathbf{a}) - Df(\mathbf{a})\mathbf{h}$, the definition says

$$f(\mathbf{a}+\mathbf{h}) = f(\mathbf{a}) + Df(\mathbf{a})\mathbf{h} + \mathbf{r}(\mathbf{h}), \qquad \frac{\|\mathbf{r}(\mathbf{h})\|}{\|\mathbf{h}\|} \to 0.$$

Near $\mathbf{a}$, f is a constant plus a linear map, plus a remainder that is small compared with the step $\mathbf{h}$. The definition says "the" derivative, which the next theorem justifies.

Theorem 1.3. *If two linear maps L and M both satisfy Equation 1.2, then $L = M$.*

Proof. Fix a unit vector $\mathbf{u}$ and put $\mathbf{h} = t\mathbf{u}$ with $t > 0$. By the triangle inequality,

$$\|(L-M)(t\mathbf{u})\| \le \|f(\mathbf{a}+t\mathbf{u}) - f(\mathbf{a}) - L(t\mathbf{u})\| + \|f(\mathbf{a}+t\mathbf{u}) - f(\mathbf{a}) - M(t\mathbf{u})\|.$$

Divide by $t = \|t\mathbf{u}\|$. The left side becomes $\|(L-M)\mathbf{u}\|$, which does not depend on t, and as $t \to 0$ the right side tends to 0. So $(L-M)\mathbf{u} = \mathbf{0}$ for every unit vector $\mathbf{u}$. Every non-zero vector is a multiple of a unit vector, so $L - M$ sends every vector to $\mathbf{0}$. □

Theorem 1.4. *If f is differentiable at $\mathbf{a}$, then f is continuous at $\mathbf{a}$.*

Proof. Let A be the matrix of $Df(\mathbf{a})$. Then $f(\mathbf{a}+\mathbf{h}) - f(\mathbf{a}) = A\mathbf{h} + \mathbf{r}(\mathbf{h})$. By Lemma 1.1, $\|A\mathbf{h}\| \le \|A\|_F \|\mathbf{h}\|$, which tends to 0. Also $\|\mathbf{r}(\mathbf{h})\| = \|\mathbf{h}\| \cdot \|\mathbf{r}(\mathbf{h})\|/\|\mathbf{h}\|$, a product of two quantities that tend to 0. So $f(\mathbf{a}+\mathbf{h}) \to f(\mathbf{a})$. □

By this theorem, the function of Example 1.1 is not differentiable at the origin, although both its partial derivatives exist there.

The derivative is a linear map, so by Theorem 1.2 it has a matrix. The next theorem identifies the entries of that matrix as partial derivatives, and that is how derivatives are computed.

Theorem 1.5 (The Jacobian matrix)**.** *Let $f = (f_1, \dots, f_m)$ be differentiable at $\mathbf{a}$. Then every partial derivative $\partial f_i/\partial x_j(\mathbf{a})$ exists, and the matrix of $Df(\mathbf{a})$ is the $m \times n$* Jacobian matrix

$$J_f(\mathbf{a}) = \begin{pmatrix} \dfrac{\partial f_1}{\partial x_1}(\mathbf{a}) & \cdots & \dfrac{\partial f_1}{\partial x_n}(\mathbf{a}) \\ \vdots & & \vdots \\ \dfrac{\partial f_m}{\partial x_1}(\mathbf{a}) & \cdots & \dfrac{\partial f_m}{\partial x_n}(\mathbf{a}) \end{pmatrix}.$$

Proof. Put $\mathbf{h} = t\mathbf{e}_j$ in Equation 1.2 and let $t \to 0$. Then

$$\left\| \frac{f(\mathbf{a}+t\mathbf{e}_j) - f(\mathbf{a})}{t} - Df(\mathbf{a})\mathbf{e}_j \right\| = \frac{\|\mathbf{r}(t\mathbf{e}_j)\|}{\|t\mathbf{e}_j\|} \to 0.$$

So the vector of difference quotients tends to $Df(\mathbf{a})\mathbf{e}_j$, and therefore so does each of its components:

$$\frac{f_i(\mathbf{a}+t\mathbf{e}_j) - f_i(\mathbf{a})}{t} \to \big(Df(\mathbf{a})\mathbf{e}_j\big)_i.$$

The left side is the difference quotient that defines $\partial f_i/\partial x_j(\mathbf{a})$, which therefore exists. The right side is entry (i,j) of the matrix of $Df(\mathbf{a})$, because column j of that matrix is $Df(\mathbf{a})\mathbf{e}_j$. □

Two special cases are worth stating. For a real-valued function ($m = 1$) the Jacobian is the row $(\partial f/\partial x_1 \ \cdots \ \partial f/\partial x_n)$. For $n = m = 1$ it is the 1×1 matrix $(f'(a))$, so the new definition agrees with the old one.

Theorem 1.5 tells us how to compute a derivative that exists, but Example 1.3 shows that partial derivatives can exist without a derivative. The next theorem gives a practical test that covers every function we meet.

Theorem 1.6 (Continuous partial derivatives)**.** *Suppose every partial derivative $\partial f_i/\partial x_j$ exists at every point of an open disc around $\mathbf{a}$, and is continuous at $\mathbf{a}$. Then f is differentiable at $\mathbf{a}$.*

Proof. A vector-valued function satisfies Equation 1.2 if and only if each component f_i satisfies it with row i of L, because for every vector $\mathbf{v}$, each $|v_i| \le \|\mathbf{v}\|$ and $\|\mathbf{v}\| \le |v_1| + \cdots + |v_m|$. So it is enough to treat a real-valued f. Write $g_j = \partial f/\partial x_j$. We show that

$$\frac{\big|f(\mathbf{a}+\mathbf{h}) - f(\mathbf{a}) - \sum_j g_j(\mathbf{a})h_j\big|}{\|\mathbf{h}\|} \to 0.$$

Take $\mathbf{h}$ with $\|\mathbf{h}\|$ less than the radius of the disc, and walk from $\mathbf{a}$ to $\mathbf{a}+\mathbf{h}$ one coordinate at a time, through the points

$$\mathbf{p}_0 = \mathbf{a}, \qquad \mathbf{p}_k = \mathbf{a} + h_1\mathbf{e}_1 + \cdots + h_k\mathbf{e}_k \quad (k = 1, \dots, n),$$

so that $\mathbf{p}_n = \mathbf{a}+\mathbf{h}$. Every point on the segment from $\mathbf{p}_{k-1}$ to $\mathbf{p}_k$ has the form $\mathbf{a} + (h_1, \dots, h_{k-1}, sh_k, 0, \dots, 0)^\top$ with $0 \le s \le 1$, so its distance from $\mathbf{a}$ is at most $\|\mathbf{h}\|$, and the segment lies in the disc. Along the segment only x_k changes. The function $\varphi(s) = f(\mathbf{p}_{k-1} + s\mathbf{e}_k)$, for s between 0 and h_k, has derivative $\varphi'(s) = g_k(\mathbf{p}_{k-1} + s\mathbf{e}_k)$, so by the mean value theorem there is a point $\mathbf{q}_k$ on the segment with

$$f(\mathbf{p}_k) - f(\mathbf{p}_{k-1}) = h_k\, g_k(\mathbf{q}_k).$$

(If $h_k = 0$ both sides are 0, and we take $\mathbf{q}_k = \mathbf{p}_{k-1}$.) Adding these n equations, $f(\mathbf{a}+\mathbf{h}) - f(\mathbf{a}) = \sum_k h_k\, g_k(\mathbf{q}_k)$, and so

$$\Big|f(\mathbf{a}+\mathbf{h}) - f(\mathbf{a}) - \sum_k g_k(\mathbf{a})h_k\Big| = \Big|\sum_k h_k\big(g_k(\mathbf{q}_k) - g_k(\mathbf{a})\big)\Big| \le \|\mathbf{h}\| \sum_k \big|g_k(\mathbf{q}_k) - g_k(\mathbf{a})\big|,$$

using $|h_k| \le \|\mathbf{h}\|$. As $\mathbf{h} \to \mathbf{0}$, each $\mathbf{q}_k \to \mathbf{a}$, because $\|\mathbf{q}_k - \mathbf{a}\| \le \|\mathbf{h}\|$, and each g_k is continuous at $\mathbf{a}$. So the sum tends to 0, which proves the claim. □

A function whose partial derivatives all exist and are continuous on U is *continuously differentiable* on U. Every function built from polynomials, exp, log, sin and cos by arithmetic and composition is continuously differentiable wherever it is defined, because its partial derivatives are built the same way. The one important exception in this book is the function $\max(x, 0)$, which has a corner at 0, and Chapter 7 deals with it.

Example 1.4. Let $f\colon \mathbb{R}^2 \to \mathbb{R}^2$ be $f(x, y) = (x^2 y,\ x + \sin y)$. Its partial derivatives are continuous everywhere, so by Theorem 1.6 and Theorem 1.5 it is differentiable everywhere, with

$$J_f(x, y) = \begin{pmatrix} 2xy & x^2 \\ 1 & \cos y \end{pmatrix}, \qquad J_f(1, 0) = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}.$$

So near $(1,0)$, $f(1+h_1,h_2) \approx (0,1)+(h_2,\ h_1+h_2)$. For $\mathbf{h}=(0.1,0.1)$ the approximation gives $(0.1,1.2)$, and the true value is $f(1.1,0.1) = (0.121, 1.19983\ldots)$. The errors, about 0.021 and 0.00017, are small compared with $\|\mathbf{h}\| \approx 0.141$. Halving $\mathbf{h}$ divides the first error by about 4 and the second by about 8, so both shrink faster than $\mathbf{h}$ itself, as Equation 1.2 requires.

1.7. The gradient

For a real-valued function, the case of a loss, the Jacobian is a single row. It is convenient to stand it upright.

Definition 1.7 (Gradient)**.** If $f\colon U \to \mathbb{R}$ is differentiable at $\mathbf{a}$, its *gradient* at $\mathbf{a}$ is the column vector

$$\nabla f(\mathbf{a}) = \Big(\frac{\partial f}{\partial x_1}(\mathbf{a}), \ldots, \frac{\partial f}{\partial x_n}(\mathbf{a})\Big)^{\top},$$

so that $Df(\mathbf{a})\mathbf{h} = \nabla f(\mathbf{a}) \cdot \mathbf{h}$ for every $\mathbf{h}$.

Definition 1.8 (Directional derivative)**.** For a unit vector $\mathbf{u}$, the *directional derivative* of f at $\mathbf{a}$ in the direction $\mathbf{u}$ is

$$D_{\mathbf{u}} f(\mathbf{a}) = \lim_{t \to 0} \frac{f(\mathbf{a}+t\mathbf{u}) - f(\mathbf{a})}{t},$$

if the limit exists.

The partial derivatives are the directional derivatives in the directions $\mathbf{e}_1, \ldots, \mathbf{e}_n$.

Theorem 1.7. *If f is differentiable at $\mathbf{a}$, then for every unit vector $\mathbf{u}$ the directional derivative exists and*

$$D_{\mathbf{u}} f(\mathbf{a}) = \nabla f(\mathbf{a}) \cdot \mathbf{u}.$$

Proof. Put $\mathbf{h} = t\mathbf{u}$ in Equation 1.2. Since $\|t\mathbf{u}\| = |t|$,

$$\left| \frac{f(\mathbf{a} + t\mathbf{u}) - f(\mathbf{a})}{t} - \nabla f(\mathbf{a}) \cdot \mathbf{u} \right| = \frac{|r(t\mathbf{u})|}{\|t\mathbf{u}\|} \to 0 \quad \text{as } t \to 0.$$

□

The next theorem is the reason gradients matter for learning.

Theorem 1.8 (Steepest ascent)**.** *Let f be differentiable at $\mathbf{a}$ with $\nabla f(\mathbf{a}) \neq \mathbf{0}$. Among all unit vectors $\mathbf{u}$, the directional derivative $D_{\mathbf{u}} f(\mathbf{a})$ is largest for $\mathbf{u} = \nabla f(\mathbf{a})/\|\nabla f(\mathbf{a})\|$, where it equals $\|\nabla f(\mathbf{a})\|$, and smallest for the opposite unit vector, where it equals $-\|\nabla f(\mathbf{a})\|$.*

Proof. By Theorem 1.7 and Theorem 1.1, $|D_{\mathbf{u}} f(\mathbf{a})| = |\nabla f(\mathbf{a}) \cdot \mathbf{u}| \leq \|\nabla f(\mathbf{a})\| \, \|\mathbf{u}\| = \|\nabla f(\mathbf{a})\|$, so every directional derivative lies between $-\|\nabla f(\mathbf{a})\|$ and $\|\nabla f(\mathbf{a})\|$. For $\mathbf{u} = \nabla f(\mathbf{a})/\|\nabla f(\mathbf{a})\|$,

$$\nabla f(\mathbf{a}) \cdot \mathbf{u} = \frac{\nabla f(\mathbf{a}) \cdot \nabla f(\mathbf{a})}{\|\nabla f(\mathbf{a})\|} = \|\nabla f(\mathbf{a})\|,$$

and replacing $\mathbf{u}$ by $-\mathbf{u}$ changes the sign. □

So the gradient points uphill, in the direction in which f increases fastest, and $-\nabla f(\mathbf{a})$ points downhill. Gradient descent, the subject of Chapter 5, changes the parameters of a model by a small step in the direction of minus the gradient of its loss. Chapter 3 and Chapter 4 show how to compute that gradient when n is in the thousands or the billions.

Example 1.5. Let $f(x, y) = x^2 + 3y^2$ and $\mathbf{a} = (1, 1)$. Then $\nabla f(x, y) = (2x, 6y)$, so $\nabla f(1, 1) = (2, 6)$ with norm $2\sqrt{10} \approx 6.32$, and the direction of steepest descent is $-(1, 3)/\sqrt{10}$. The minimum of f is at the origin, which lies in the direction $-(1, 1)/\sqrt{2}$ from $\mathbf{a}$: steepest descent does not head straight for the minimum. The level curves of f are ellipses, and the

gradient is perpendicular to them (Chapter 2 proves this), so it crosses each ellipse at a right angle instead of pointing at the centre. Chapter 5 shows how this makes gradient descent zigzag along long, narrow valleys.

1.8. Three derivatives used throughout the book

Three kinds of function recur in every model in this book: affine maps, functions applied to each component of a vector, and quadratic forms.

Example 1.6 (Affine maps)**.** Let A be an $m \times n$ matrix, $\mathbf{b}$ in $\mathbb{R}^m$ and $f(\mathbf{x}) = A\mathbf{x}+\mathbf{b}$. Then $f(\mathbf{a}+\mathbf{h})-f(\mathbf{a})-A\mathbf{h} = \mathbf{0}$ exactly, so f is differentiable everywhere with $Df(\mathbf{a}) = A$. A linear map is its own derivative.

Example 1.7 (Functions applied componentwise)**.** Let $\sigma\colon \mathbb{R} \to \mathbb{R}$ be differentiable, and apply it to each component of a vector: $F(\mathbf{x}) = (\sigma(x_1), \dots, \sigma(x_n))^\top$. Component i of the remainder is $\sigma(a_i + h_i) - \sigma(a_i) - \sigma'(a_i)h_i$. It is 0 when $h_i = 0$. Otherwise, divided by $|h_i|$, it tends to 0 because σ is differentiable at a_i, and since $|h_i| \le \|\mathbf{h}\|$, it also tends to 0 when divided by $\|\mathbf{h}\|$. So F is differentiable, and its Jacobian is diagonal:

$$J_F(\mathbf{a}) = \begin{pmatrix} \sigma'(a_1) & & \\ & \ddots & \\ & & \sigma'(a_n) \end{pmatrix}.$$

Component i of F depends only on x_i, and the zeros off the diagonal record that.

Example 1.8 (Quadratic forms)**.** Let A be an $n \times n$ matrix and $f(\mathbf{x}) = \mathbf{x}^\top A\mathbf{x} = \sum_{i,j} a_{ij}x_ix_j$. Expanding,

$$f(\mathbf{a}+\mathbf{h}) = \mathbf{a}^\top A\mathbf{a}+\mathbf{a}^\top A\mathbf{h}+\mathbf{h}^\top A\mathbf{a}+\mathbf{h}^\top A\mathbf{h} = f(\mathbf{a})+(A+A^\top)\mathbf{a}\cdot\mathbf{h}+\mathbf{h}^\top A\mathbf{h},$$

because $\mathbf{h}^\top A\mathbf{a} = (A\mathbf{a})\cdot\mathbf{h}$ and $\mathbf{a}^\top A\mathbf{h} = (A^\top\mathbf{a})\cdot\mathbf{h}$. By Theorem 1.1 and Lemma 1.1, the remainder satisfies $|\mathbf{h}^\top A\mathbf{h}| \le \|\mathbf{h}\|\,\|A\mathbf{h}\| \le \|A\|_F\,\|\mathbf{h}\|^2$, so divided by $\|\mathbf{h}\|$ it tends to 0. Hence

$$\nabla f(\mathbf{a}) = (A + A^\top)\mathbf{a},$$

which is $2A\mathbf{a}$ when A is symmetric. For $n = 1$ this is the familiar $\frac{d}{dx}(ax^2) = 2ax$.

1.9. Notes and further reading

Zorich (2015, ch. 8) develops differential calculus in several variables with full proofs and many more examples. Deisenroth et al. (2020, ch. 5) covers the same material from the point of view of machine learning, with an emphasis on computing gradients of vector and matrix expressions.

1.10. Exercises

Exercise 1.1. Find the three partial derivatives of $f(x, y, z) = xe^{yz}$, and $\nabla f(1, 0, 2)$.

Exercise 1.2. Prove the *parallelogram law*: $\|\mathbf{x}+\mathbf{y}\|^2 + \|\mathbf{x}-\mathbf{y}\|^2 = 2\|\mathbf{x}\|^2 + 2\|\mathbf{y}\|^2$ for all $\mathbf{x}$, $\mathbf{y}$ in $\mathbb{R}^n$.

Exercise 1.3. Prove that $\|\mathbf{x} + \mathbf{y}\| = \|\mathbf{x}\| + \|\mathbf{y}\|$ if and only if $\mathbf{x} = \mathbf{0}$ or $\mathbf{y} = c\mathbf{x}$ for some $c \ge 0$.

Exercise 1.4. The polar-coordinate map is $P(r, \theta) = (r\cos\theta,\ r\sin\theta)$. Find its Jacobian matrix and show that the determinant of the Jacobian is r.

Exercise 1.5.

(a) Show that $f(\mathbf{x}) = \|\mathbf{x}\|^2$ is differentiable on $\mathbb{R}^n$ with $\nabla f(\mathbf{x}) = 2\mathbf{x}$.

(b) Show that $g(\mathbf{x}) = \|\mathbf{x}\|$ is differentiable at every $\mathbf{x} \neq \mathbf{0}$, with $\nabla g(\mathbf{x}) = \mathbf{x}/\|\mathbf{x}\|$, and that it is not differentiable at $\mathbf{0}$.

Exercise 1.6. For $f(x, y) = x^2 + 3y^2$, find the directional derivative at $(1, 1)$ in the direction $(3, 4)/5$. Then find every unit vector $\mathbf{u}$ with $D_{\mathbf{u}} f(1, 1) = 0$.

Exercise 1.7. Let $g(x, y) = x^2 y/(x^2 + y^2)$ for $(x, y) \neq (0, 0)$, and $g(0, 0) = 0$.

(a) Show that g is continuous at the origin.

(b) Show that for every unit vector $\mathbf{u} = (u_1, u_2)$, $D_{\mathbf{u}} g(0, 0) = u_1^2 u_2$.

(c) Deduce that g is not differentiable at the origin, although it is continuous there and has a directional derivative in every direction.

Exercise 1.8. Let R be the matrix of rotation of the plane through an angle θ, and $F(\mathbf{x}) = R\mathbf{x} + \mathbf{b}$. Show that $J_F = R$ at every point, and that $\|F(\mathbf{x}) - F(\mathbf{y})\| = \|\mathbf{x} - \mathbf{y}\|$ for all $\mathbf{x}$ and $\mathbf{y}$.

2. The chain rule

A neural network computes its output in stages. An affine map feeds a function applied to each component, which feeds another affine map, and at the end a loss function turns the output into a single number. Every model in this book is a composition of this kind, and the chain rule says how to differentiate one: the derivative of a composition is the product of the derivatives of its stages. Automatic differentiation, the subject of the next two chapters, is a way of organising that product.

The chapter proves the chain rule for maps between spaces of any dimension and recasts it as a sum over paths through the intermediate variables, the form that Chapter 3 and Chapter 4 build on. The product and quotient rules and the fact that the gradient is perpendicular to level sets follow from it. The second half turns to second derivatives, which the Hessian matrix collects. At a point where the gradient is zero, Taylor's theorem and the eigenvalues of the Hessian decide what kind of point it is, and Chapter 5 uses the same eigenvalues to measure how fast gradient descent converges.

2.1. The chain rule

When a function is computed in two stages, an increment of the input first changes the intermediate values, and those changes then change the output. Each stage has a first-order approximation of the kind Chapter 1

describes, and the question is how the two approximations combine. Take the two stages

$$F(x,y) = (x^2y,\ x+3y), \qquad g(u,v) = uv,$$

so that F has two inputs and two outputs, and g multiplies its two inputs. Their *composition* $g \circ F$ applies F first and g to the result: $(g \circ F)(x,y) = g(F(x,y)) = x^2y(x+3y)$. Start at $(1,2)$, where $F(1,2) = (2,7)$ and $(g \circ F)(1,2) = 14$, and follow an increment (h_1, h_2) of the input through the stages.

- **First stage.** The partial derivatives of F at $(1,2)$ give the Jacobian $J_F(1,2) = \begin{pmatrix} 4 & 1 \\ 1 & 3 \end{pmatrix}$, so the intermediate values change by about $(k_1, k_2) = (4h_1 + h_2,\ h_1 + 3h_2)$.
- **Second stage.** The partial derivatives of g are $\partial g/\partial u = v$ and $\partial g/\partial v = u$. The second stage receives the point $(u,v) = (2,7)$, so its Jacobian there is the row $J_g(2,7) = (7\ \ 2)$, and the output changes by about $7k_1 + 2k_2$.

Feeding the first approximation into the second predicts

$$\Delta(g\circ F) \approx 7(4h_1+h_2)+2(h_1+3h_2) = 30h_1+13h_2, \qquad (7\ \ 2)\begin{pmatrix} 4 & 1 \\ 1 & 3 \end{pmatrix} = (30\ \ 13).$$

The predicted Jacobian of the composition is the product of the two Jacobians, with J_g taken at the point $F(1,2)$ that the first stage delivers. Expanding $(g \circ F)(x,y) = x^3y + 3x^2y^2$ and differentiating directly confirms it: the partial derivatives $3x^2y + 6xy^2$ and $x^3 + 6x^2y$ are 30 and 13 at $(1,2)$. With the increment $(0.01, 0)$ the prediction is 0.30, and the true increment is $g(F(1.01, 2)) - 14 = 2.0402 \cdot 7.01 - 14 = 0.301802$.

The entry $30 = 7{\cdot}4{+}2{\cdot}1$ shows what the matrix product does. An increment of x reaches the output by two routes. Through u, it is multiplied by 4 (the rate of change of u per unit increment of x) and then by 7 (the rate of change of g per unit increment of u), contributing 28. Through v, it is

multiplied by 1 and then 2, contributing 2. The two contributions add, and in the same way $13 = 7 \cdot 1 + 2 \cdot 3$ adds the two routes from y. Summing over the middle index of a matrix product is exactly this adding up of routes. Theorem 2.1 below proves that the Jacobian of a composition is always the product of the Jacobians of its stages.

Turning this prediction into a proof needs some care, even for functions of one variable, where the chain rule reads $(g \circ f)'(a) = g'(f(a))\, f'(a)$. The obvious argument writes the difference quotient of $g \circ f$ as

$$\frac{g(f(a+h)) - g(f(a))}{f(a+h) - f(a)} \cdot \frac{f(a+h) - f(a)}{h}$$

and lets $h \to 0$. It fails when $f(a+h) = f(a)$ for values of $h \neq 0$ arbitrarily close to 0, as happens for $f(x) = x^2 \sin(1/x)$ (with $f(0) = 0$) at $a = 0$, because the first fraction then divides by zero. The proof below avoids that division by working with the remainder form of the derivative from Definition 1.6, $f(\mathbf{a} + \mathbf{h}) = f(\mathbf{a}) + Df(\mathbf{a})\mathbf{h} + \mathbf{r}(\mathbf{h})$ with $\|\mathbf{r}(\mathbf{h})\|/\|\mathbf{h}\| \to 0$, which divides only by the increment $\mathbf{h}$ of the input.

We need one fact about functions with several outputs. The proof of Theorem 1.6 used it in passing; here it is as a lemma. It lets us assemble a function with several outputs from real-valued ones, as the product rule in Section 2.4 does with the pair (f, g), and read off its Jacobian one row at a time.

Lemma 2.1 (Differentiating component by component)**.** *Let $U \subseteq \mathbb{R}^n$ be open and $f = (f_1, \dots, f_m) \colon U \to \mathbb{R}^m$. Then f is differentiable at $\mathbf{a} \in U$ if and only if every component f_i is, and then row i of $J_f(\mathbf{a})$ is $J_{f_i}(\mathbf{a}) = \nabla f_i(\mathbf{a})^\top$.*

Proof. Let L be a linear map from $\mathbb{R}^n$ to $\mathbb{R}^m$ with rows $L_1, \dots, L_m$, and let $\mathbf{v}$ be the vector $f(\mathbf{a} + \mathbf{h}) - f(\mathbf{a}) - L\mathbf{h}$, whose component i is $f_i(\mathbf{a} + \mathbf{h}) - f_i(\mathbf{a}) - L_i\mathbf{h}$. Every vector satisfies $|v_i| \leq \|\mathbf{v}\|$ and $\|\mathbf{v}\| \leq |v_1| + \cdots + |v_m|$. So $\|\mathbf{v}\|/\|\mathbf{h}\| \to 0$ if and only if $|v_i|/\|\mathbf{h}\| \to 0$ for every i. That is, L satisfies Equation 1.2 for f if and only if each row L_i satisfies it for f_i. □

Theorem 2.1 (The chain rule)**.** *Let $U \subseteq \mathbb{R}^n$ and $V \subseteq \mathbb{R}^m$ be open, and let $f\colon U \to \mathbb{R}^m$ and $g\colon V \to \mathbb{R}^p$ with $f(U) \subseteq V$. If f is differentiable at $\mathbf{a} \in U$ and g is differentiable at $f(\mathbf{a})$, then $g \circ f$ is differentiable at $\mathbf{a}$ and*

$$D(g \circ f)(\mathbf{a}) = Dg(f(\mathbf{a}))\, Df(\mathbf{a}), \qquad \text{so} \qquad J_{g\circ f}(\mathbf{a}) = J_g(f(\mathbf{a}))\, J_f(\mathbf{a}). \tag{2.1}$$

Proof. Write $\mathbf{b} = f(\mathbf{a})$, $A = J_f(\mathbf{a})$ and $B = J_g(\mathbf{b})$. By Definition 1.6,

$$f(\mathbf{a}+\mathbf{h}) = f(\mathbf{a}) + A\mathbf{h} + \mathbf{r}(\mathbf{h}), \qquad g(\mathbf{b}+\mathbf{k}) = g(\mathbf{b}) + B\mathbf{k} + \mathbf{s}(\mathbf{k}),$$

where $\|\mathbf{r}(\mathbf{h})\|/\|\mathbf{h}\| \to 0$ as $\mathbf{h} \to \mathbf{0}$ and $\|\mathbf{s}(\mathbf{k})\|/\|\mathbf{k}\| \to 0$ as $\mathbf{k} \to \mathbf{0}$. To deal with $\mathbf{k} = \mathbf{0}$, define $\varepsilon(\mathbf{k}) = \|\mathbf{s}(\mathbf{k})\|/\|\mathbf{k}\|$ for $\mathbf{k} \neq \mathbf{0}$ and $\varepsilon(\mathbf{0}) = 0$. Then ε is continuous at $\mathbf{0}$, and $\|\mathbf{s}(\mathbf{k})\| = \varepsilon(\mathbf{k})\,\|\mathbf{k}\|$ for every $\mathbf{k}$, including $\mathbf{k} = \mathbf{0}$, where both sides are 0 because $\mathbf{s}(\mathbf{0}) = \mathbf{0}$.

For $\mathbf{h}$ with $\mathbf{a}+\mathbf{h} \in U$, put $\mathbf{k}(\mathbf{h}) = f(\mathbf{a}+\mathbf{h}) - f(\mathbf{a}) = A\mathbf{h} + \mathbf{r}(\mathbf{h})$. Then $\mathbf{b} + \mathbf{k}(\mathbf{h}) = f(\mathbf{a}+\mathbf{h})$ lies in V, and

$$g(f(\mathbf{a}+\mathbf{h})) = g(\mathbf{b}) + B\mathbf{k}(\mathbf{h}) + \mathbf{s}(\mathbf{k}(\mathbf{h})) = g(f(\mathbf{a})) + BA\mathbf{h} + \underbrace{B\mathbf{r}(\mathbf{h}) + \mathbf{s}(\mathbf{k}(\mathbf{h}))}_{\mathbf{R}(\mathbf{h})}.$$

It remains to show that $\|\mathbf{R}(\mathbf{h})\|/\|\mathbf{h}\| \to 0$. By Lemma 1.1, $\|B\mathbf{r}(\mathbf{h})\| \le \|B\|_F\,\|\mathbf{r}(\mathbf{h})\|$, so the first term divided by $\|\mathbf{h}\|$ tends to 0. For the second, Lemma 1.1 and the triangle inequality give $\|\mathbf{k}(\mathbf{h})\| \le \|A\|_F\,\|\mathbf{h}\| + \|\mathbf{r}(\mathbf{h})\|$, and so

$$\frac{\|\mathbf{s}(\mathbf{k}(\mathbf{h}))\|}{\|\mathbf{h}\|} = \varepsilon(\mathbf{k}(\mathbf{h}))\,\frac{\|\mathbf{k}(\mathbf{h})\|}{\|\mathbf{h}\|} \le \varepsilon(\mathbf{k}(\mathbf{h}))\left(\|A\|_F + \frac{\|\mathbf{r}(\mathbf{h})\|}{\|\mathbf{h}\|}\right).$$

The bracket tends to $\|A\|_F$. By Theorem 1.4, $\mathbf{k}(\mathbf{h}) \to \mathbf{0}$ as $\mathbf{h} \to \mathbf{0}$, and ε is continuous at $\mathbf{0}$ with value 0, so $\varepsilon(\mathbf{k}(\mathbf{h})) \to 0$. The product tends to 0, and the triangle inequality completes the proof that $D(g \circ f)(\mathbf{a}) = BA$. Composing linear maps multiplies their matrices, which gives the statement about Jacobians. □

The function ε is what rescues the argument when $f(\mathbf{a}+\mathbf{h}) = f(\mathbf{a})$: the proof never divides by $\|\mathbf{k}\|$.

In Equation 2.1, $J_g(f(\mathbf{a}))$ is $p \times m$ and $J_f(\mathbf{a})$ is $m \times n$, so their product is $p \times n$: row i belongs to output i of $g \circ f$, and column j to input x_j. The middle index, which matrix multiplication sums over, runs over the m outputs of f, the *intermediate variables* $y_k = f_k(\mathbf{x})$ that g receives as its inputs. Written out entry by entry, Equation 2.1 says

$$\frac{\partial (g \circ f)_i}{\partial x_j}(\mathbf{a}) = \sum_{k=1}^{m} \frac{\partial g_i}{\partial y_k}(f(\mathbf{a}))\, \frac{\partial f_k}{\partial x_j}(\mathbf{a}), \tag{2.2}$$

where the partial derivatives of g are taken with respect to its variables $y_1, \dots, y_m$. Term k of the sum follows one route, $x_j \to y_k \to$ output i: the rate at which y_k changes per unit increment of x_j, times the rate at which output i changes per unit increment of y_k. The sum adds up the m routes, as $30 = 7 \cdot 4 + 2 \cdot 1$ did in the example.

Two special cases come up constantly. When $n = m = p = 1$ every Jacobian is a 1×1 matrix and Equation 2.1 is the familiar $(g \circ f)'(a) = g'(f(a))\, f'(a)$, now with a proof that covers every case. When $n = 1$ and $p = 1$, f is a curve and g is a real-valued function.

A *differentiable curve* is a function $\gamma \colon I \to \mathbb{R}^n$ on an open interval I that is differentiable at every point of I. Its Jacobian at t is the column of derivatives $\gamma'(t) = (\gamma_1'(t), \dots, \gamma_n'(t))^{\top}$, the *velocity* of the curve.

Corollary 2.1 (Differentiating along a curve)**.** *Let $U \subseteq \mathbb{R}^n$ be open, I an open interval, $\gamma \colon I \to U$ a curve that is differentiable at $t \in I$, and $f \colon U \to \mathbb{R}$ differentiable at $\gamma(t)$. Then*

$$(f \circ \gamma)'(t) = \nabla f(\gamma(t)) \cdot \gamma'(t).$$

Proof. By Theorem 2.1 the 1×1 Jacobian of $f \circ \gamma$ is the row $\nabla f(\gamma(t))^{\top}$ times the column $\gamma'(t)$. □

The rate at which f changes as a point moves along the curve is the dot product of the gradient with the velocity: each coordinate moves at rate $\gamma_j'(t)$, and f responds to it at rate $\partial f/\partial x_j$.

Theorem 1.7 is the case $\gamma(t) = \mathbf{a} + t\mathbf{u}$, whose velocity is $\mathbf{u}$.

2.2. Worked examples

The first example composes the three kinds of map from Section 1.8. It is a network with one layer, differentiated with respect to its input.

Example 2.1 (One layer and a squared error)**.** Let W be an $m \times n$ matrix, $\mathbf{b}$ and $\mathbf{t}$ vectors in $\mathbb{R}^m$, and $\sigma\colon \mathbb{R} \to \mathbb{R}$ differentiable. Define

$$L(\mathbf{x}) = \|\sigma(W\mathbf{x} + \mathbf{b}) - \mathbf{t}\|^2,$$

where σ acts on each component. Then L is the composition of four maps:

$$\mathbf{x} \;\mapsto\; \mathbf{z} = W\mathbf{x} + \mathbf{b} \;\mapsto\; \mathbf{y} = \sigma(\mathbf{z}) \;\mapsto\; \mathbf{e} = \mathbf{y} - \mathbf{t} \;\mapsto\; \mathbf{e}^\top\mathbf{e}.$$

Their Jacobians are W (Example 1.6), the diagonal matrix $\operatorname{diag}(\sigma'(\mathbf{z}))$ with entries $\sigma'(z_1), \dots, \sigma'(z_m)$ (Example 1.7), the identity (Example 1.6 again) and the row $2\mathbf{e}^\top$ (Example 1.8 with $A = I$). Applying Theorem 2.1 three times,

$$J_L(\mathbf{x}) = 2\mathbf{e}^\top I \operatorname{diag}(\sigma'(\mathbf{z}))\, W, \qquad \nabla L(\mathbf{x}) = 2\, W^\top \operatorname{diag}(\sigma'(\mathbf{z}))\, (\mathbf{y} - \mathbf{t}). \tag{2.3}$$

The four factors of J_L are $1 \times m$, $m \times m$, $m \times m$ and $m \times n$, so J_L is a row of length n whose entry j is $\partial L/\partial x_j$. The gradient is that row stood upright. Transposing a product reverses the order of its factors, and the diagonal matrix and the identity are their own transposes, which gives the second formula. For a concrete case take $\sigma = \tanh$, whose derivative is $1 - \tanh^2$, and

$$W = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \quad \mathbf{t} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Then $\mathbf{z} = (0, 1)$, $\mathbf{y} = (0, \tanh 1) \approx (0, 0.7616)$, $\sigma'(\mathbf{z}) \approx (1, 0.4200)$ and $\mathbf{y} - \mathbf{t} \approx (-1, -0.2384)$. So $2\,\mathrm{diag}(\sigma'(\mathbf{z}))(\mathbf{y} - \mathbf{t}) \approx (-2, -0.2003)$, and multiplying by $W^\top$ gives $\nabla L(1, 1) \approx (-2.2003,\ 1.7997)$.

The product in Equation 2.3 can be evaluated in two orders. From the right, the first step multiplies two matrices. From the left, every step multiplies a row vector by a matrix, which costs far less when the matrices are large. That observation is the seed of reverse mode (Chapter 4). In a network the quantities being trained are the entries of W and $\mathbf{b}$ rather than $\mathbf{x}$, and Chapter 7 differentiates with respect to them.

Example 2.2 (Polar coordinates)**.** Let $f\colon \mathbb{R}^2 \to \mathbb{R}$ be differentiable, and express it in polar coordinates: $g(r, \theta) = f(r\cos\theta, r\sin\theta) = f(P(r, \theta))$, with P the map of Exercise 1.4. By Theorem 2.1,

$$\begin{pmatrix} \dfrac{\partial g}{\partial r} & \dfrac{\partial g}{\partial \theta} \end{pmatrix} = \begin{pmatrix} \dfrac{\partial f}{\partial x} & \dfrac{\partial f}{\partial y} \end{pmatrix} \begin{pmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{pmatrix},$$

with the partial derivatives of f evaluated at $P(r, \theta)$. So

$$\frac{\partial g}{\partial r} = \frac{\partial f}{\partial x}\cos\theta + \frac{\partial f}{\partial y}\sin\theta, \qquad \frac{\partial g}{\partial \theta} = -\frac{\partial f}{\partial x}\, r\sin\theta + \frac{\partial f}{\partial y}\, r\cos\theta.$$

For $f(x, y) = x^2 + 3y^2$, the function of Example 1.5, these give $\partial g/\partial r = 2r\cos^2\theta + 6r\sin^2\theta$ and $\partial g/\partial\theta = 4r^2\sin\theta\cos\theta$. Substituting first, $g(r, \theta) = r^2(1 + 2\sin^2\theta)$, and differentiating directly gives the same answers.

The Jacobian of P factorises as $R_\theta D_r$, where

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

is the matrix of rotation through θ and $D_r = \mathrm{diag}(1, r)$ is the diagonal matrix with entries 1 and r. For $r > 0$, multiplying $J_g = \nabla f^\top R_\theta D_r$ on the right by $D_r^{-1} = \mathrm{diag}(1, 1/r)$ gives

$$\begin{pmatrix} \dfrac{\partial g}{\partial r}, & \dfrac{1}{r}\dfrac{\partial g}{\partial \theta} \end{pmatrix} = \nabla f^\top R_\theta,$$

whose transpose is $R_\theta^\top \nabla f$. The matrix $R_\theta^\top$ is the rotation through $-\theta$, which preserves lengths (Exercise 1.8), so

$$\|\nabla f\|^2 = \Big(\frac{\partial g}{\partial r}\Big)^2 + \frac{1}{r^2}\Big(\frac{\partial g}{\partial \theta}\Big)^2.$$

2.3. Derivatives as sums over paths

Entry (i, j) of a product of two matrices is a sum of products, $(BA)_{ij} = \sum_k b_{ik}a_{kj}$, and Equation 2.2 reads this sum as a list of routes from x_j to output i, one through each intermediate variable. Theorem 2.1 describes a computation in stages, where each stage sees only the output of the stage before. A program that evaluates a function is less tidy: a value computed early can be used by any later step, and by several of them. Computing x^3 as $x \cdot x^2$ is the smallest case:

$$v_1 = x, \qquad v_2 = v_1^2, \qquad v_3 = v_1 v_2.$$

The input v_1 reaches v_3 twice over, once directly and once through v_2. Each step is a formula in earlier variables. Write d_{ki} for the partial derivative of the formula for v_k with respect to its argument v_i, with its other arguments held fixed. At $x = 2$, where $v_2 = 4$,

$$d_{21} = 2v_1 = 4, \qquad d_{31} = v_2 = 4, \qquad d_{32} = v_1 = 2.$$

These *local partial derivatives* are not the derivative of v_3 with respect to x: d_{31} holds v_2 fixed, but v_2 changes whenever x does. Follow an increment Δx instead, writing $i \to k$ when v_i is an argument of the formula for v_k. Along the direct route $1 \to 3$ it changes v_3 by about $d_{31}\Delta x = 4\Delta x$. Along the route $1 \to 2 \to 3$ it first changes v_2 by about $d_{21}\Delta x = 4\Delta x$, which then changes v_3 by about $d_{32} \cdot 4\Delta x = 8\Delta x$. The two routes add to $12\,\Delta x$, and $12 = 3 \cdot 2^2$ is the derivative of x^3 at 2. With $\Delta x = 0.01$ the routes predict $0.04 + 0.08 = 0.12$, and the true increment is $2.01^3 - 8 = 0.120601$.

With more variables the routes become longer, and they branch and rejoin wherever a variable feeds several later ones. The definition below names the parts of such a computation, and Theorem 2.2 says that the derivative of any variable with respect to any input is the sum, over all routes, of the products of the local partial derivatives along them. A program that evaluates a function produces exactly this kind of structure, so the theorem is the basis for the computational graphs of Chapter 3.

Definition 2.1 (A function computed through intermediate variables)**.** Fix integers $1 \le n < N$. A *system of intermediate variables* with inputs $v_1, \dots, v_n$ consists of, for each $k = n+1, \dots, N$, a non-empty set $P(k) \subseteq \{1, \dots, k-1\}$ of *parents* of k and a real-valued function φ_k of the variables v_i with $i \in P(k)$ (in increasing order of i), defined on an open subset of $\mathbb{R}^{|P(k)|}$. Given values of the inputs, the system computes

$$v_k = \varphi_k\big((v_i)_{i \in P(k)}\big), \qquad k = n+1, \dots, N,$$

in this order. Draw an arrow $i \to k$ whenever $i \in P(k)$. A *path* from j to k is a sequence $j = i_0 \to i_1 \to \dots \to i_r = k$ of arrows, with $r \ge 0$; the path with $r = 0$ is the *empty path* from k to itself.

The computation of x^3 above is such a system with $n = 1$ input and $N = 3$ variables: $P(2) = \{1\}$ with $\varphi_2(v_1) = v_1^2$, and $P(3) = \{1, 2\}$ with $\varphi_3(v_1, v_2) = v_1 v_2$. Its arrows are $1 \to 2$, $1 \to 3$ and $2 \to 3$, and its two paths from 1 to 3 are $1 \to 3$ and $1 \to 2 \to 3$.

The condition $P(k) \subseteq \{1, \dots, k-1\}$ says that each variable is computed from variables computed before it. A program that runs its steps in order meets it automatically, and it rules out a variable that depends, directly or through others, on itself. Every arrow goes from a smaller index to a larger one, so a path never returns to a variable it has visited, and there are finitely many paths. Inputs have no parents, so no arrow ends at an input.

Suppose the inputs range over an open set $U \subseteq \mathbb{R}^n$ on which every step is defined: for every $\mathbf{x} \in U$, each φ_k receives arguments inside its domain.

Then each v_k is a function of the inputs, and we write it as $F_k \colon U \to \mathbb{R}$. For the inputs themselves, $F_k(\mathbf{x}) = x_k$.

Theorem 2.2 (The chain rule as a sum over paths)**.** *Take a system of intermediate variables as in Definition 2.1, with inputs ranging over an open set U, and a point $\mathbf{a} \in U$. Suppose that each φ_k is differentiable at the values its parents take when the inputs equal $\mathbf{a}$. For an arrow $i \to k$, write*

$$d_{ki} = \frac{\partial \varphi_k}{\partial v_i}$$

for the local partial derivative*, evaluated at those values. Then every F_k is differentiable at $\mathbf{a}$, and for every input j and every k,*

$$\frac{\partial F_k}{\partial x_j}(\mathbf{a}) = \sum_{\text{paths } j=i_0 \to i_1 \to \dots \to i_r = k} \prod_{s=1}^{r} d_{i_s i_{s-1}}, \tag{2.4}$$

where the empty path contributes the empty product 1, and a sum with no paths is 0.

Proof. We prove the statement for $k = 1, 2, \dots, N$ in turn, assuming it for all smaller indices (strong induction).

For an input $k \le n$, $F_k(\mathbf{x}) = x_k$ is linear, so it is differentiable with $\partial F_k / \partial x_j = 1$ if $j = k$ and 0 otherwise. No arrow ends at an input, so the only path from j to k is the empty path when $j = k$, and there is none when $j \neq k$. The right side of Equation 2.4 is 1 or 0 accordingly.

Now let $k > n$, and suppose the theorem holds for every index below k. Write $G_k(\mathbf{x}) = (F_i(\mathbf{x}))_{i \in P(k)}$, so that $F_k = \varphi_k \circ G_k$. Each component F_i of G_k has $i < k$, so it is differentiable at $\mathbf{a}$ by the induction hypothesis, and by Lemma 2.1 so is G_k, with the row for i of its Jacobian holding the partial derivatives of F_i. By hypothesis φ_k is differentiable at $G_k(\mathbf{a})$, so Theorem 2.1 shows that F_k is differentiable at $\mathbf{a}$, and Equation 2.2 gives

$$\frac{\partial F_k}{\partial x_j}(\mathbf{a}) = \sum_{i \in P(k)} d_{ki} \, \frac{\partial F_i}{\partial x_j}(\mathbf{a}). \tag{2.5}$$

By the induction hypothesis, each $\partial F_i/\partial x_j(\mathbf{a})$ is the sum over paths from j to i of products of local partial derivatives, so the right side of Equation 2.5 is the sum, over pairs consisting of a parent i of k and a path from j to i, of that path's product times d_{ki}. Appending the arrow $i \to k$ turns such a pair into a path from j to k, with product multiplied by d_{ki}. Every path from j to k arises exactly once in this way: it is not empty, since $j \le n < k$, and removing its last arrow, which ends at k and so starts at a parent of k, recovers the pair. So the right side of Equation 2.5 equals the right side of Equation 2.4. □

In words: to differentiate v_k with respect to the input x_j, list every path from j to k, multiply the local partial derivatives along each path, and add the products. For $x^3 = x \cdot x^2$ at $x = 2$ the two paths give $d_{31} = 4$ and $d_{32}d_{21} = 8$, and the derivative is their sum 12. The empty path covers the case $k = j$, where the derivative of x_j with respect to itself is 1.

The theorem places no condition on the functions φ_k beyond differentiability. They can be as simple as an addition or a sine, whose local partial derivatives are easy to write down, and the theorem assembles the whole derivative from them.

Example 2.3. Let $f(x, y) = xy\,e^{xy}$, computed through intermediate variables as

$$v_1 = x, \quad v_2 = y, \quad v_3 = v_1 v_2, \quad v_4 = e^{v_3}, \quad v_5 = v_3 v_4.$$

The arrows are $1 \to 3$, $2 \to 3$, $3 \to 4$, $3 \to 5$ and $4 \to 5$, and the local partial derivatives are

$$d_{31} = v_2, \quad d_{32} = v_1, \quad d_{43} = e^{v_3} = v_4, \quad d_{53} = v_4, \quad d_{54} = v_3.$$

The variable v_3 reaches v_5 in two ways, directly and through v_4, so there are two paths from 1 to 5: $1 \to 3 \to 5$, with product $v_2 v_4$, and $1 \to 3 \to 4 \to 5$, with product $v_2 v_4 v_3$. By Theorem 2.2,

$$\frac{\partial f}{\partial x} = v_2 v_4 + v_2 v_4 v_3 = y\,e^{xy}(1 + xy),$$

which agrees with the product rule applied to $xy\,e^{xy}$. In the same way $\partial f/\partial y = x\,e^{xy}(1+xy)$, and at $(1,2)$ the gradient is $(6e^2, 3e^2)$.

When the system is arranged in layers, with every variable of one layer a function of all the variables of the layer before, a path picks one variable from each layer, and Equation 2.4 becomes the expanded product of the Jacobians. The opening example of Section 2.1 has two maps, F and then g, with the layer (u, v) between them, and its two routes from x are the two choices of a variable in that layer.

Corollary 2.2 (Paths through layers)**.** *Let $D \geq 1$, let $U_0, \dots, U_{D-1}$ be open sets with $U_l \subseteq \mathbb{R}^{n_l}$, and let $f_l \colon U_{l-1} \to \mathbb{R}^{n_l}$ for $l = 1, \dots, D$, with $f_l(U_{l-1}) \subseteq U_l$ for $l < D$. Given $\mathbf{x}^{(0)} \in U_0$, put $\mathbf{x}^{(l)} = f_l(\mathbf{x}^{(l-1)})$ for $l = 1, \dots, D$, and suppose that each f_l is differentiable at $\mathbf{x}^{(l-1)}$. Then the composition $f_D \circ \cdots \circ f_1$ is differentiable at $\mathbf{x}^{(0)}$ with Jacobian $J = J_{f_D}(\mathbf{x}^{(D-1)}) \cdots J_{f_1}(\mathbf{x}^{(0)})$, and*

$$J_{ij} = \sum_{k_1=1}^{n_1} \cdots \sum_{k_{D-1}=1}^{n_{D-1}} \frac{\partial x_i^{(D)}}{\partial x_{k_{D-1}}^{(D-1)}} \cdots \frac{\partial x_{k_2}^{(2)}}{\partial x_{k_1}^{(1)}} \frac{\partial x_{k_1}^{(1)}}{\partial x_j^{(0)}},$$

where $\partial x_a^{(l)}/\partial x_b^{(l-1)}$ stands for entry (a, b) of $J_{f_l}(\mathbf{x}^{(l-1)})$.

Proof. The first statement follows from Theorem 2.1 by induction on D. The formula for J_{ij} is Equation 2.4 for the system whose variables are the components of $\mathbf{x}^{(0)}, \dots, \mathbf{x}^{(D)}$, in that order, with the parents of each component of $\mathbf{x}^{(l)}$ being all the components of $\mathbf{x}^{(l-1)}$. The inputs range over U_0, and every step is defined there because $f_l(U_{l-1}) \subseteq U_l$. Each component of f_l is differentiable by Lemma 2.1, and its local partial derivatives are the entries of $J_{f_l}(\mathbf{x}^{(l-1)})$. A path from $x_j^{(0)}$ to $x_i^{(D)}$ visits one component $x_{k_l}^{(l)}$ of each layer in turn, so the paths correspond to the choices of $k_1, \dots, k_{D-1}$. □

Each term of the sum chooses one variable in every intermediate layer and multiplies the rates of change along the route through them. With $D = 2$ the formula is Equation 2.2.

The number of paths can be enormous. With D maps and intermediate layers of width w there are w^{D-1} paths from each input to each output, and Exercise 2.4 builds a system with only $3k+1$ variables and 2^k paths. Computing the derivative takes far less work than listing the paths: the product of Jacobians can be multiplied out one matrix at a time, and the distributive law collects the paths into shared partial sums. Chapter 3 accumulates these partial sums from the inputs forward, and Chapter 4 from the output backward.

2.4. The product and quotient rules

The product rule is the chain rule applied to multiplication. The opening example of Section 2.1 already used it: there $x^2y(x+3y)$ is the product of the two components of F, and its derivative came from composing F with $g(u,v)=uv$, whose Jacobian at (u,v) is the row $(v \;\; u)$. The same argument works for any two differentiable functions, once multiplication is known to be differentiable.

Lemma 2.2. *The map $\mu(u,v)=uv$ from $\mathbb{R}^2$ to $\mathbb{R}$ is differentiable everywhere, with $J_\mu(u,v)=(v \;\; u)$.*

Proof. $\mu(u+h,v+k)-uv-(vh+uk)=hk$, and $|hk|\le\frac{1}{2}(h^2+k^2)=\frac{1}{2}\|(h,k)\|^2$ because $(|h|-|k|)^2\ge 0$. Divided by $\|(h,k)\|$ this tends to 0. □

Theorem 2.3 (Product rule)**.** *Let $U\subseteq\mathbb{R}^n$ be open and $f,g\colon U\to\mathbb{R}$ differentiable at $\mathbf{a}$. Then fg is differentiable at $\mathbf{a}$ and*

$$\nabla(fg)(\mathbf{a})=g(\mathbf{a})\nabla f(\mathbf{a})+f(\mathbf{a})\nabla g(\mathbf{a}).$$

Proof. Let $F = (f, g)\colon U \to \mathbb{R}^2$. By Lemma 2.1, F is differentiable at $\mathbf{a}$ with rows $\nabla f(\mathbf{a})^\top$ and $\nabla g(\mathbf{a})^\top$. Then $fg = \mu \circ F$, and by Theorem 2.1 and Lemma 2.2,

$$J_{fg}(\mathbf{a}) = \begin{pmatrix} g(\mathbf{a}) & f(\mathbf{a}) \end{pmatrix} \begin{pmatrix} \nabla f(\mathbf{a})^\top \\ \nabla g(\mathbf{a})^\top \end{pmatrix} = g(\mathbf{a})\nabla f(\mathbf{a})^\top + f(\mathbf{a})\nabla g(\mathbf{a})^\top.$$

□

Corollary 2.3 (Quotient rule)**.** *Let $U \subseteq \mathbb{R}^n$ be open, and $f, g\colon U \to \mathbb{R}$ differentiable at $\mathbf{a}$, with $g(\mathbf{x}) \neq 0$ for every $\mathbf{x} \in U$. Then f/g is differentiable at $\mathbf{a}$ and*

$$\nabla\Big(\frac{f}{g}\Big)(\mathbf{a}) = \frac{g(\mathbf{a})\nabla f(\mathbf{a}) - f(\mathbf{a})\nabla g(\mathbf{a})}{g(\mathbf{a})^2}.$$

Proof. The function $\rho(v) = 1/v$ on the open set of non-zero reals has derivative $-1/v^2$. By Theorem 2.1, $1/g = \rho \circ g$ is differentiable at $\mathbf{a}$ with gradient $-\nabla g(\mathbf{a})/g(\mathbf{a})^2$. Apply Theorem 2.3 to $f \cdot (1/g)$:

$$\nabla\Big(\frac{f}{g}\Big)(\mathbf{a}) = \frac{1}{g(\mathbf{a})}\nabla f(\mathbf{a}) - f(\mathbf{a})\frac{\nabla g(\mathbf{a})}{g(\mathbf{a})^2},$$

which is the stated formula over a common denominator. □

With $n = 1$ these are the product and quotient rules of one-variable calculus. Exercise 2.3 asks for the corresponding rule for the dot product of two vector-valued functions.

Example 2.4. Let $f(x, y) = x/(1 + x^2 + y^2)$. The denominator $1 + x^2 + y^2$ is never 0, so Corollary 2.3 gives

$$\nabla f(x, y) = \frac{(1 + x^2 + y^2)(1, 0) - x(2x, 2y)}{(1 + x^2 + y^2)^2} = \frac{(1 - x^2 + y^2,\ -2xy)}{(1 + x^2 + y^2)^2},$$

and $\nabla f(1, 1) = (1/9, -2/9)$.

2.5. Gradients and level sets

A contour map shows the height of the ground by drawing curves that join points of equal height. The same picture shows a function of two variables, such as a loss that depends on two parameters. For $f(x, y) = x^2 + 3y^2$ of Example 1.5, the points where $f = 4$ form the ellipse $x^2 + 3y^2 = 4$ through $(1, 1)$, and the points where $f = 1$ form a smaller ellipse inside it. Moving along one of these curves leaves f unchanged, while the gradient points in the direction of fastest increase (Theorem 1.8).

Definition 2.2 (Level set)**.** Let $U \subseteq \mathbb{R}^n$ be open, $f\colon U \to \mathbb{R}$ and c a real number. The *level set* of f at c is $\{\mathbf{x} \in U : f(\mathbf{x}) = c\}$. For $n = 2$ it is usually a curve, a *level curve* or contour line.

Example 1.5 claimed that the gradient is perpendicular to level curves. The chain rule proves it in any dimension, for every direction in which a curve can pass through a point while staying in the level set.

Theorem 2.4 (The gradient is perpendicular to level sets)**.** *Let $U \subseteq \mathbb{R}^n$ be open and $f\colon U \to \mathbb{R}$ differentiable at $\mathbf{a}$, with $f(\mathbf{a}) = c$. Let $\gamma\colon I \to U$ be a curve on an open interval I, differentiable at $t_0 \in I$, with $\gamma(t_0) = \mathbf{a}$ and $f(\gamma(t)) = c$ for every $t \in I$. Then*

$$\nabla f(\mathbf{a}) \cdot \gamma'(t_0) = 0.$$

Proof. The function $f \circ \gamma$ is constant on I, so its derivative at t_0 is 0. By Corollary 2.1 that derivative is $\nabla f(\mathbf{a}) \cdot \gamma'(t_0)$. □

So every velocity of a curve that stays in the level set is perpendicular to the gradient. When $\nabla f(\mathbf{a}) \neq \mathbf{0}$ and f is continuously differentiable near $\mathbf{a}$, the converse also holds: every vector perpendicular to $\nabla f(\mathbf{a})$ is the velocity of some curve in the level set through $\mathbf{a}$. That needs the implicit function theorem, which this book does not prove; see Zorich (2015, sec. 8.5 and

8.7.2). With the converse, the vectors perpendicular to the gradient are exactly the directions along the level set, and by Theorem 1.7 they are the directions in which f has zero directional derivative.

Example 2.5. Return to $f(x, y) = x^2 + 3y^2$ and the point $(1, 1)$ of Example 1.5, where $f = 4$. The level curve $x^2 + 3y^2 = 4$ is an ellipse, traced by $\gamma(t) = (2\cos t,\ (2/\sqrt{3})\sin t)$, and $\gamma(\pi/3) = (1, 1)$. Its velocity is $\gamma'(t) = (-2\sin t,\ (2/\sqrt{3})\cos t)$, so $\gamma'(\pi/3) = (-\sqrt{3},\ 1/\sqrt{3})$, and

$$\nabla f(1, 1) \cdot \gamma'(\pi/3) = 2(-\sqrt{3}) + 6/\sqrt{3} = 0.$$

At a general point of the ellipse, $\nabla f(\gamma(t)) = (4\cos t,\ 4\sqrt{3}\sin t)$, and the dot product with $\gamma'(t)$ is $-8\sin t\cos t + 8\sin t\cos t = 0$.

2.6. Second derivatives and the Hessian

At a minimum of a differentiable function the gradient is zero (Theorem 2.7 below), so there the linear approximation $f(\mathbf{a} + \mathbf{h}) \approx f(\mathbf{a}) + \nabla f(\mathbf{a}) \cdot \mathbf{h}$ reduces to $f(\mathbf{a})$ and says nothing about the shape of f nearby. For a function of one variable the second derivative takes over: a point where $f'(a) = 0$ and $f''(a) > 0$ is a local minimum. In several variables the gradient is itself a function with n outputs, and the role of f'' passes to the rates at which its components change.

Take $f(x, y) = x^2y$ (the example that opens Chapter 1) at $(1, 2)$. Its gradient is $\nabla f(x, y) = (2xy,\ x^2)$, equal to $(4, 1)$ at $(1, 2)$.

- Increment x alone by 0.01: $\nabla f(1.01, 2) = (4.04,\ 1.0201)$. So $\partial f/\partial x$ changes at rate 4 (its partial derivative $2y$ with respect to x) and $\partial f/\partial y$ at rate about 2 (exactly $2x = 2$ at $(1, 2)$).
- Increment y alone by 0.01: $\nabla f(1, 2.01) = (4.02,\ 1)$. So $\partial f/\partial x$ changes at rate 2 (its partial derivative $2x$ with respect to y) and $\partial f/\partial y = x^2$ does not change.

These four rates, 4, 2, 2 and 0, are the second partial derivatives of f at $(1,2)$. Two of them came out equal: the rate at which $\partial f/\partial y$ changes with x and the rate at which $\partial f/\partial x$ changes with y are both 2. Theorem 2.5 shows that this is no accident.

In general, the partial derivatives of $f\colon U \to \mathbb{R}$ are themselves functions on U, and they may have partial derivatives of their own. Write

$$\frac{\partial^2 f}{\partial x_i\,\partial x_j} = \frac{\partial}{\partial x_i}\Big(\frac{\partial f}{\partial x_j}\Big), \qquad \frac{\partial^2 f}{\partial x_i^2} = \frac{\partial}{\partial x_i}\Big(\frac{\partial f}{\partial x_i}\Big).$$

The symbol $\partial^2 f/\partial x_i\,\partial x_j$ means: differentiate with respect to x_j first, then with respect to x_i, reading the denominator from right to left. For $i \neq j$ these are *mixed* partial derivatives. A function is *twice continuously differentiable* on U if all its first and second partial derivatives exist and are continuous on U.

Definition 2.3 (Hessian matrix)**.** Let $U \subseteq \mathbb{R}^n$ be open, and let $f\colon U \to \mathbb{R}$ have all second partial derivatives at $\mathbf{a} \in U$. Its *Hessian* at $\mathbf{a}$ is the $n \times n$ matrix

$$\nabla^2 f(\mathbf{a}) = \begin{pmatrix} \dfrac{\partial^2 f}{\partial x_1^2}(\mathbf{a}) & \cdots & \dfrac{\partial^2 f}{\partial x_1\,\partial x_n}(\mathbf{a}) \\ \vdots & & \vdots \\ \dfrac{\partial^2 f}{\partial x_n\,\partial x_1}(\mathbf{a}) & \cdots & \dfrac{\partial^2 f}{\partial x_n^2}(\mathbf{a}) \end{pmatrix},$$

with entry (i,j) equal to $\partial^2 f/\partial x_i\,\partial x_j(\mathbf{a})$.

Entry (i,j) is the rate at which $\partial f/\partial x_j$ changes per unit increment of x_i. So column j lists how the partial derivative $\partial f/\partial x_j$ responds to each input, and row i lists how all the partial derivatives respond to an increment of x_i alone. For x^2y at $(1,2)$ the four rates above give

$$\nabla^2 f(1,2) = \begin{pmatrix} 4 & 2 \\ 2 & 0 \end{pmatrix}.$$

Row j of the Jacobian of the gradient map $\mathbf{x} \mapsto \nabla f(\mathbf{x})$ holds the partial derivatives of $\partial f/\partial x_j$, so $J_{\nabla f}(\mathbf{a})$ is the transpose of $\nabla^2 f(\mathbf{a})$. Theorem 2.5 below shows that for twice continuously differentiable functions the Hessian is symmetric, so the two matrices agree.

Example 2.6. For the function $f(x, y) = x^2 y + \sin(xy)$ of Example 1.2,

$$\nabla^2 f(x,y) = \begin{pmatrix} 2y - y^2\sin(xy) & 2x + \cos(xy) - xy\sin(xy) \\ 2x + \cos(xy) - xy\sin(xy) & -x^2\sin(xy) \end{pmatrix}, \qquad \nabla^2 f(1,0) = \begin{pmatrix} 0 & 3 \\ 3 & 0 \end{pmatrix}.$$

The two mixed partial derivatives, computed in different orders, came out equal. For a quadratic form $f(\mathbf{x}) = \mathbf{x}^\top A\mathbf{x}$, Example 1.8 gives $\nabla f(\mathbf{x}) = (A + A^\top)\mathbf{x}$, a linear map, so $\nabla^2 f(\mathbf{x}) = A + A^\top$ at every point, which is symmetric whether or not A is.

Theorem 2.5 (Symmetry of mixed partial derivatives)**.** *Let $f\colon U \to \mathbb{R}$, with U open, and $i \neq j$. Suppose that $\partial f/\partial x_i$, $\partial f/\partial x_j$ and both mixed partial derivatives $\partial^2 f/\partial x_i\,\partial x_j$ and $\partial^2 f/\partial x_j\,\partial x_i$ exist on an open disc around $\mathbf{a}$, and that the two mixed partial derivatives are continuous at $\mathbf{a}$. Then*

$$\frac{\partial^2 f}{\partial x_i\,\partial x_j}(\mathbf{a}) = \frac{\partial^2 f}{\partial x_j\,\partial x_i}(\mathbf{a}).$$

In particular, if f is twice continuously differentiable on U, then $\nabla^2 f(\mathbf{a})$ is symmetric for every $\mathbf{a} \in U$.

Proof. Only x_i and x_j vary in the argument, so write $\mathbf{p}(s,t) = \mathbf{a} + s\mathbf{e}_i + t\mathbf{e}_j$, write $\partial_i f$ for $\partial f/\partial x_i$, and let ρ be the radius of the disc. For $0 < |h| < \rho/2$, every point $\mathbf{p}(s,t)$ with $|s|, |t| \le |h|$ lies within $\sqrt{2}|h| < \rho$ of $\mathbf{a}$, so in the disc. Consider

$$\Delta(h) = f(\mathbf{p}(h,h)) - f(\mathbf{p}(h,0)) - f(\mathbf{p}(0,h)) + f(\mathbf{a}).$$

First group it as $\varphi(h) - \varphi(0)$ with $\varphi(s) = f(\mathbf{p}(s,h)) - f(\mathbf{p}(s,0))$. The function φ is differentiable for $|s| \le |h|$, with $\varphi'(s) = \partial_i f(\mathbf{p}(s,h)) - \partial_i f(\mathbf{p}(s,0))$.

By the mean value theorem, $\Delta(h) = h\varphi'(s_1)$ for some s_1 between 0 and h. The function $t \mapsto \partial_i f(\mathbf{p}(s_1, t))$ has derivative $\partial^2 f / \partial x_j \, \partial x_i$, so a second application of the mean value theorem gives a t_1 between 0 and h with

$$\Delta(h) = h^2 \, \frac{\partial^2 f}{\partial x_j \, \partial x_i}(\mathbf{p}(s_1, t_1)).$$

Grouping $\Delta(h)$ instead as $\psi(h) - \psi(0)$ with $\psi(t) = f(\mathbf{p}(h, t)) - f(\mathbf{p}(0, t))$ and repeating the argument with the roles of i and j exchanged gives s_2 and t_2 between 0 and h with

$$\Delta(h) = h^2 \, \frac{\partial^2 f}{\partial x_i \, \partial x_j}(\mathbf{p}(s_2, t_2)).$$

Divide both expressions by h^2 and let $h \to 0$. Both points $\mathbf{p}(s_1, t_1)$ and $\mathbf{p}(s_2, t_2)$ lie within $\sqrt{2}|h|$ of $\mathbf{a}$, and both mixed partial derivatives are continuous at $\mathbf{a}$, so the two limits are the two mixed partial derivatives at $\mathbf{a}$, and they are equal. □

Without continuity the conclusion can fail, as Exercise 2.5 shows.

2.7. Taylor's theorem

In one variable, the Taylor polynomial of degree 2 improves on the linear approximation: $f(a + h) \approx f(a) + f'(a)\, h + \frac{1}{2} f''(a)\, h^2$. In several variables the linear term becomes $\nabla f(\mathbf{a}) \cdot \mathbf{h}$, and the quadratic term becomes $\frac{1}{2}\, \mathbf{h}^\top \nabla^2 f(\mathbf{a})\, \mathbf{h}$, where

$$\mathbf{h}^\top \nabla^2 f(\mathbf{a})\, \mathbf{h} = \sum_{i=1}^{n} \sum_{j=1}^{n} h_i h_j \, \frac{\partial^2 f}{\partial x_i \, \partial x_j}(\mathbf{a})$$

is a quadratic form in the increment $\mathbf{h}$, as in Example 1.8: each product $h_i h_j$ of two input increments, weighted by the matching second partial derivative.

The function $f(x,y) = x^2y$ shows the improvement. Expanding,

$$f(1+h_1, 2+h_2) - f(1,2) = 4h_1 + h_2 + 2h_1^2 + 2h_1h_2 + h_1^2h_2.$$

The first two terms are $\nabla f(1,2) \cdot \mathbf{h}$, with $\nabla f(1,2) = (4,1)$. With $\nabla^2 f(1,2) = \begin{pmatrix} 4 & 2 \\ 2 & 0 \end{pmatrix}$ from Section 2.6, the quadratic term is $\frac{1}{2}(4h_1^2 + 2 \cdot 2h_1h_2 + 0 \cdot h_2^2) = 2h_1^2 + 2h_1h_2$, the next two terms. Only $h_1^2h_2$, a product of three increments, is left over. At $\mathbf{h} = (0.01, 0.01)$, $f(1.01, 2.01) = 2.050401$: the linear approximation 2.05 is off by 0.000401, and the quadratic approximation 2.0504 by 0.000001.

Adding the quadratic term gives a better approximation, and that term decides what kind of point a zero of the gradient is (Section 2.8). The proof reduces to one variable by restricting f to a segment.

Lemma 2.3 (Taylor's theorem in one variable, second order)**.** *Let g be twice differentiable on an open interval containing $[0,1]$. Then there is a $\tau \in (0,1)$ with*

$$g(1) = g(0) + g'(0) + \tfrac{1}{2}g''(\tau).$$

Proof. Choose the constant M so that $k(t) = g(1) - g(t) - g'(t)(1-t) - M(1-t)^2$ satisfies $k(0) = 0$. Also $k(1) = 0$, and k is differentiable, with

$$k'(t) = -g'(t) - g''(t)(1-t) + g'(t) + 2M(1-t) = (1-t)(2M - g''(t)).$$

By Rolle's theorem $k'(\tau) = 0$ for some $\tau \in (0,1)$, and since $1 - \tau \neq 0$, $M = \frac{1}{2}g''(\tau)$. Then $k(0) = 0$ reads $g(1) = g(0) + g'(0) + \frac{1}{2}g''(\tau)$. □

Theorem 2.6 (Taylor's theorem in several variables, second order)**.** *Let $U \subseteq \mathbb{R}^n$ be open and $f\colon U \to \mathbb{R}$ twice continuously differentiable on U. If $\mathbf{a} \in U$ and the segment from $\mathbf{a}$ to $\mathbf{a}+\mathbf{h}$, the set of points $\mathbf{a}+t\mathbf{h}$ with $0 \le t \le 1$, lies in U, there is a $\tau \in (0,1)$ with*

$$f(\mathbf{a}+\mathbf{h}) = f(\mathbf{a}) + \nabla f(\mathbf{a}) \cdot \mathbf{h} + \tfrac{1}{2}\,\mathbf{h}^{\top} \nabla^2 f(\mathbf{a}+\tau\mathbf{h})\,\mathbf{h}. \tag{2.6}$$

Consequently, for every $\mathbf{a} \in U$ *and every* $\mathbf{h}$ *with* $\mathbf{a}+\mathbf{h} \in U$,

$$f(\mathbf{a}+\mathbf{h}) = f(\mathbf{a}) + \nabla f(\mathbf{a}) \cdot \mathbf{h} + \tfrac{1}{2}\,\mathbf{h}^{\top} \nabla^2 f(\mathbf{a})\,\mathbf{h} + R(\mathbf{h}), \qquad \frac{|R(\mathbf{h})|}{\|\mathbf{h}\|^2} \to 0 \text{ as } \mathbf{h} \to \mathbf{0}. \tag{2.7}$$

Proof. The set of t with $\mathbf{a}+t\mathbf{h} \in U$ is open, because U is, and it contains $[0,1]$. On it define $g(t) = f(\mathbf{a}+t\mathbf{h})$. By Theorem 1.6, f is differentiable on U, so by Corollary 2.1, with the curve $t \mapsto \mathbf{a}+t\mathbf{h}$ of velocity $\mathbf{h}$,

$$g'(t) = \nabla f(\mathbf{a}+t\mathbf{h}) \cdot \mathbf{h} = \sum_{j=1}^{n} h_j \, \frac{\partial f}{\partial x_j}(\mathbf{a}+t\mathbf{h}).$$

Each $\partial f/\partial x_j$ has continuous partial derivatives, so it is differentiable by Theorem 1.6, and the same corollary gives

$$g''(t) = \sum_{j=1}^{n} h_j \sum_{i=1}^{n} h_i \, \frac{\partial^2 f}{\partial x_i \, \partial x_j}(\mathbf{a}+t\mathbf{h}) = \mathbf{h}^{\top} \nabla^2 f(\mathbf{a}+t\mathbf{h})\,\mathbf{h}.$$

Lemma 2.3, applied on an open interval containing $[0,1]$ inside the domain of g, gives Equation 2.6, since $g(0) = f(\mathbf{a})$, $g(1) = f(\mathbf{a}+\mathbf{h})$ and $g'(0) = \nabla f(\mathbf{a}) \cdot \mathbf{h}$.

For Equation 2.7, let $\rho > 0$ be the radius of a disc around $\mathbf{a}$ inside U. For $\|\mathbf{h}\| < \rho$ the segment lies in the disc, and subtracting Equation 2.7 from Equation 2.6 gives $R(\mathbf{h}) = \frac{1}{2}\mathbf{h}^{\top} E\,\mathbf{h}$ with $E = \nabla^2 f(\mathbf{a}+\tau\mathbf{h}) - \nabla^2 f(\mathbf{a})$. By Theorem 1.1 and Lemma 1.1, $|R(\mathbf{h})| \le \frac{1}{2}\|\mathbf{h}\|\,\|E\mathbf{h}\| \le \frac{1}{2}\|E\|_F\,\|\mathbf{h}\|^2$. The point $\mathbf{a}+\tau\mathbf{h}$ is within $\|\mathbf{h}\|$ of $\mathbf{a}$, and every second partial derivative is continuous at $\mathbf{a}$, so each entry of E, and therefore $\|E\|_F$, tends to 0 as $\mathbf{h} \to \mathbf{0}$. □

The two forms of the remainder serve different purposes. Equation 2.6 is exact, but it evaluates the Hessian at an unknown point $\mathbf{a}+\tau\mathbf{h}$ of the segment; this is the *Lagrange form*, and bounds on the Hessian along the segment turn it into bounds on f. Equation 2.7 evaluates the Hessian at $\mathbf{a}$

itself and says only that the remainder $R(\mathbf{h})$ is small compared with $\|\mathbf{h}\|^2$, as the remainder in Definition 1.6 is small compared with $\|\mathbf{h}\|$; this is the *Peano form*, and it is what the second-derivative test uses.

The open interval used for Equation 2.6 exists because the set of t with $\mathbf{a}+t\mathbf{h}\in U$ contains a small open interval around each of 0 and 1, and the whole of $[0,1]$ in between. The computation of g'' did not need the Hessian to be symmetric: the double sum counts entry (i,j) and entry (j,i) with the same weight h_ih_j.

Example 2.7. Let $f(x,y)=e^x\cos y$ and $\mathbf{a}=\mathbf{0}$. Then $f(\mathbf{0})=1$, $\nabla f(\mathbf{0})=(1,0)$ and

$$\nabla^2 f(x,y)=\begin{pmatrix} e^x\cos y & -e^x\sin y\\ -e^x\sin y & -e^x\cos y\end{pmatrix},\qquad \nabla^2 f(\mathbf{0})=\begin{pmatrix}1&0\\0&-1\end{pmatrix},$$

so Equation 2.7 reads $e^{h_1}\cos h_2 = 1+h_1+\frac{1}{2}(h_1^2-h_2^2)+R(\mathbf{h})$. For $\mathbf{h}=(0.1,0.2)$ the quadratic approximation is 1.085 and the true value is $1.08314\ldots$, an error of about 0.0019, small compared with $\|\mathbf{h}\|^2=0.05$. Halving $\mathbf{h}$ divides the error by about 8, so it shrinks like $\|\mathbf{h}\|^3$, faster than Equation 2.7 requires.

2.8. Critical points and the second-derivative test

Training a model searches for parameters at which the loss is smaller than at every nearby point. In one variable, such a point of a differentiable function has $f'(a)=0$, and conversely a point with $f'(a)=0$ and $f''(a)>0$ is one. This section carries both facts to several variables, with the gradient in place of f' and the Hessian in place of f''. The bowl $f(x,y)=x^2+3y^2$ of Example 1.5 shows what to expect: $f(\mathbf{0})=0$ and $f>0$ everywhere else, and its gradient $(2x,6y)$ is zero at the origin. Its Hessian $\operatorname{diag}(2,6)$ has positive entries on the diagonal and zeros off it.

Definition 2.4 (Local extrema and critical points)**.** Let $f\colon U \to \mathbb{R}$ with U open. A point $\mathbf{a} \in U$ is a *local minimum* of f if there is an $r > 0$ with $f(\mathbf{x}) \geq f(\mathbf{a})$ whenever $\|\mathbf{x} - \mathbf{a}\| < r$, and a *strict local minimum* if moreover $f(\mathbf{x}) > f(\mathbf{a})$ whenever $0 < \|\mathbf{x} - \mathbf{a}\| < r$. Reversing the inequalities defines local maxima and strict local maxima. A *critical point* of f is a point where f is differentiable and $\nabla f(\mathbf{a}) = \mathbf{0}$.

Theorem 2.7. *Let $U \subseteq \mathbb{R}^n$ be open and $f\colon U \to \mathbb{R}$. If f is differentiable at $\mathbf{a} \in U$ and $\mathbf{a}$ is a local minimum or a local maximum of f, then $\nabla f(\mathbf{a}) = \mathbf{0}$.*

Proof. For each j, the function $t \mapsto f(\mathbf{a} + t\mathbf{e}_j)$ is defined on an open interval around 0 and has a local minimum or maximum at $t = 0$, so its derivative there, $\partial f/\partial x_j(\mathbf{a})$, is 0. □

A critical point need not be either. The function $x^2 - y^2$ has a critical point at the origin, but it increases along the x-axis and decreases along the y-axis. To sort critical points, look at Equation 2.7. At a critical point the gradient term vanishes, and

$$f(\mathbf{a} + \mathbf{h}) - f(\mathbf{a}) = \tfrac{1}{2}\,\mathbf{h}^\top H \mathbf{h} + R(\mathbf{h}), \qquad H = \nabla^2 f(\mathbf{a}),$$

so for small $\mathbf{h}$ the sign of the quadratic form $\mathbf{h}^\top H \mathbf{h}$ should decide the sign of the increment of f. For a diagonal matrix the sign is easy to read: $\operatorname{diag}(\lambda_1, \lambda_2)$ gives $\lambda_1 h_1^2 + \lambda_2 h_2^2$, which is positive for every $\mathbf{h} \neq \mathbf{0}$ exactly when both λ_1 and λ_2 are positive. For other matrices the entries can mislead. The matrix

$$H = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \quad \text{gives} \quad \mathbf{h}^\top H \mathbf{h} = h_1^2 + 4h_1 h_2 + h_2^2,$$

which is 6 at $\mathbf{h} = (1, 1)$ but -2 at $\mathbf{h} = (1, -1)$, although every entry of H is positive. Measuring $\mathbf{h}$ along the perpendicular unit vectors $\mathbf{u}_1 = (1, 1)/\sqrt{2}$ and $\mathbf{u}_2 = (1, -1)/\sqrt{2}$ instead of along the axes removes the cross term. With $c_1 = \mathbf{u}_1 \cdot \mathbf{h} = (h_1 + h_2)/\sqrt{2}$ and $c_2 = \mathbf{u}_2 \cdot \mathbf{h} = (h_1 - h_2)/\sqrt{2}$,

$$\mathbf{h}^\top H \mathbf{h} = 3c_1^2 - c_2^2.$$

In these coordinates H acts like $\operatorname{diag}(3, -1)$, and the form is positive along $\mathbf{u}_1$ and negative along $\mathbf{u}_2$. The vectors $\mathbf{u}_1$ and $\mathbf{u}_2$ are eigenvectors of H, with eigenvalues 3 and -1: $H\mathbf{u}_1 = 3\mathbf{u}_1$ and $H\mathbf{u}_2 = -\mathbf{u}_2$. The rest of this section shows that every symmetric matrix, and so every Hessian of a twice continuously differentiable function, has such perpendicular eigenvectors, and then reads the type of a critical point from the signs of the eigenvalues.

2.8.1. Symmetric matrices and their eigenvalues

A real matrix can have complex eigenvalues: a rotation of the plane through a right angle has eigenvalues $\pm i$. A symmetric matrix cannot, and its eigenvectors can be chosen perpendicular to each other. For 2×2 matrices this is a calculation.

Theorem 2.8 (Symmetric 2×2 matrices)**.** *Let* $A = \begin{pmatrix} a & b \\ b & d \end{pmatrix}$ *with* a*,* b*,* d *real. Then the eigenvalues of* A *are real, say* $\lambda_1 \geq \lambda_2$*, and there are unit vectors* $\mathbf{u}_1$ *and* $\mathbf{u}_2$ *with* $\mathbf{u}_1 \cdot \mathbf{u}_2 = 0$ *and* $A\mathbf{u}_k = \lambda_k \mathbf{u}_k$*.*

Proof. The eigenvalues are the roots of the characteristic polynomial $\lambda^2 - (a+d)\lambda + (ad - b^2)$, whose discriminant is

$$(a+d)^2 - 4(ad - b^2) = (a-d)^2 + 4b^2 \geq 0.$$

So both roots are real.

If the discriminant is 0, then $a = d$ and $b = 0$, so $A = aI$, and $\mathbf{u}_1 = \mathbf{e}_1$, $\mathbf{u}_2 = \mathbf{e}_2$ will do.

Otherwise $\lambda_1 > \lambda_2$. Each $A - \lambda_k I$ has determinant 0, so there are non-zero vectors $\mathbf{v}_k$ with $A\mathbf{v}_k = \lambda_k \mathbf{v}_k$. Since $A^\top = A$,

$$\lambda_1 \, \mathbf{v}_1 \cdot \mathbf{v}_2 = (A\mathbf{v}_1) \cdot \mathbf{v}_2 = \mathbf{v}_1^\top A^\top \mathbf{v}_2 = \mathbf{v}_1 \cdot (A\mathbf{v}_2) = \lambda_2 \, \mathbf{v}_1 \cdot \mathbf{v}_2,$$

so $(\lambda_1 - \lambda_2) \, \mathbf{v}_1 \cdot \mathbf{v}_2 = 0$ and $\mathbf{v}_1 \cdot \mathbf{v}_2 = 0$. Take $\mathbf{u}_k = \mathbf{v}_k / \|\mathbf{v}_k\|$. □

The orthogonality argument works for symmetric matrices of any size. The existence of real eigenvalues and of a full set of eigenvectors holds in any size too, but its proof needs more linear algebra than this book assumes.

Theorem 2.9 (Spectral theorem)**.** *Let A be a real symmetric $n \times n$ matrix. Then there are real numbers $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$ and an orthonormal basis $\mathbf{u}_1, \dots, \mathbf{u}_n$ of $\mathbb{R}^n$ (unit vectors, each perpendicular to the others) with $A\mathbf{u}_k = \lambda_k \mathbf{u}_k$ for every k. The λ_k are the eigenvalues of A, each listed as many times as it is a root of the characteristic polynomial* $\det(A - \lambda I)$.

The book does not prove Theorem 2.9 for $n > 2$; see Axler (2024, result 7.29) or Deisenroth et al. (2020, Theorem 4.15).

Corollary 2.4 (Quadratic forms through eigenvalues)**.** *Let A be a real symmetric $n \times n$ matrix, with eigenvalues $\lambda_1 \geq \dots \geq \lambda_n$ and orthonormal eigenvectors $\mathbf{u}_1, \dots, \mathbf{u}_n$ as in Theorem 2.9. Then every $\mathbf{h} \in \mathbb{R}^n$ satisfies*

$$\mathbf{h}^\top A \mathbf{h} = \sum_{k=1}^{n} \lambda_k (\mathbf{u}_k \cdot \mathbf{h})^2 \qquad \text{and} \qquad \lambda_n \|\mathbf{h}\|^2 \leq \mathbf{h}^\top A \mathbf{h} \leq \lambda_1 \|\mathbf{h}\|^2,$$

with equality on the left at $\mathbf{h} = \mathbf{u}_n$ and on the right at $\mathbf{h} = \mathbf{u}_1$.

Proof. Write $\mathbf{h} = \sum_k c_k \mathbf{u}_k$, which is possible because the $\mathbf{u}_k$ form a basis. Taking the dot product with $\mathbf{u}_l$ and using orthonormality gives $c_l = \mathbf{u}_l \cdot \mathbf{h}$. Then $A\mathbf{h} = \sum_k c_k \lambda_k \mathbf{u}_k$, and expanding the dot products,

$$\mathbf{h}^\top A \mathbf{h} = \sum_{k,l} c_l c_k \lambda_k \, \mathbf{u}_l \cdot \mathbf{u}_k = \sum_k \lambda_k c_k^2, \qquad \|\mathbf{h}\|^2 = \sum_{k,l} c_l c_k \, \mathbf{u}_l \cdot \mathbf{u}_k = \sum_k c_k^2.$$

Each λ_k lies between λ_n and λ_1, which gives the inequalities, and $\mathbf{h} = \mathbf{u}_1$ has $c_1 = 1$ and every other $c_k = 0$, and similarly for $\mathbf{u}_n$. □

The number $\mathbf{u}_k \cdot \mathbf{h}$ is the component of $\mathbf{h}$ along the eigenvector $\mathbf{u}_k$, so the first formula writes the quadratic form as a sum of squares of these

components, each weighted by its eigenvalue, as $3c_1^2 - c_2^2$ did for $\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$. The bounds say that for $\mathbf{h} \neq \mathbf{0}$ the form, divided by $\|\mathbf{h}\|^2$, lies between the smallest and the largest eigenvalue.

The sign of the quadratic form is what the second-derivative test needs. The Hessian $\operatorname{diag}(2, 6)$ of the bowl $x^2 + 3y^2$ gives $2h_1^2 + 6h_2^2$, positive for every $\mathbf{h} \neq \mathbf{0}$, while the form of $\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$ takes both signs. Matrices of the first kind have a name.

Definition 2.5 (Definite matrices)**.** A real symmetric $n \times n$ matrix A is *positive definite* if $\mathbf{h}^\top A\mathbf{h} > 0$ for every $\mathbf{h} \neq \mathbf{0}$ in $\mathbb{R}^n$, *positive semidefinite* if $\mathbf{h}^\top A\mathbf{h} \geq 0$ for every $\mathbf{h}$, *negative definite* if $\mathbf{h}^\top A\mathbf{h} < 0$ for every $\mathbf{h} \neq \mathbf{0}$, and *negative semidefinite* if $\mathbf{h}^\top A\mathbf{h} \leq 0$ for every $\mathbf{h}$.

By Corollary 2.4, with $\lambda_1 \geq \ldots \geq \lambda_n$ the eigenvalues of A, the form is at least $\lambda_n\|\mathbf{h}\|^2$ and equals λ_n at $\mathbf{h} = \mathbf{u}_n$. So A is positive definite exactly when all its eigenvalues are positive, and positive semidefinite exactly when none is negative. Applying this to $-A$, it is negative definite exactly when all its eigenvalues are negative, and negative semidefinite exactly when none is positive. The bowl's Hessian, with eigenvalues 2 and 6, is positive definite. The matrix $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ gives the form $(h_1 + h_2)^2$ and has eigenvalues 2 and 0, so it is positive semidefinite but not positive definite. The matrix $\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$, with eigenvalues 3 and -1, is neither positive nor negative semidefinite.

2.8.2. The test

At a critical point, Equation 2.7 writes the increment of f as half the quadratic form of the Hessian plus a remainder that is small compared with $\|\mathbf{h}\|^2$. By Corollary 2.4 the quadratic form is at least $\lambda_n\|\mathbf{h}\|^2$, so

when $\lambda_n > 0$ it outweighs the remainder for small $\mathbf{h}$, and the increment is positive. This is the one-variable test with the smallest eigenvalue in place of $f''(a)$.

Theorem 2.10 (Second-derivative test)**.** *Let $U \subseteq \mathbb{R}^n$ be open, $f\colon U \to \mathbb{R}$ twice continuously differentiable on U, and $\mathbf{a}$ a critical point of f. Let $\lambda_1 \geq \dots \geq \lambda_n$ be the eigenvalues of the symmetric matrix $\nabla^2 f(\mathbf{a})$.*

(a) If $\nabla^2 f(\mathbf{a})$ is positive definite (Definition 2.5), that is, every eigenvalue is positive ($\lambda_n > 0$), then $\mathbf{a}$ is a strict local minimum.

(b) If $\nabla^2 f(\mathbf{a})$ is negative definite, that is, every eigenvalue is negative ($\lambda_1 < 0$), then $\mathbf{a}$ is a strict local maximum.

(c) If $\lambda_1 > 0 > \lambda_n$, then $\mathbf{a}$ is a saddle point*: every disc around $\mathbf{a}$ contains points where $f > f(\mathbf{a})$ and points where $f < f(\mathbf{a})$, so $\mathbf{a}$ is neither a local minimum nor a local maximum.*

Proof. Write $H = \nabla^2 f(\mathbf{a})$, which is symmetric by Theorem 2.5, and let $\mathbf{u}_1, \dots, \mathbf{u}_n$ be orthonormal eigenvectors from Theorem 2.9. Since $\nabla f(\mathbf{a}) = \mathbf{0}$, Equation 2.7 reads

$$f(\mathbf{a}+\mathbf{h}) - f(\mathbf{a}) = \tfrac{1}{2}\mathbf{h}^\top H \mathbf{h} + R(\mathbf{h}), \qquad \frac{|R(\mathbf{h})|}{\|\mathbf{h}\|^2} \to 0.$$

(a) Choose $r > 0$ so that the disc of radius r around $\mathbf{a}$ lies in U and $|R(\mathbf{h})| < \tfrac{1}{4}\lambda_n \|\mathbf{h}\|^2$ whenever $0 < \|\mathbf{h}\| < r$. By Corollary 2.4, for these $\mathbf{h}$,

$$f(\mathbf{a}+\mathbf{h}) - f(\mathbf{a}) > \tfrac{1}{2}\lambda_n\|\mathbf{h}\|^2 - \tfrac{1}{4}\lambda_n\|\mathbf{h}\|^2 = \tfrac{1}{4}\lambda_n\|\mathbf{h}\|^2 > 0.$$

(b) Apply (a) to $-f$, whose Hessian $-H$ has eigenvalues $-\lambda_n \geq \dots \geq -\lambda_1 > 0$.

(c) Choose $r > 0$ as in (a) but with $|R(\mathbf{h})| < \tfrac{1}{4}\min(\lambda_1, -\lambda_n)\|\mathbf{h}\|^2$. For $0 < |t| < r$, $\mathbf{h} = t\mathbf{u}_1$ has $\mathbf{h}^\top H \mathbf{h} = \lambda_1 t^2$, so $f(\mathbf{a}+t\mathbf{u}_1) - f(\mathbf{a}) > \tfrac{1}{2}\lambda_1 t^2 - \tfrac{1}{4}\lambda_1 t^2 > 0$. In the same way $f(\mathbf{a}+t\mathbf{u}_n) - f(\mathbf{a}) < \tfrac{1}{2}\lambda_n t^2 - \tfrac{1}{4}\lambda_n t^2 < 0$.

□

The test says nothing when some eigenvalue is 0 and the others all have the same sign. The functions $x^4 + y^4$ and $x^4 - y^4$ both have zero Hessian at the origin, which is a minimum of the first and a saddle point of the second. Exercise 2.8 gives a pair of examples with a non-zero Hessian.

In two variables the eigenvalues need not be computed. Write f_{xx}, f_{xy} and f_{yy} for the second partial derivatives $\partial^2 f/\partial x^2$, $\partial^2 f/\partial x\,\partial y$ and $\partial^2 f/\partial y^2$ at $\mathbf{a}$. For a twice continuously differentiable f the Hessian is symmetric (Theorem 2.5), so $H = \begin{pmatrix} f_{xx} & f_{xy} \\ f_{xy} & f_{yy} \end{pmatrix}$. The characteristic polynomial of H is $\lambda^2 - (f_{xx} + f_{yy})\lambda + (f_{xx}f_{yy} - f_{xy}^2)$, so $\lambda_1\lambda_2 = \det H = f_{xx}f_{yy} - f_{xy}^2$ and $\lambda_1 + \lambda_2 = f_{xx} + f_{yy}$.

Corollary 2.5 (The test in two variables)**.** *Let $U \subseteq \mathbb{R}^2$ be open, $f\colon U \to \mathbb{R}$ twice continuously differentiable on U, and $\mathbf{a}$ a critical point of f, and write $\nabla^2 f(\mathbf{a}) = \begin{pmatrix} f_{xx} & f_{xy} \\ f_{xy} & f_{yy} \end{pmatrix}$.*

(a) If $f_{xx}f_{yy} - f_{xy}^2 > 0$ and $f_{xx} > 0$, $\mathbf{a}$ is a strict local minimum.

(b) If $f_{xx}f_{yy} - f_{xy}^2 > 0$ and $f_{xx} < 0$, $\mathbf{a}$ is a strict local maximum.

(c) If $f_{xx}f_{yy} - f_{xy}^2 < 0$, $\mathbf{a}$ is a saddle point.

Proof. If $f_{xx}f_{yy} - f_{xy}^2 < 0$ the product of the eigenvalues is negative, so one is positive and the other negative, and part (c) of Theorem 2.10 applies. If $f_{xx}f_{yy} - f_{xy}^2 > 0$ the eigenvalues have the same sign, and so does their sum $f_{xx} + f_{yy}$. Also $f_{xx}f_{yy} > f_{xy}^2 \geq 0$, so f_{xx} and f_{yy} have the same sign, which is therefore the sign of $f_{xx} + f_{yy}$ and of both eigenvalues. □

Example 2.8. Let $f(x, y) = x^3 - 3xy + y^3$. Then $\nabla f = (3x^2 - 3y,\ 3y^2 - 3x)$, so a critical point has $y = x^2$ and $x = y^2$. Hence $x = x^4$, so $x = 0$ or $x = 1$, and the critical points are $(0, 0)$ and $(1, 1)$. The Hessian is

$$\nabla^2 f(x, y) = \begin{pmatrix} 6x & -3 \\ -3 & 6y \end{pmatrix}.$$

At $(0, 0)$ it has eigenvalues 3 and -3, with eigenvectors $(1, -1)/\sqrt{2}$ and $(1, 1)/\sqrt{2}$, so the origin is a saddle point. The eigenvectors show where to look: for small $t \neq 0$, $f(t, -t) = 3t^2$ is positive and $f(t, t) = 2t^3 - 3t^2$ is negative. At $(1, 1)$ the Hessian $\begin{pmatrix} 6 & -3 \\ -3 & 6 \end{pmatrix}$ has eigenvalues 9 and 3, both positive, so $(1, 1)$ is a strict local minimum, with $f(1, 1) = -1$. The two-variable test agrees: $f_{xx} f_{yy} - f_{xy}^2 = -9$ at the origin, and 27 at $(1, 1)$, where $f_{xx} = 6 > 0$. The local minimum is not a global one, since $f(x, 0) = x^3$ takes every real value.

Near a strict local minimum, the eigenvalues of the Hessian describe the shape of the valley: along $\mathbf{u}_k$ the function rises like $\frac{1}{2}\lambda_k t^2$. When the largest eigenvalue is much bigger than the smallest, the valley is long and narrow, as for $x^2 + 3y^2$ in Example 1.5, whose Hessian has eigenvalues 6 and 2. Chapter 5 shows that the ratio λ_1/λ_n governs how fast gradient descent converges.

2.9. Notes and further reading

Zorich (2015) proves the chain rule for maps between spaces of any dimension in sec. 8.3.2 and Taylor's formula in several variables in sec. 8.4.4, and studies extrema of functions of several variables in sec. 8.4.5. Zorich (2016, secs. 10.5–10.6) gives the same results for maps between normed spaces, where the derivative of each order is a multilinear map. Deisenroth et al. (2020, ch. 5) covers the chain rule and Taylor series from the point of view of machine learning. Its sec. 5.6 writes a computation as a sequence

of intermediate variables, each a function of its parents, as Definition 2.1 does, and applies the chain rule to it one step at a time. The proof of the spectral theorem for general n uses inner product spaces; Axler (2024, sec. 7B) gives one.

2.10. Exercises

Exercise 2.1. Let $f(x,y) = (xy,\ x+y)$ and $g(u,v) = (u^2+v,\ \sin u)$. Compute $J_{g\circ f}(1,2)$ in two ways: as the product $J_g(f(1,2))\,J_f(1,2)$, and by writing out $g\circ f$ and differentiating directly.

Exercise 2.2. A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is *homogeneous of degree k* if $f(t\mathbf{x}) = t^k f(\mathbf{x})$ for all $t>0$ and all $\mathbf{x}$. Show that a differentiable function that is homogeneous of degree k satisfies $\mathbf{x}\cdot\nabla f(\mathbf{x}) = k\,f(\mathbf{x})$ (Euler's theorem). Check it for $f(x,y) = x^2y - 4y^3$.

Exercise 2.3.

(a) Let $U\subseteq\mathbb{R}^n$ be open and $\mathbf{f},\mathbf{g}\colon U\to\mathbb{R}^m$ differentiable at $\mathbf{a}$. Show that $h(\mathbf{x}) = \mathbf{f}(\mathbf{x})\cdot\mathbf{g}(\mathbf{x})$ is differentiable at $\mathbf{a}$ with $J_h(\mathbf{a}) = \mathbf{g}(\mathbf{a})^{\top}J_{\mathbf{f}}(\mathbf{a}) + \mathbf{f}(\mathbf{a})^{\top}J_{\mathbf{g}}(\mathbf{a})$.

(b) Deduce that if a differentiable curve γ stays on the sphere $\|\mathbf{x}\| = 1$, then $\gamma(t)\cdot\gamma'(t) = 0$ for every t.

Exercise 2.4. Fix $k\ge 1$ and compute, from an input $v_0 = x$, for $s = 1,\dots,k$,

$$p_s = v_{s-1}^2, \qquad q_s = \sin v_{s-1}, \qquad v_s = p_s + q_s.$$

(a) Show that there are 2^k paths from x to v_k.

(b) Use Theorem 2.2 to show that $dv_k/dx = \prod_{s=1}^{k}(2v_{s-1}+\cos v_{s-1})$, a product that takes $k-1$ multiplications once its factors are known.

Exercise 2.5. Let $f(x,y) = xy(x^2-y^2)/(x^2+y^2)$ for $(x,y) \neq (0,0)$ and $f(0,0) = 0$.

(a) Show that $\partial f/\partial x(0,y) = -y$ and $\partial f/\partial y(x,0) = x$ for all x and y.

(b) Deduce that $\partial^2 f/\partial y\,\partial x(0,0) = -1$ and $\partial^2 f/\partial x\,\partial y(0,0) = 1$. Which hypothesis of Theorem 2.5 fails?

Exercise 2.6. Let $\mathbf{w} \in \mathbb{R}^n$ be non-zero, $b \in \mathbb{R}$, and $\sigma\colon \mathbb{R} \to \mathbb{R}$ twice continuously differentiable. A single neuron computes $f(\mathbf{x}) = \sigma(\mathbf{w}\cdot\mathbf{x} + b)$.

(a) Show that $\nabla f(\mathbf{x}) = \sigma'(z)\,\mathbf{w}$ and $\nabla^2 f(\mathbf{x}) = \sigma''(z)\,\mathbf{w}\mathbf{w}^\top$, where $z = \mathbf{w}\cdot\mathbf{x} + b$.

(b) Find the eigenvalues of $\mathbf{w}\mathbf{w}^\top$, and deduce that for $n \geq 2$ the Hessian of f always has the eigenvalue 0.

Exercise 2.7. Find and classify the critical points of $f(x,y) = x^4 + y^4 - 4xy$.

Exercise 2.8. Show that $f(x,y) = x^2 + y^4$ and $g(x,y) = x^2 - y^4$ have the same Hessian at the origin, that the origin is a critical point of both, and that it is a strict local minimum of f and a saddle point of g. Why does Theorem 2.10 not decide either case?

Part II.

Automatic differentiation

3. Computational graphs and forward mode

A computer evaluates a formula one elementary operation at a time, an addition or an exponential, say. Each of these operations has a derivative that we know in closed form, and Chapter 2 showed how to combine them: by Theorem 2.2, a partial derivative of the whole computation is a sum, over paths through the intermediate variables, of products of local partial derivatives. Automatic differentiation is the bookkeeping that does this combining, inside the computer, while it evaluates the formula. It gives derivatives that are exact apart from rounding, and each pass through the computation costs a constant multiple of one evaluation of the function.

This chapter writes a function as a list of elementary steps, an evaluation trace, and as the directed graph that the list describes. It then carries a derivative along with each value through the list. That is *forward mode*, and dual numbers, an extension of the real numbers by a new element ε with $\varepsilon^2 = 0$, give it an algebraic form. Forward mode computes one directional derivative per pass, so a gradient with respect to n inputs costs n passes. Chapter 4 removes that factor of n.

3.1. Evaluation traces

How does a computer evaluate $f(x, y) = x^2 y + \sin(xy)$, the function of Example 1.2? It cannot work with the formula as a whole. It computes one

operation at a time and gives each result a name, so that later operations can refer to it. At $(x, y) = (2, 0.5)$ the steps are those of Table 3.1.

Table 3.1.: Evaluating $x^2y + \sin(xy)$ at $(2, 0.5)$ one operation at a time, with values rounded to six decimal places.

variable	operation	value
v_1	x	2
v_2	y	0.5
v_3	v_1^2	4
v_4	$v_3 v_2$	2
v_5	$v_1 v_2$	1
v_6	$\sin v_5$	0.841471
v_7	$v_4 + v_6$	2.841471

So $f(2, 0.5) = 2 + \sin 1 \approx 2.841471$. Each row applies one operation to variables computed in earlier rows. The variable v_1 is an argument of two later rows, v_3 and v_5, and so is v_2: the table computes v_1 once and reuses it, where the formula mentions x twice.

The operations in such a list come from a short repertoire. The *elementary operations* of this book are addition, subtraction, multiplication and division of two numbers; adding a constant or multiplying by a constant; powers $t \mapsto t^c$ with a constant exponent; and exp, log, sin and cos. Each is a function of one or two variables with continuous partial derivatives on an open domain (division needs a non-zero denominator, and $\log t$ and non-integer powers t^c need $t > 0$). So by Theorem 1.6 each is differentiable at every point of its domain, with partial derivatives we can write down. The function $\max(t, 0)$, which has a corner at 0, waits until Chapter 7.

A list like Table 3.1 is a system of intermediate variables in the sense of Definition 2.1, in which every step is an elementary operation.

Definition 3.1 (Evaluation trace)**.** An *evaluation trace* with inputs $x_1, \dots, x_n$ is a system of intermediate variables $v_1, \dots, v_N$ (Definition 2.1) with inputs $v_j = x_j$ for $j \le n$, in which each step

$$v_k = \varphi_k\big((v_i)_{i \in P(k)}\big) \qquad (k = n+1, \dots, N)$$

applies an elementary operation φ_k, together with a choice of m of the variables as *outputs* $y_1, \dots, y_m$. We call step k *line* k of the trace, and the variables v_i with $i \in P(k)$ its *arguments*. The trace is *defined* at $\mathbf{a} \in \mathbb{R}^n$ if, with $\mathbf{x} = \mathbf{a}$, every line applies its operation at a point of the operation's domain. On the set D of such $\mathbf{a}$, the trace *computes* the function $f\colon D \to \mathbb{R}^m$ that sends $\mathbf{a}$ to the values of the outputs.

The condition $P(k) \subseteq \{1, \dots, k-1\}$ of Definition 2.1 says that a line uses only variables already computed. Constants live inside the operations: $4t$ and $1 - t$ are operations of one variable. In Table 3.1, $n = 2$, $N = 7$, the output is v_7, and the parent sets are $P(3) = \{1\}$, $P(4) = \{2, 3\}$, $P(5) = \{1, 2\}$, $P(6) = \{5\}$ and $P(7) = \{4, 6\}$. Each line names one operation and at most two earlier variables, so a trace is a program in *three-address form*: every instruction has at most two operands and one result. A trace is also called a *Wengert list*, after the program of Wengert (1964) that evaluated derivatives this way.

As in Definition 2.1, draw an arrow $i \to k$ whenever v_i is an argument of line k. Figure 3.1 shows the result for Table 3.1.

The figure keeps what the computation needs, namely which values feed which operations, and drops the numbering. The numbering of Table 3.1 is one choice among several: v_5 could be computed before v_3, since it uses only the inputs, and the value of f would not change. A program that records a computation as it runs need not number its variables either: it can store, with each result, links to the arguments it was computed from, as the engines described in Section 4.8 do. So we want a description of the figure that does not depend on a numbering. Its boxes are called *nodes* and its arrows *edges*, and the next definition makes these terms precise.

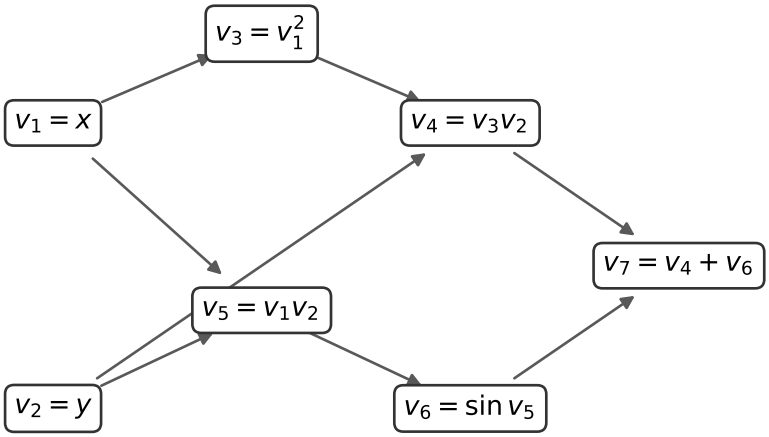

Figure 3.1.: The computational graph of the trace in Table 3.1. Each arrow runs from an argument to the line that uses it.

Definition 3.2 (Directed acyclic graph, computational graph)**.** A *directed graph* is a finite set of *nodes* together with a set of *edges*, each an ordered pair (i, k) of distinct nodes, drawn as an arrow $i \to k$. Then i is a *parent* of k, and k a *child* of i. A *path* from j to k is a sequence of edges $j = i_0 \to i_1 \to ... \to i_r = k$ with $r \geq 0$, where $r = 0$ gives the *empty path* from k to itself, as in Definition 2.1. A *cycle* is a path with $r \geq 1$ from a node to itself. A graph with no cycle is *acyclic*. A directed graph that is acyclic is called a *directed acyclic graph*, or *DAG*.

A *computational graph* with inputs $x_1, ..., x_n$ is a DAG in which n nodes, labelled $x_1, ..., x_n$, have no parents, every other node k is labelled with an elementary operation φ_k whose arguments are the values at the parents of k (taken in a fixed order, which matters for operations such as $t_1 - t_2$), and some nodes are marked as outputs.

In Figure 3.1 the nodes are the seven variables. Node 4 has parents 2 and 3, node 1 has children 3 and 5, and there are two paths from node 1 to node 7, namely $1 \to 3 \to 4 \to 7$ and $1 \to 5 \to 6 \to 7$. Every edge goes from a smaller number to a larger one. The graph of any trace has this property, because an edge $i \to k$ needs $i < k$. So the numbers strictly increase along every non-empty path, no such path returns to its start, and the graph of a trace is a computational graph.

3.2. Topological orders

Going the other way, from a computational graph to a trace, means numbering the nodes so that every node comes after its parents. The graph of Figure 3.1 allows several such numberings. Its nodes could be computed in the order $v_1, v_2, v_5, v_3, v_6, v_4, v_7$, for example, since v_5 needs only the inputs and v_6 needs only v_5. The order $v_1, v_2, v_4, v_3, ...$ fails, because v_4 uses v_3, which would not yet exist.

Definition 3.3 (Topological order)**.** A *topological order* of a directed graph with N nodes is a numbering of the nodes by $1, \dots, N$ such that $i < k$ for every edge $i \to k$.

A graph with a cycle $k_0 \to k_1 \to \dots \to k_r = k_0$ has no topological order, since the numbers would satisfy $k_0 < k_1 < \cdots < k_r = k_0$. The converse is the main result of this section. Its proof needs one fact about DAGs.

Lemma 3.1. *Every DAG with at least one node has a node with no parents.*

Proof. Suppose that every node has a parent, and let the graph have N nodes. Start at any node u_0, let u_1 be a parent of u_0, u_2 a parent of u_1, and so on, up to u_N. These are $N+1$ nodes from a set of N, so by the pigeonhole principle $u_i = u_j$ for some $i < j$. Then $u_j \to u_{j-1} \to \dots \to u_i = u_j$ is a cycle, which is impossible. □

Theorem 3.1 (Topological orders exist)**.** *Every DAG with at least one node has a topological order, and it has one in which all the nodes with no parents come first.*

Proof. By induction on the number of nodes N. A graph with one node has no edges, and numbering the node 1 is a topological order.

Let $N \geq 2$ and suppose the result for graphs with $N - 1$ nodes. Let S be the set of nodes without parents, which is not empty by Lemma 3.1, and choose s in S. Remove s and its edges. The remaining graph has $N - 1$ nodes and no cycle, since a cycle in it would be a cycle in the whole graph. By the induction hypothesis it has a topological order in which its nodes without parents come first. These include the nodes of S other than s, and possibly children of s that had no other parent. No edge ends at a node without parents, so rearranging this first block among itself keeps the order topological, and we may put the nodes of S other than s at its start. Number s as 1 and add 1 to every other number. An edge either

leaves s, and then goes from 1 to a larger number, or joins two remaining nodes, and then goes forward because the order of the remaining graph is topological. The nodes of S receive the numbers $1, \dots, |S|$. □

The proof is an algorithm: repeatedly remove a node with no parents and give it the next number. Kahn (1962) described this method of *topological sorting*. Chapter 4 gives a second algorithm, based on depth-first search. The file `lean/AutodiffBook/Ch03.lean` checks the theorem by machine.

A topological order that numbers the inputs first turns a computational graph into an evaluation trace: line k applies φ_k to the values at the parents of node k, and those parents have smaller numbers. The graph of Figure 3.1 has 12 topological orders that number the inputs first (Exercise 3.3 counts them). The value at each node does not depend on which one we use.

Corollary 3.1 (Evaluation is well defined)**.** *Let a computational graph be numbered by two topological orders, each with the inputs first, and evaluate the two resulting traces at the same input* $\mathbf{a}$*. Every node receives the same value in both, and one trace is defined at* $\mathbf{a}$ *exactly when the other is.*

Proof. List the nodes in the first order and use strong induction along this list. An input node gets its component of $\mathbf{a}$ in both traces. An operation node k gets φ_k applied to the values at its parents in both traces. Its parents come before k in the first order, so by the induction hypothesis their values agree, and so does the value at k, or the fact that it is undefined. □

From now on, a function is given by a trace $v_1, \dots, v_N$ in a topological order, with inputs $v_1, \dots, v_n$.

3.3. Forward mode

In Table 3.1, give x an increment h and hold y fixed. Every variable that depends on x changes, and for small h each changes by about a fixed multiple of h, its rate of change. These rates can be found line by line, in the same order as the values, using the partial derivatives of each operation.

- $v_1 = x$ changes at rate 1, and $v_2 = y$ at rate 0.
- $v_3 = v_1^2$ has derivative $2v_1 = 4$ with respect to v_1, so it changes at rate $4 \cdot 1 = 4$.
- $v_4 = v_3 v_2$ changes at rate $v_2 \cdot 4 + v_3 \cdot 0 = 2$, by the product rule (Theorem 2.3).
- $v_5 = v_1 v_2$ changes at rate $v_2 \cdot 1 + v_1 \cdot 0 = 0.5$.
- $v_6 = \sin v_5$ changes at rate $\cos(v_5) \cdot 0.5 = 0.5 \cos 1$.
- $v_7 = v_4 + v_6$ changes at rate $2 + 0.5 \cos 1 \approx 2.270151$.

The formula of Example 1.2 gives $\partial f/\partial x = 2xy + y\cos(xy)$, which at $(2, 0.5)$ is $2 + 0.5\cos 1$, the rate found in the last line. Each rate was the sum, over the arguments of the line, of the rate of the argument times the local partial derivative of the operation. That recipe is forward mode.

For each k and each $\mathbf{x}$ in the set D where the trace is defined, write $F_k(\mathbf{x})$ for the value of v_k, as in Definition 2.1. So $F_j(\mathbf{x}) = x_j$ for $j \le n$, and the outputs are the components of f. As in Theorem 2.2, write $d_{ki} = \partial \varphi_k / \partial v_i$ for the local partial derivative of line k with respect to its argument v_i, evaluated at the values of the arguments of line k.

Definition 3.4 (Seed and tangents)**.** Let a trace be defined at $\mathbf{a}$, and let $\mathbf{u} \in \mathbb{R}^n$, called the *seed*. The *tangents* $\dot{v}_1, \dots, \dot{v}_N$ are the numbers given by $\dot{v}_j = u_j$ for $j \le n$ and

$$\dot{v}_k = \sum_{i \in P(k)} d_{ki}\, \dot{v}_i \qquad (k = n+1, \dots, N), \tag{3.1}$$

with the local partial derivatives evaluated at $\mathbf{x} = \mathbf{a}$. The *forward pass* computes each value v_k and then its tangent $\dot{v}_k$, for $k = 1, \dots, N$.

In words, Equation 3.1 says that the tangent of a line is the sum, over its arguments, of the argument's tangent times the local partial derivative. For an operation of one variable it is the one-variable chain rule: if $v_k = \sin v_i$ then $\dot{v}_k = \cos(v_i)\,\dot{v}_i$. For a product $v_k = v_i v_l$ it is the product rule, $\dot{v}_k = v_l \dot{v}_i + v_i \dot{v}_l$. In the example above the seed was $\mathbf{u} = \mathbf{e}_1$, and $\dot{v}_k$ was the rate of change of v_k.

The seed says in which direction the inputs move. Along the line $\mathbf{x} = \mathbf{a} + s\mathbf{u}$, each variable becomes a function $s \mapsto F_k(\mathbf{a} + s\mathbf{u})$ of one real variable s, and Theorem 3.2 below, with Theorem 1.7, shows that $\dot{v}_k$ is its derivative at $s = 0$: the slope of the tangent line to its graph, which is where the name comes from. The dot follows Newton's notation for a derivative with respect to a single parameter. With seed $\mathbf{u} = (0.3, -1.7)$ in Table 3.1, for instance, both inputs move at once, and $\dot{v}_7 = 0.3\,(2 + 0.5\cos 1) - 1.7\,(4 + 2\cos 1)$ is the rate of change of f along that line.

Theorem 2.2 assumes that the inputs range over an open set on which every line is defined. The set D is such a set.

Lemma 3.2. *Let $D \subseteq \mathbb{R}^n$ be the set of points at which an evaluation trace with inputs $x_1, \dots, x_n$ is defined. Then D is open, and each F_k is continuous on D.*

Proof. Let D_k be the set of $\mathbf{x}$ at which lines $1, \dots, k$ are defined, so $D_n = \mathbb{R}^n$ and $D_N = D$. We show by induction on k that D_k is open and $F_1, \dots, F_k$ are continuous on it. The case $k = n$ holds because each coordinate function is continuous. For $k > n$, let $G(\mathbf{x}) = (F_i(\mathbf{x}))_{i \in P(k)}$ and let U_k be the open domain of φ_k. Then D_k is the set of $\mathbf{x}$ in D_{k-1} with $G(\mathbf{x}) \in U_k$. Take such an $\mathbf{x}$. There is a disc of radius $\rho > 0$ around $G(\mathbf{x})$ inside U_k, and since G is continuous at $\mathbf{x}$ there is a $\delta > 0$ such that every $\mathbf{x}'$ in D_{k-1} within δ of $\mathbf{x}$ has $G(\mathbf{x}')$ within ρ of $G(\mathbf{x})$. Shrinking δ so that the disc of radius δ

around $\mathbf{x}$ lies in the open set D_{k-1}, that whole disc lies in D_k. So D_k is open, and $F_k = \varphi_k \circ G$ is continuous on it as a composition of continuous functions. □

Theorem 3.2 (Forward mode computes Jacobian–vector products)**.** *Let an evaluation trace with inputs $x_1, \dots, x_n$ compute $f\colon D \to \mathbb{R}^m$, let $\mathbf{a} \in D$, and let $\mathbf{u} \in \mathbb{R}^n$. Then every F_k is differentiable at $\mathbf{a}$, and the tangents of Definition 3.4 with seed $\mathbf{u}$ satisfy*

$$\dot{v}_k = \nabla F_k(\mathbf{a}) \cdot \mathbf{u} \qquad (k = 1, \dots, N).$$

In particular f is differentiable at $\mathbf{a}$, and the tangents of the outputs $y_1, \dots, y_m$ are the components of the Jacobian–vector product *$J_f(\mathbf{a})\,\mathbf{u}$.*

Proof. By Lemma 3.2 the inputs range over an open set on which every line is defined, and every elementary operation is differentiable on its domain. So Theorem 2.2 applies: every F_k is differentiable at $\mathbf{a}$, and $\partial F_k/\partial x_j(\mathbf{a})$ is the sum over paths from j to k of the products of local partial derivatives. For $k > n$, every path from an input j to k is non-empty, and grouping the paths by their last arrow $i \to k$ gives

$$\frac{\partial F_k}{\partial x_j}(\mathbf{a}) = \sum_{i \in P(k)} d_{ki}\, \frac{\partial F_i}{\partial x_j}(\mathbf{a}),$$

which is Equation 2.5. Multiplying by u_j and adding over j,

$$\nabla F_k(\mathbf{a}) \cdot \mathbf{u} = \sum_{i \in P(k)} d_{ki}\, \nabla F_i(\mathbf{a}) \cdot \mathbf{u}.$$

So the numbers $\nabla F_k(\mathbf{a}) \cdot \mathbf{u}$ satisfy the recurrence Equation 3.1. They also agree with the tangents on the inputs, since $\nabla F_j(\mathbf{a}) = \mathbf{e}_j$ and $\mathbf{e}_j \cdot \mathbf{u} = u_j$ for $j \le n$. The recurrence determines $\dot{v}_k$ from the earlier values, so by induction on k, $\dot{v}_k = \nabla F_k(\mathbf{a}) \cdot \mathbf{u}$ for every k.

The components of f are the functions F_k of the output variables. Each is differentiable at $\mathbf{a}$, so f is (Lemma 2.1), and component l of $J_f(\mathbf{a})\mathbf{u}$ is $\nabla F_k(\mathbf{a}) \cdot \mathbf{u}$ for the variable v_k that is output y_l. □

So one forward pass gives the product of the Jacobian with one vector, without forming the Jacobian. With $\mathbf{u} = \mathbf{e}_j$ it gives column j of $J_f(\mathbf{a})$, the partial derivatives of every output with respect to x_j. With a unit vector it gives a directional derivative (Theorem 1.7).

Example 3.1. Run the trace of Table 3.1 at $(2, 0.5)$ with the seeds $\mathbf{e}_1$ and $\mathbf{e}_2$. Write $c = \cos 1 \approx 0.540302$.

variable	tangent rule	$\dot{v}$ for $\mathbf{e}_1$	$\dot{v}$ for $\mathbf{e}_2$
v_1	seed	1	0
v_2	seed	0	1
v_3	$2v_1\dot{v}_1$	4	0
v_4	$v_2\dot{v}_3 + v_3\dot{v}_2$	2	4
v_5	$v_2\dot{v}_1 + v_1\dot{v}_2$	0.5	2
v_6	$\cos(v_5)\,\dot{v}_5$	$0.5c$	$2c$
v_7	$\dot{v}_4 + \dot{v}_6$	$2 + 0.5c$	$4 + 2c$

The first column of tangents repeats the computation that opened this section. The last row gives $\nabla f(2, 0.5) = (2 + 0.5\cos 1,\ 4 + 2\cos 1) \approx (2.270151, 5.080605)$, and the formula $\partial f/\partial y = x^2 + x\cos(xy)$ of Example 1.2 gives the same second component at $(2, 0.5)$.

Each tangent line costs a few arithmetic operations more than the value line beside it: the partial derivatives of the operation, then one multiplication and one addition for each argument. So a forward pass costs a constant multiple of evaluating f, and Chapter 4 makes the constant precise. A gradient of $f\colon \mathbb{R}^n \to \mathbb{R}$ needs n passes, one for each $\mathbf{e}_j$, and costs about n times as much as f. For the language model of Chapter 11, $n = 4192$.

3.4. Dual numbers

Each line of a forward pass handles a pair: a value and its tangent. Pairs of this kind form a number system of their own, in which the rules for tangents become ordinary arithmetic. To see it, take $p(t) = t^3 - 2t$ and evaluate it at $2 + \varepsilon$, where ε is a formal symbol, an increment too small for its square to matter. Expanding and then discarding every term with ε^2 or a higher power,

$$p(2+\varepsilon) = (8 + 12\varepsilon + 6\varepsilon^2 + \varepsilon^3) - (4 + 2\varepsilon) \longrightarrow 4 + 10\varepsilon.$$

The number without ε is $p(2) = 4$, and the coefficient of ε is $p'(2) = 3 \cdot 2^2 - 2 = 10$. Discarding ε^2 keeps exactly the first-order part of the increment, which is the derivative. Making "discard ε^2" a rule of arithmetic gives the dual numbers.

Definition 3.5 (Dual numbers)**.** A *dual number* is an expression $a + b\varepsilon$ with a, b real. Its *real part* is a and its *ε-part* is b, and two dual numbers are equal when both parts are equal. Dual numbers are added and multiplied as polynomials in ε, followed by the rule $\varepsilon^2 = 0$:

$$(a+b\varepsilon)+(c+d\varepsilon) = (a+c)+(b+d)\varepsilon, \qquad (a+b\varepsilon)(c+d\varepsilon) = ac+(ad+bc)\varepsilon.$$

We write $\mathbb{R}[\varepsilon]$ for the set of dual numbers, and identify a real number a with $a + 0\varepsilon$.

The ε-part of a product is the product rule. If a and b are the value and the derivative of a function p at some point, and c and d those of q, then ac and $ad + bc$ are the value and the derivative of pq.

Dual numbers have a model in 2×2 matrices. Let

$$E = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \qquad M(a + b\varepsilon) = aI + bE = \begin{pmatrix} a & b \\ 0 & a \end{pmatrix}.$$

The matrix $M(a+b\varepsilon)$ holds the real part twice on its diagonal and the ε-part in its top right corner, and $E = M(\varepsilon)$. Then $E^2 = 0$, and expanding $(aI+bE)(cI+dE)$ gives $acI + (ad+bc)E$. So M sends sums to sums and products to products, and different dual numbers go to different matrices. The associative and distributive laws of matrix arithmetic therefore hold for dual numbers. The formula for the product is symmetric in the two factors, so multiplication of dual numbers is also commutative, which matrix multiplication in general is not. In particular a polynomial $p(t) = c_0 + c_1 t + \cdots + c_d t^d$ with real coefficients has a well-defined value $p(a+b\varepsilon)$.

A dual number $a + b\varepsilon$ with $a \neq 0$ has an inverse, since

$$(a+b\varepsilon)\Big(\frac{1}{a} - \frac{b}{a^2}\varepsilon\Big) = 1 + \Big(\frac{b}{a} - \frac{b}{a}\Big)\varepsilon = 1.$$

A dual number $b\varepsilon$ with zero real part has none, because $(b\varepsilon)(c+d\varepsilon) = bc\,\varepsilon$ is never 1.

Lemma 3.3. *For every integer $k \geq 1$ and all real a, b, $(a+b\varepsilon)^k = a^k + ka^{k-1}b\,\varepsilon$.*

Proof. For $k = 1$ both sides are $a + b\varepsilon$. If the formula holds for k, then

$$(a+b\varepsilon)^{k+1} = (a^k + ka^{k-1}b\varepsilon)(a+b\varepsilon) = a^{k+1} + (a^k b + ka^k b)\varepsilon = a^{k+1} + (k+1)a^k b\,\varepsilon.$$

□

Part (a) of the next theorem says that the opening example is no accident: for polynomials, dual arithmetic computes the value and the derivative. For any other operation, such as sin or exp, the value at a dual number is not yet defined, and we *define* it by the same formula: for a function g of one variable, differentiable at a,

$$g(a+b\varepsilon) = g(a) + g'(a)\,b\,\varepsilon, \tag{3.2}$$

so the real part is the value of g and the ε-part is the derivative times b. For an operation φ of two variables, differentiable at (a_1, a_2),

$$\varphi(a_1 + b_1\varepsilon,\ a_2 + b_2\varepsilon) = \varphi(a_1, a_2) + \Big(\frac{\partial \varphi}{\partial t_1}\, b_1 + \frac{\partial \varphi}{\partial t_2}\, b_2\Big)\varepsilon,$$

with the partial derivatives evaluated at (a_1, a_2): the ε-part is the directional derivative of φ in the direction (b_1, b_2). For addition and multiplication this agrees with Definition 3.5, and for division it agrees with multiplying by the inverse (Exercise 3.4). Parts (b) and (c) say that the definition is consistent with composition, so that any function built from elementary operations obeys the same formula.

Theorem 3.3 (Derivatives from dual numbers)**.**

(a) For every polynomial p with real coefficients and all real a and b,

$$p(a + b\varepsilon) = p(a) + p'(a)\, b\, \varepsilon.$$

(b) Let h be differentiable at a and g differentiable at $h(a)$, both extended to dual numbers by Equation 3.2. Then $g(h(a + b\varepsilon)) = (g \circ h)(a) + (g \circ h)'(a)\, b\, \varepsilon$: extending $g \circ h$ directly gives the same dual number as extending h and g and composing.

(c) Let an evaluation trace with inputs $x_1, \dots, x_n$ compute $f\colon D \to \mathbb{R}^m$, and let $\mathbf{a} \in D$ and $\mathbf{u} \in \mathbb{R}^n$. Evaluate the trace in dual arithmetic, with each operation extended as above and inputs $x_j = a_j + u_j\varepsilon$. Then output l is $f_l(\mathbf{a}) + \big(J_f(\mathbf{a})\mathbf{u}\big)_l\, \varepsilon$. In particular, for $n = m = 1$, $f(a + b\varepsilon) = f(a) + f'(a)\, b\, \varepsilon$.

Proof.

(a) Write $p(t) = \sum_{k=0}^{d} c_k t^k$. By Lemma 3.3, and because the term $k = 0$ is the constant c_0,

$$p(a{+}b\varepsilon) = c_0 {+} \sum_{k=1}^{d} c_k(a^k {+} k a^{k-1} b\varepsilon) = \sum_{k=0}^{d} c_k a^k {+} \Big(\sum_{k=1}^{d} k c_k a^{k-1}\Big) b\, \varepsilon = p(a) {+} p'(a)\, b\, \varepsilon.$$

(b) Apply Equation 3.2 twice and then the chain rule of one-variable calculus:

$$g(h(a+b\varepsilon)) = g\big(h(a)+h'(a)b\,\varepsilon\big) = g(h(a))+g'(h(a))\,h'(a)\,b\,\varepsilon = (g\circ h)(a)+(g\circ h)'(a)\,b\,\varepsilon.$$

(c) We show by induction on k that the dual value of v_k is $v_k+\dot{v}_k\varepsilon$, where v_k and $\dot{v}_k$ are the value and the tangent of Definition 3.4 with seed $\mathbf{u}$. For $k \le n$ this is the choice of inputs. For $k > n$, the extended operation applied to the dual arguments $v_i + \dot{v}_i\varepsilon$, $i \in P(k)$, has real part $\varphi_k((v_i)_{i\in P(k)}) = v_k$ and ε-part $\sum_{i\in P(k)} d_{ki}\,\dot{v}_i$, which is $\dot{v}_k$ by Equation 3.1. The claim about the outputs is now Theorem 3.2.

□

So a program that knows nothing about derivatives computes them when it receives dual numbers instead of real ones, provided each elementary operation knows its own derivative. The file `lean/AutodiffBook/Ch03.lean` checks part (a) by machine, and also the product rule in the ε-part of a product.

Example 3.2. Take $f(t) = e^{\sin t}$ at $t = 0$, which is not a polynomial. By Equation 3.2, $\sin(0+\varepsilon) = \sin 0+\cos(0)\,\varepsilon = \varepsilon$, and then $\exp(\varepsilon) = e^0+e^0\varepsilon = 1+\varepsilon$. By Theorem 3.3 (b), $f(0) = 1$ and $f'(0) = 1$, which agrees with $f'(t) = \cos(t)\,e^{\sin t}$.

3.5. Forward mode in code

A dual number is a pair of floating-point numbers, and the rules of Definition 3.5 and Equation 3.2 take a few lines each. The class below holds the real part in `a` and the ε-part in `b`, and it tells Python what `+`, `-`, `*` and `/` mean for such pairs (programmers call this *operator overloading*). An ordinary function written with these symbols then works on dual numbers unchanged.

```
import math


class Dual:
    """The dual number a + b eps, where eps**2 = 0."""

    def __init__(self, a, b=0.0):
        self.a, self.b = a, b

    def __add__(self, other):
        other = as_dual(other)
        return Dual(self.a + other.a, self.b + other.b)

    def __mul__(self, other):
        other = as_dual(other)
        return Dual(self.a * other.a, self.a * other.b + self.b *
         ↪  other.a)

    def __neg__(self):
        return Dual(-self.a, -self.b)

    def __sub__(self, other):
        return self + (-other)

    def __truediv__(self, other):
        other = as_dual(other)
        return self * Dual(1 / other.a, -other.b / other.a**2)

    def __rsub__(self, other):
        return as_dual(other) - self

    def __rtruediv__(self, other):
        return as_dual(other) / self

    __radd__ = __add__
    __rmul__ = __mul__

    def __repr__(self):
```

```
        return f"{self.a:.6f} + {self.b:.6f} eps"


def as_dual(x):
    """Treat a real number x as x + 0 eps."""
    return x if isinstance(x, Dual) else Dual(x)


def extend(g, dg):
    """Extend g, with derivative dg, to dual numbers:
    g(a + b eps) = g(a) + g'(a) b eps."""
    def g_dual(x):
        x = as_dual(x)
        return Dual(g(x.a), dg(x.a) * x.b)
    return g_dual


exp = extend(math.exp, math.exp)
log = extend(math.log, lambda a: 1 / a)
sin = extend(math.sin, math.cos)
cos = extend(math.cos, lambda a: -math.sin(a))
```

The methods `__add__` and `__mul__` are the two formulas of Definition 3.5. Subtraction adds the negative, and division multiplies by the inverse $1/c - (d/c^2)\varepsilon$ found above, so it needs a non-zero real part in the divisor. The methods whose names begin with `__r` handle a real number on the left, as in `1 - x`, by first turning it into the dual number $a + 0\varepsilon$ with `as_dual`. The function `extend` takes a function g and its derivative g' and returns the extension Equation 3.2, and the last four lines use it for exp, log, sin and cos.

One forward pass with seed $\mathbf{u}$ gives $f(\mathbf{a})$ and $J_f(\mathbf{a})\mathbf{u}$, by Theorem 3.3 (c). For a function with one output, n passes give the gradient. Below, `jvp` builds the inputs $a_j + u_j\varepsilon$, evaluates f on them and returns the two parts of the output, and `gradient` calls it once with each seed $\mathbf{e}_j$.

```
def jvp(f, a, u):
    """Return f(a) and the directional derivative grad f(a) . u,
    by one forward pass."""
    out = f(*[Dual(ai, ui) for ai, ui in zip(a, u)])
    return out.a, out.b


def gradient(f, a):
    """Return grad f(a) by n forward passes with seeds e_1, ...,
     ↪  e_n."""
    n = len(a)
    seeds = [[1.0 if i == j else 0.0 for i in range(n)] for j in
 ↪  range(n)]
    return [jvp(f, a, e)[1] for e in seeds]
```

Applied to the function of Table 3.1, the passes reproduce the tangent table of Example 3.1.

```
def f(x, y):
    return x * x * y + sin(x * y)


print(f(Dual(2.0, 1.0), Dual(0.5, 0.0)))   # seed e_1
print(f(Dual(2.0, 0.0), Dual(0.5, 1.0)))   # seed e_2
print(gradient(f, [2.0, 0.5]))
```

```
2.841471 + 2.270151 eps
2.841471 + 5.080605 eps
[2.27015115293407, 5.0806046117362795]
```

3.6. Symbolic differentiation and expression swell

A computer algebra system differentiates formulas by the rules of calculus and returns a formula. That is exact, but the formula can be far larger

than the computation it describes, because a formula written out in full repeats every subexpression each time it is used, while a trace computes it once and refers to it by name.

Example 3.3 (Expression swell)**.** Let $g_0(x) = x$ and $g_k(x) = 4\,g_{k-1}(x)(1 - g_{k-1}(x))$ for $k \geq 1$, the kth iterate of the logistic map. A trace computes g_k with $3k$ operations, three per step:

$$u = 1 - v, \qquad w = v\,u, \qquad v_{\text{new}} = 4w.$$

Written out as a formula, g_k contains g_{k-1} twice, so the number L_k of occurrences of x satisfies $L_k = 2L_{k-1}$ with $L_0 = 1$, and $L_k = 2^k$. Differentiating by the product and constant-multiple rules, without simplifying,

$$g_k' = 4(g_{k-1}'\,(1 - g_{k-1}) + g_{k-1}\,(-g_{k-1}')),$$

which contains g_{k-1}' twice and g_{k-1} twice. The number D_k of occurrences of x in the formula for g_k' therefore satisfies the first-order recurrence

$$D_k = 2D_{k-1} + 2L_{k-1} = 2D_{k-1} + 2^k, \qquad D_0 = 0,$$

because $g_0' = 1$ contains no x. Dividing by 2^k gives $D_k/2^k = D_{k-1}/2^{k-1} + 1$, so $D_k = k\,2^k$. For $k = 10$ the formula for g_{10} contains x 1024 times and the formula for its derivative 10 240 times, while forward mode computes both with a trace of 30 operations and 30 tangent lines. Expanding g_k into powers of x does not help, since g_k is a polynomial of degree 2^k.

Computer algebra systems store formulas with shared subexpressions, which removes much of this growth, and differentiating a formula with sharing while keeping the sharing amounts to automatic differentiation. The remaining difference is that automatic differentiation never builds a formula for the derivative at all: it computes numbers, at one point.

3.7. Finite differences

The simplest way to estimate a derivative numerically is a difference quotient. For f of one variable and a small step $h > 0$, the *forward difference* and *central difference* are

$$\frac{f(a+h)-f(a)}{h} \qquad \text{and} \qquad \frac{f(a+h)-f(a-h)}{2h}.$$

If f'' is continuous on $[a, a+h]$, Taylor's theorem with the Lagrange remainder gives $f(a+h) = f(a)+f'(a)h+\frac{1}{2}f''(\xi)h^2$ for some ξ in $(a, a+h)$, so the forward difference differs from $f'(a)$ by $\frac{1}{2}f''(\xi)h$: the *truncation error* is of order h. The central difference cancels the h^2 terms and has truncation error of order h^2 (Exercise 3.8).

Making h small reduces the truncation error but not the total error. A computer stores $f(a+h)$ and $f(a)$ with a relative rounding error of about 10^{-16}, and when h is small these two numbers agree in most of their digits, so their difference keeps only a few correct ones. The *rounding error* in the quotient grows like $10^{-16}/h$. The two errors balance near $h = 10^{-8}$ for the forward difference, where the total error is about 10^{-8}, so at best about half the digits of the derivative are correct. Table 3.2 shows both effects, and Theorem A.2 in Appendix A proves the estimates.

Table 3.2.: Errors of difference quotients for the derivative of e^x at $x = 1$, computed in double precision.

h	forward difference error	central difference error
10^{-1}	1.4×10^{-1}	4.5×10^{-3}
10^{-2}	1.4×10^{-2}	4.5×10^{-5}
10^{-4}	1.4×10^{-4}	4.5×10^{-9}
10^{-6}	1.4×10^{-6}	1.6×10^{-10}
10^{-8}	6.6×10^{-9}	6.6×10^{-9}
10^{-10}	1.5×10^{-6}	6.7×10^{-7}

Table 3.2.: Errors of difference quotients for the derivative of e^x at $x = 1$, computed in double precision.

h	forward difference error	central difference error
10^{-12}	4.3×10^{-4}	2.1×10^{-4}

A forward difference for each input costs $n + 1$ evaluations of f for a gradient, about the same as n forward passes, and gives around eight correct digits instead of sixteen. Finite differences still have a use: they need no access to how f is computed, which makes them a check on a gradient computed some other way. The source of this book uses central differences to test the gradients that automatic differentiation produces.

3.8. Notes and further reading

Wengert (1964) described a program that splits a function into elementary steps and differentiates each step, which is forward mode. Griewank and Walther (2008) is the standard reference on automatic differentiation; its early chapters treat evaluation traces and forward mode, with their cost, in far more detail than here. The survey of Baydin et al. (2018) compares automatic differentiation with symbolic and numerical differentiation for readers from machine learning, and uses the logistic map of Example 3.3 as its example of expression swell. Dual numbers go back to Clifford (1871). The source-removal algorithm in the proof of Theorem 3.1 is the one of Kahn (1962).

3.9. Exercises

Exercise 3.1. Write an evaluation trace for $f(x_1, x_2, x_3) = (x_1 + x_2)(x_2 + x_3)\, e^{x_1}$, draw its computational graph, and evaluate it at $(0, 1, 2)$.

Exercise 3.2. For the trace of Exercise 3.1, run one forward pass at $(0, 1, 2)$ with seed $\mathbf{u} = (1, -1, 2)$. Check the result against $\nabla f(0, 1, 2) \cdot \mathbf{u}$ computed from the formula.

Exercise 3.3. Show that the graph of Figure 3.1 has exactly 12 topological orders that number the two inputs first. (Hint: the nodes 3 and 4 form one chain, 5 and 6 another, and node 7 comes last.)

Exercise 3.4.

(a) Show that if $c \neq 0$, $(a + b\varepsilon)/(c + d\varepsilon) = a/c + ((bc - ad)/c^2)\varepsilon$, where division means multiplication by the inverse.

(b) Deduce that for polynomials p and q with $q(a) \neq 0$, the rational function $r = p/q$ satisfies $r(a + b\varepsilon) = r(a) + r'(a)\, b\, \varepsilon$, and that this agrees with Equation 3.2.

Exercise 3.5. Extend $g(t) = \sqrt{t}$ to dual numbers with $a > 0$ by Equation 3.2. Show that $\left(\sqrt{a + b\varepsilon}\right)^2 = a + b\varepsilon$, and compute $\sqrt{4 + \varepsilon}$.

Exercise 3.6. Let $\mathbb{R}[\varepsilon]_3$ be the numbers $a + b\varepsilon + c\varepsilon^2$, multiplied as polynomials in ε with the rule $\varepsilon^3 = 0$. Show that for every polynomial p,

$$p(a + \varepsilon) = p(a) + p'(a)\, \varepsilon + \tfrac{1}{2} p''(a)\, \varepsilon^2,$$

and use it to find $p''(1)$ for $p(t) = t^4 - 3t^2$.

Exercise 3.7. Let $s_0(x) = x$ and $s_k(x) = s_{k-1}(x) \cdot s_{k-1}(x) + x$.

(a) Find the number of occurrences of x in the formula for s_k written out in full.

(b) Differentiating by the sum and product rules without simplifying gives $s_k' = s_{k-1}'\, s_{k-1} + s_{k-1}\, s_{k-1}' + 1$. Find the number of occurrences of x in the formula for s_k'.

(c) How many operations does a trace for s_k need, and how many tangent lines does forward mode add?

Exercise 3.8. Suppose f''' is continuous on $[a-h, a+h]$ and $|f'''(t)| \leq M$ there. Show that

$$\left| \frac{f(a+h) - f(a-h)}{2h} - f'(a) \right| \leq \frac{Mh^2}{6}.$$

4. Reverse mode and backpropagation

Training a model means computing the gradient of one number, the loss, with respect to every parameter, thousands of times over. Forward mode (Chapter 3) gets one directional derivative per pass, so a gradient with respect to n parameters costs n passes. This chapter runs the chain rule the other way. After one ordinary evaluation of the function, a single sweep backwards through the evaluation trace computes the derivative of the output with respect to every variable in it, inputs included. The cost of the whole gradient is then a constant multiple of the cost of evaluating the function, whatever n is.

This is *reverse mode*, and applied to a neural network it goes by the name *backpropagation*. The chapter proves that it is correct on any acyclic computational graph and bounds its cost. It then builds a small reverse-mode engine in Python that records the evaluation trace as a table, line by line, and compares it with the design of linked nodes that the program behind the model of Chapter 11 uses.

4.1. Adjoints

Until Section 4.5, an evaluation trace $v_1, \dots, v_N$ (Definition 3.1) with inputs $v_1, \dots, v_n$ computes a function $f\colon D \to \mathbb{R}$ with a single output, and the output is the last variable, $L = v_N$. We fix $\mathbf{a}$ in D and write v_k also for the value of variable k when $\mathbf{x} = \mathbf{a}$. As in Chapter 3, $d_{ki} = \partial\varphi_k/\partial v_i$ is the

local partial derivative of line k with respect to its argument v_i, evaluated at these values.

Forward mode asks how every variable responds to an increment of the inputs. Reverse mode asks the opposite question: how does the output respond to a perturbation of one variable? Take $L = ab + a$ at $a = 2$, $b = 3$, with the trace

$$v_1 = a, \quad v_2 = b, \quad v_3 = v_1 v_2, \quad v_4 = v_3 + v_1 = L,$$

whose values are $2, 3, 6, 8$. Perturb one variable at a time by a small amount t, keeping the variables before it fixed and recomputing those after it.

- Perturbing v_4 changes L by t: rate 1.
- Perturbing v_3 to $6 + t$ gives $L = (6 + t) + 2$: rate 1.
- Perturbing v_2 to $3 + t$ gives $v_3 = 2(3 + t)$ and $L = 8 + 2t$: rate 2.
- Perturbing v_1 to $2 + t$ gives $v_3 = 3(2 + t)$ and $L = 3(2 + t) + (2 + t) = 8 + 4t$: rate 4.

The variable a reaches L by two routes, through v_3 and directly, and its rate 4 is the sum $3 + 1$ of the two contributions: the local partial derivative $d_{31} = v_2 = 3$ times the rate 1 of v_3, plus $d_{41} = 1$ times the rate 1 of v_4. In general the rate for a variable is built from the rates of the later variables that use it, so the rates can be computed from the output backwards.

The obvious candidate for "the derivative of L with respect to v_1" is the partial derivative of the last line, $\partial(v_3 + v_1)/\partial v_1 = 1$. It holds v_3 fixed, and so misses the route through v_3. The right notion recomputes every later variable, and holds only the earlier ones fixed.

Definition 4.1 (Adjoint)**.** For $n \le k \le N$, let $L_k(w_1, \dots, w_k)$ be the value of v_N obtained by setting $v_i = w_i$ for $i \le k$ and then running lines $k + 1, \dots, N$ of the trace. So $L_N(w_1, \dots, w_N) = w_N$ and $L_n = f$. The *adjoint* of the variable v_k is

$$\bar{v}_k = \frac{\partial L_k}{\partial w_k}(v_1, \dots, v_k) \quad (k > n), \qquad \bar{v}_k = \frac{\partial f}{\partial x_k}(\mathbf{a}) \quad (k \le n),$$

and we also write $\bar{v}_k = \partial L/\partial v_k$.

In words, $\bar{v}_k$ is the rate at which the output changes when v_k alone is perturbed, with every earlier variable held fixed and every later variable recomputed. In the example, $L_3(w_1, w_2, w_3) = w_3 + w_1$ and $L_2 = f$, and the adjoints are $(\bar{v}_1, \bar{v}_2, \bar{v}_3, \bar{v}_4) = (4, 2, 1, 1)$. The adjoints of the inputs form the gradient $\nabla f(\mathbf{a})$, and $\bar{v}_N = 1$. Theorem 4.1 shows that each partial derivative in the definition exists.

For $n \le k < N$, lines $k+1, \dots, N$ with $v_1, \dots, v_k$ as inputs form a system of intermediate variables (Definition 2.1) with the same parent sets and the same local partial derivatives, and L_k is its last variable as a function of those inputs. This is what lets Theorem 2.2 describe the adjoints.

4.2. The reverse sweep

In the example, the adjoint of v_1 was assembled from one contribution for each line that uses v_1: line 3 contributed $d_{31}\bar{v}_3$ and line 4 contributed $d_{41}\bar{v}_4$. Each contribution needs the adjoint of a later variable. So we process the lines from the last to the first, and let line k hand its contribution $d_{ki}\bar{v}_k$ to each of its arguments v_i. Every line that uses v_k has a number larger than k, so by the time the process reaches line k, all of them have handed over their contributions, and the adjoint of v_k is complete and ready to pass on. A variable may receive contributions from several lines, so the method keeps a running total for each variable, called its *accumulator*, and adds each contribution to it. This is the reverse sweep.

Definition 4.2 (Reverse sweep)**.**

1. *Forward pass.* Evaluate the trace at $\mathbf{a}$ and store every value $v_1, \dots, v_N$.
2. Set the accumulator of v_N to 1 and every other accumulator to 0.

3. For $k = N, N-1, \dots, n+1$, and for each $i \in P(k)$, add (accumulator of v_k) $\cdot\, d_{ki}$ to the accumulator of v_i, with d_{ki} evaluated at the stored arguments of line k.
4. Return the accumulators of $v_1, \dots, v_n$.

The next example runs the sweep on $L = ab + a$, and shows that step 3 must add to the accumulators rather than overwrite them.

Example 4.1. Run the sweep on $L = ab + a$ at $a = 2$, $b = 3$. It starts with accumulators $(0, 0, 0, 1)$.

- Line 4, $v_4 = v_3 + v_1$, has $d_{43} = d_{41} = 1$. It adds $1 \cdot 1$ to the accumulators of v_3 and v_1, giving $(1, 0, 1, 1)$.
- Line 3, $v_3 = v_1 v_2$, has $d_{31} = v_2 = 3$ and $d_{32} = v_1 = 2$. It adds $1 \cdot 3$ to the accumulator of v_1 and $1 \cdot 2$ to that of v_2, giving $(4, 2, 1, 1)$.

The final accumulators are the adjoints found by perturbation above, and $\nabla L(2, 3) = (4, 2)$ agrees with $\partial L/\partial a = b + 1$ and $\partial L/\partial b = a$. A sweep that overwrote the accumulator of v_1 instead of adding to it would end with 3, the contribution through the product alone.

Theorem 4.1 (Reverse accumulation computes the gradient)**.** *Let an evaluation trace $v_1, \dots, v_N$ with inputs $v_1, \dots, v_n$ compute $f \colon D \to \mathbb{R}$ with output v_N, and let $\mathbf{a} \in D$. Then every function L_k of Definition 4.1 is differentiable at $(v_1, \dots, v_k)$, so every adjoint exists, and:*

(a) each adjoint $\bar{v}_k$ is the sum, over all paths from k to N, of the product of the local partial derivatives along the path;

(b) the adjoints satisfy $\bar{v}_N = 1$ and

$$\bar{v}_i = \sum_{k \,:\, i \in P(k)} \bar{v}_k \, d_{ki} \qquad (i = 1, \dots, N-1); \tag{4.1}$$

(c) *when the reverse sweep of Definition 4.2 processes line k, the accumulator of v_k holds $\bar{v}_k$, and when the sweep finishes the accumulator of every variable holds its adjoint. In particular the sweep returns $\nabla f(\mathbf{a})$.*

In words, Equation 4.1 says that the adjoint of a variable is the sum, over the lines that use it, of the adjoint of that line's result times the local partial derivative.

Proof.

(a) For $k = N$, $L_N(\mathbf{w}) = w_N$ is linear, $\bar{v}_N = 1$, and the only path from N to N is the empty path, with product 1. For $k < N$, let $m = \max(k, n)$, so $m < N$. Lines $m+1, \dots, N$ with $v_1, \dots, v_m$ as inputs form a system of intermediate variables, and L_m is its last variable as a function of the inputs. The argument of Lemma 3.2 shows that the set on which L_m is defined is open, and each elementary operation is differentiable at its arguments. By Theorem 2.2, L_m is differentiable at $(v_1, \dots, v_m)$, and $\partial L_m / \partial w_k$ is the sum over paths from k to N of the products of the local partial derivatives. This partial derivative is $\bar{v}_k$ in both cases of Definition 4.1.

(b) The only path from N to N is the empty path, with product 1, so $\bar{v}_N = 1$. For $i < N$, every path from i to N starts with an arrow $i \to k$, which means $i \in P(k)$, followed by a path from k to N. Grouping the paths by their first arrow and applying (a) to each k, the sum over paths from i is $\sum_{k : i \in P(k)} d_{ki}\, \bar{v}_k$.

(c) Line k changes only the accumulators of its arguments, whose numbers are less than k. So once the sweep has reached line k, no later step changes the accumulator of v_k, and its value from then on is its starting value plus one contribution from each line k' with $k \in P(k')$: the accumulator of $v_{k'}$ when line k' was processed, times $d_{k'k}$. We show by induction downwards from N that this value is $\bar{v}_k$. For

$k = N$ no line uses v_N, and the starting value is $1 = \bar{v}_N$. For $k < N$ the starting value is 0, and every k' with $k \in P(k')$ is larger than k, so by the induction hypothesis the accumulator of $v_{k'}$ held $\bar{v}_{k'}$ when line k' was processed. So the accumulator of v_k holds $\sum_{k':k\in P(k')} \bar{v}_{k'}\, d_{k'k} = \bar{v}_k$, by Equation 4.1. For an input $k \le n$ this is the value when the sweep finishes, which is $\partial f/\partial x_k(\mathbf{a})$.

□

The theorem applies to any acyclic computational graph with one output. By Theorem 3.1 the graph has a topological order with the inputs first. Nodes from which no path leads to the output do not affect it; remove them, and the output has no children left, so some topological order numbers it last. (The removed inputs have adjoint 0, which is what the sweep's starting accumulators say.)

In Example 4.1 there are two paths from a to L: $1 \to 3 \to 4$ with product $d_{31}d_{43} = 3 \cdot 1$, and $1 \to 4$ with product $d_{41} = 1$, which add to $\bar{v}_1 = 4$. The number of paths can grow exponentially with the size of the graph (Exercise 2.4, Exercise 4.8), while the sweep handles each arrow once. It computes the path sums of part (a) by sharing work: the sum over all paths from v_k to the output is computed once, as $\bar{v}_k$, and reused by every argument of line k.

4.3. Forward and reverse mode compute the same numbers

Forward mode with seed $\mathbf{e}_i$ and reverse mode both produce $\partial f/\partial x_i(\mathbf{a})$, by recurrences that run in opposite directions. In Example 4.1, forward mode with seed $\mathbf{e}_1$ gives the tangents $\dot{v}_1 = 1$, $\dot{v}_2 = 0$, $\dot{v}_3 = d_{31}\dot{v}_1 + d_{32}\dot{v}_2 = 3$ and $\dot{v}_4 = d_{43}\dot{v}_3 + d_{41}\dot{v}_1 = 4$, while reverse mode gives $\bar{v}_1 = 4$. One identity

explains the agreement. Collect the local partial derivatives in the $N \times N$ matrix C whose entry in row i and column k is

$$c_{ik} = d_{ki} \text{ if } i \in P(k), \qquad c_{ik} = 0 \text{ otherwise.}$$

Column k of C holds the local partial derivatives of line k with respect to its arguments, and row i holds the local partial derivatives of the lines that use v_i. Since $i \in P(k)$ only when $i < k$, the matrix is strictly upper triangular: every entry on or below the diagonal is 0. For Example 4.1,

$$C = \begin{pmatrix} 0 & 0 & 3 & 1 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Column 3 holds $d_{31} = 3$ and $d_{32} = 2$, the local partial derivatives of $v_3 = v_1 v_2$, and column 4 holds $d_{41} = 1$ and $d_{43} = 1$, those of $v_4 = v_3 + v_1$. Row 1 says that v_1 is used by lines 3 and 4. Columns 1 and 2 are zero because the inputs have no arguments, and row 4 is zero because no line uses the output.

The tangent recurrence Equation 3.1 with seed $\mathbf{u}$ says $\dot{\mathbf{v}} = \mathbf{s} + C^\top \dot{\mathbf{v}}$ with $\mathbf{s} = (u_1, \dots, u_n, 0, \dots, 0)$, because entry k of $C^\top \dot{\mathbf{v}}$ is $\sum_{i \in P(k)} d_{ki} \dot{v}_i$ and the inputs have no arguments. The adjoint equations Equation 4.1 say $\bar{\mathbf{v}} = \mathbf{e}_N + C\bar{\mathbf{v}}$, because entry i of $C\bar{\mathbf{v}}$ is $\sum_{k : i \in P(k)} d_{ki} \bar{v}_k$. So forward mode multiplies by $C^\top$, running down the columns of C, and reverse mode multiplies by C, running along its rows. In the example, $\bar{\mathbf{v}} = (4, 2, 1, 1)$ and $C\bar{\mathbf{v}} = (3 + 1,\ 2,\ 1,\ 0)$, so $\mathbf{e}_4 + C\bar{\mathbf{v}} = \bar{\mathbf{v}}$. The next theorem shows that two such equations, one with $C^\top$ and one with C, always produce matching numbers.

Theorem 4.2 (Transposition)**.** *Let C be any $N \times N$ matrix and $\mathbf{s}$, $\mathbf{w}$ any vectors in $\mathbb{R}^N$. If $\mathbf{t} \in \mathbb{R}^N$ satisfies $\mathbf{t} = \mathbf{s} + C^\top \mathbf{t}$ and $\bar{\mathbf{v}} \in \mathbb{R}^N$ satisfies $\bar{\mathbf{v}} = \mathbf{w} + C\bar{\mathbf{v}}$, then $\mathbf{w} \cdot \mathbf{t} = \bar{\mathbf{v}} \cdot \mathbf{s}$.*

Proof. Using $\mathbf{w} = \bar{\mathbf{v}} - C\bar{\mathbf{v}}$ and $(C\bar{\mathbf{v}}) \cdot \mathbf{t} = \bar{\mathbf{v}}^\top C^\top \mathbf{t} = \bar{\mathbf{v}} \cdot (C^\top \mathbf{t})$,

$$\mathbf{w} \cdot \mathbf{t} = \bar{\mathbf{v}} \cdot \mathbf{t} - \bar{\mathbf{v}} \cdot (C^\top \mathbf{t}) = \bar{\mathbf{v}} \cdot (\mathbf{t} - C^\top \mathbf{t}) = \bar{\mathbf{v}} \cdot \mathbf{s}.$$

□

With $\mathbf{w} = \mathbf{e}_N$ and the seed $\mathbf{u} = \mathbf{e}_i$, the left side is the tangent of the output, which is $\partial f/\partial x_i(\mathbf{a})$ by Theorem 3.2, and the right side is $\bar{v}_i$. This gives a second proof that the sweep returns the gradient: n forward passes and one reverse sweep fill in the same vector. The file `lean/AutodiffBook/Ch04.lean` checks the identity by machine.

4.4. The cost of a gradient

How much more does the gradient cost than the function? Count the work for Example 4.1, charging one unit for each arithmetic operation and one for each number read or set. Evaluating $L = ab + a$ costs 4 units: 2 for reading the inputs and 2 for the operations. The sweep sets 4 accumulators, at 4 units. The local partial derivatives cost 2 units: those of $v_3 + v_1$ are the constant 1, set twice, and those of $v_1 v_2$ are the stored arguments. Each of the 4 arrows needs one multiplication and one addition, 8 units in all. Forward pass and sweep together cost $4 + 4 + 2 + 8 = 18$ units, 4.5 times the cost of f. The same count for any trace gives a bound that does not depend on the number of inputs.

Definition 4.3 (Cost model)**.** Let a trace with inputs $v_1, \dots, v_n$ compute f. Give each line k a *cost* $c_k \geq 1$: reading an input costs $c_k = 1$, and for $k > n$, c_k is the cost of evaluating φ_k. The cost of evaluating f is $\mathrm{cost}(f) = c_1 + \cdots + c_N$. The *cost model* consists of three assumptions:

(i) every operation has at most r arguments;

(ii) for each $k > n$, the local partial derivatives d_{ki}, $i \in P(k)$, can be computed from the arguments and the value of line k at cost at most $\alpha\, c_k$, for a constant α;

(iii) each addition or multiplication costs 1, and so does setting a number to a constant.

For the elementary operations of Chapter 3, $r = 2$. With every arithmetic operation and every evaluation of exp, log, sin, cos or a power costing 1, we can take $\alpha = 3$: the partial derivatives of $t_1 + t_2$ and $t_1 t_2$ are constants or arguments; the derivative of e^t is the value already computed; those of $\log t$ and $\sin t$ take one operation each, and that of $\cos t$, which is $-\sin t$, takes two; that of t^c is $c\, t^c/t$, two operations; and those of t_1/t_2 are $1/t_2$ and $-(t_1/t_2)/t_2$, three operations.

Theorem 4.3 (The cheap gradient principle)**.** *Let a trace with inputs $v_1, \dots, v_n$ compute $f\colon D \to \mathbb{R}$, with line costs $c_k \geq 1$ and $\operatorname{cost}(f) = c_1 + \cdots + c_N$. Suppose every operation has at most r arguments, the local partial derivatives of line k can be computed at cost at most αc_k, and each addition, multiplication and assignment of a constant costs 1. Then the forward pass and the reverse sweep together compute $f(\mathbf{a})$ and $\nabla f(\mathbf{a})$ at cost at most*

$$(2 + \alpha + 2r)\operatorname{cost}(f),$$

whatever the number of inputs n.

Proof. Add up the steps of Definition 4.2. The forward pass costs $\operatorname{cost}(f)$. Setting the N accumulators costs N, and $N \leq \operatorname{cost}(f)$ because every $c_k \geq 1$. Computing the local partial derivatives of every line costs at most $\sum_{k>n} \alpha c_k \leq \alpha \operatorname{cost}(f)$. In step 3, each pair (k, i) with $i \in P(k)$ costs one multiplication and one addition, and there are $\sum_{k>n} |P(k)| \leq r \sum_{k>n} 1 \leq r \operatorname{cost}(f)$ such pairs. The total is at most $(1 + 1 + \alpha + 2r)\operatorname{cost}(f)$. □

With $\alpha = 3$ and $r = 2$ the bound is $9\,\mathrm{cost}(f)$, comfortably above the factor 4.5 of the example. Forward mode needs n passes for the same gradient, each costing a constant multiple of $\mathrm{cost}(f)$. The constant 9 comes from a coarse count. Griewank and Walther (2008) analyse the cost of reverse mode in detail, and Baydin et al. (2018, sec. 3.2), citing that analysis, give the constant as less than 6 and typically between 2 and 3.

4.5. Vector–Jacobian products and the shape of the Jacobian

Reverse mode works for functions with several outputs too. In Example 4.1, treat both $y_1 = v_3 = ab$ and $y_2 = v_4 = ab + a$ as outputs, so $f(a, b) = (ab,\ ab + a)$ with

$$J_f(2,3) = \begin{pmatrix} 3 & 2 \\ 4 & 2 \end{pmatrix},$$

whose rows are the gradients of the two outputs. Run the sweep with the accumulators of v_3 and v_4 starting at chosen numbers w_1 and w_2 instead of 0 and 1. Starting from $(w_1, w_2) = (0, 1)$ is the sweep of Example 4.1, which returns $(4, 2)$, the second row. Starting from $(1, 0)$, line 4 contributes nothing and line 3 returns $(3, 2)$, the first row. Starting from $(1, 1)$, line 4 raises the accumulator of v_3 to 2 and adds 1 to that of v_1, and line 3 then adds $2 \cdot 3$ and $2 \cdot 2$, returning $(7, 4)$, the sum of the rows. In each case the sweep returns $\mathbf{w}^\top J_f(2,3)$, the combination of the rows with weights w_1, w_2.

Theorem 4.4 (Reverse mode computes vector–Jacobian products)**.** *Let an evaluation trace with inputs $v_1, \dots, v_n$ compute $f\colon D \to \mathbb{R}^m$, with outputs $y_1, \dots, y_m$ at distinct variables, let $\mathbf{a} \in D$ and let $\mathbf{w} \in \mathbb{R}^m$. Run the reverse sweep of Definition 4.2 with the accumulator of output y_l starting at w_l, every other accumulator starting at 0, and every line from N down to $n + 1$ processed. Then the sweep returns $J_f(\mathbf{a})^\top \mathbf{w}$, whose transpose is the* vector–Jacobian product $\mathbf{w}^\top J_f(\mathbf{a})$.

Proof. Let $\tilde{\mathbf{w}} \in \mathbb{R}^N$ hold w_l in the position of output y_l and 0 elsewhere, and let $\bar{\mathbf{v}}$ be the final accumulators. As in the proof of Theorem 4.1 (c), the accumulator of v_k does not change after line k is processed. So the accumulator of v_i ends as its starting value plus one contribution from each line k that uses v_i, namely $\tilde{w}_i + \sum_{k:i\in P(k)} d_{ki}\bar{v}_k$, and $\bar{\mathbf{v}} = \tilde{\mathbf{w}} + C\bar{\mathbf{v}}$ with the matrix C of Section 4.3. Run forward mode with seed $\mathbf{e}_j$, giving tangents $\mathbf{t}$ with $\mathbf{t} = \mathbf{e}_j + C^\top \mathbf{t}$. By Theorem 4.2 and Theorem 3.2,

$$\bar{v}_j = \tilde{\mathbf{w}} \cdot \mathbf{t} = \sum_{l=1}^{m} w_l \left(J_f(\mathbf{a})\mathbf{e}_j\right)_l = \left(J_f(\mathbf{a})^\top \mathbf{w}\right)_j.$$

□

With $\mathbf{w} = \mathbf{e}_l$, one sweep gives row l of the Jacobian, the gradient of output l. So for $f\colon \mathbb{R}^n \to \mathbb{R}^m$, forward mode gets the Jacobian a column at a time, in n passes, and reverse mode gets it a row at a time, in m sweeps. Each pass or sweep costs a constant multiple of evaluating f, so forward mode is the cheaper choice when $n < m$, and reverse mode when $m < n$. A loss function has $m = 1$ and n in the thousands or more, which is why training uses reverse mode. A curve $\gamma\colon \mathbb{R} \to \mathbb{R}^3$ has $n = 1$, and one forward pass gives its velocity. For a product of Jacobians, choosing the order of the matrix multiplications is the same choice (Exercise 4.6).

4.6. Memory and checkpointing

Reverse mode pays for its speed in *memory*, by which we mean the number of values a method must store at the same time. Line k of the sweep needs the arguments of line k, so the forward pass must keep every value until the sweep reaches it: memory in proportion to N, the length of the computation. Forward mode, by contrast, can discard each value once the lines that use it have run.

Take a computation that is a chain of T steps, $\mathbf{s}_t = \Phi_t(\mathbf{s}_{t-1})$ for $t = 1, \dots, T$, such as the layers of a deep network, where each *state* $\mathbf{s}_t$ is a vector. Sweeping back through step t needs its argument $\mathbf{s}_{t-1}$, so the plain sweep stores all the states. With $T = 100$ there is a cheaper way. Keep only the ten states $\mathbf{s}_0, \mathbf{s}_{10}, \dots, \mathbf{s}_{90}$ during the forward pass. To sweep back through steps $100, \dots, 91$, recompute $\mathbf{s}_{91}, \dots, \mathbf{s}_{99}$ from $\mathbf{s}_{90}$; once the sweep has passed step 91, discard them, recompute $\mathbf{s}_{81}, \dots, \mathbf{s}_{89}$ from $\mathbf{s}_{80}$, and so on. At most 20 states are stored at any moment, against 100, and each Φ_t is evaluated at most twice.

Checkpointing trades recomputation for memory in this way. During the forward pass, keep only every Kth state $\mathbf{s}_0, \mathbf{s}_K, \mathbf{s}_{2K}, \dots$, about T/K of them. During the sweep, take the segments in reverse order: recompute the states of a segment from its stored first state, keep these K states, sweep back through the segment, and discard them. At any moment the method stores at most about $T/K + K$ states, and the recomputation adds at most one more forward pass. Since

$$\frac{T}{K} + K - 2\sqrt{T} = \left(\sqrt{T/K} - \sqrt{K}\right)^2 \geq 0,$$

the memory is smallest, about $2\sqrt{T}$, when $K = \sqrt{T}$. For $T = 10^6$ steps that is 2000 stored states instead of a million.

Chen et al. (2016) use this scheme for deep networks, with memory proportional to $\sqrt{n}$ for an n-layer network at the price of one extra forward pass. Applying the idea recursively inside each segment reduces memory further (Exercise 4.9). Griewank (1992) showed that checkpointing applied recursively makes both the memory and the extra computation grow only logarithmically with the length of the computation, relative to a single evaluation, and Griewank and Walther (2000) give schedules that are optimal for this trade-off.

4.7. A reverse-mode engine on a tape

The reverse sweep of Definition 4.2 reads three things from the forward pass: the order of the lines, the arguments of each line, and the local partial derivatives d_{ki} at the values of the forward pass. A program can collect all three while it evaluates the function. Each time it applies an elementary operation, it computes the result together with the local partial derivatives of the operation, and writes one line into a list: the operation, the value, and a pair (i, d_{ki}) for each argument v_i. We call such a list a *tape*. For $L = ab + a$ at $a = 2$, $b = 3$ the tape is Table 4.1.

Table 4.1.: The tape of $L = ab + a$ at $(a, b) = (2, 3)$: the evaluation trace of Section 4.1, with the local partial derivatives of each line.

line	operation	value	pairs (i, d_{ki})
v_1	a	2	
v_2	b	3	
v_3	$v_1 v_2$	6	$(1, 3)$, $(2, 2)$
v_4	$v_3 + v_1$	8	$(3, 1)$, $(1, 1)$

The first three columns are the evaluation trace as we wrote it by hand in Section 4.1. The last column holds the non-zero entries of the matrix C of Section 4.3, one column of C per line. The sweep of Example 4.1 reads only this column, from the bottom row up: row 4 adds $1 \cdot 1$ to the accumulators of v_3 and v_1, and row 3 then adds $1 \cdot 3$ to that of v_1 and $1 \cdot 2$ to that of v_2.

A tape needs no sorting before the sweep. A program can apply an operation only to values that already exist, so every argument of a line was recorded on an earlier line, and $P(k) \subseteq \{1, \dots, k-1\}$ as Definition 3.1 requires. The order of recording is therefore a topological order of the computational graph, and the sweep can take the lines in reverse order of recording. The proof of Theorem 4.1 (c) is an induction downwards over

the line numbers and uses nothing else about the order, so it applies to the tape as recorded.

The engine below rests on three decisions. The first is what a number of the computation is: a *variable*, an object that holds a line number k and the value v_k. Python lets a class say what `+`, `*` and the other operators mean for its objects (operator overloading, as in Section 3.5). The operators of a variable compute the result and the local partial derivatives, and append the line to the tape. Ordinary arithmetic on variables then writes the trace as a side effect.

The second is how the operators know which tape to write to. The program *starts* a tape before it evaluates the function and *stops* it afterwards, and only a tape that has been started and not yet stopped, the *recording* tape, receives lines. Python's `with` statement does both: `with Tape() as tape:` starts a new tape, runs the indented block below it, and stops the tape when the block ends, even if the block fails with an error. Arithmetic outside the block, such as the update of the parameters after a step of gradient descent, then stays off the tape: it is not part of the function being differentiated.

The third is how long a tape lives: each evaluation gets a new tape, which the program discards after the sweep. A tape keeps every line it receives, so its memory grows with the length of the computation, as Section 4.6 explains. A training loop that recorded every step on one tape would keep the lines of all earlier steps, and every sweep would pass through all of them. The line number held by a variable refers to the tape that recorded it, so a variable belongs to one evaluation. A model's parameters therefore live between steps as ordinary numbers, and each step records them afresh as inputs of its own tape.

```
import math
from collections import namedtuple

# Line k of the trace: a pattern for printing the operation, the
↪  value
```

```
# v_k, and one pair (i, d_ki) for each argument v_i.
Line = namedtuple("Line", "op value partials")


class Tape:
    """An evaluation trace, recorded one line at a time."""

    recording = None  # the tape that operations on variables
 ↪  write to

    def __init__(self):
        self.lines = [None]  # line k is self.lines[k]; entry 0 is
         ↪  unused

    def __enter__(self):
        if Tape.recording is not None:
            raise RuntimeError("another tape is recording")
        Tape.recording = self
        return self

    def __exit__(self, *exc_info):
        Tape.recording = None

    def input(self, value, name="input"):
        """Record an input line holding value, and return its
         ↪  variable."""
        return record(name, float(value))


class Var:
    """Variable v_k of the trace being recorded."""

    def __init__(self, k, value):
        self.k = k          # its line number on the tape
        self.value = value  # the number v_k

    def __pow__(self, c):
        """v ** c for a constant exponent c."""
```

```
        t = self.value
        return record(f"{{}} ** {c:g}", t**c, (self.k, c * t ** (c
         ↪  - 1)))

    def __neg__(self):
        return -1.0 * self


def record(op, value, *partials):
    """Append a line to the recording tape, and return its
     ↪  variable."""
    tape = Tape.recording
    if tape is None:
        raise RuntimeError("no tape is recording; use `with
         ↪  Tape():`")
    tape.lines.append(Line(op, value, partials))
    return Var(len(tape.lines) - 1, value)


def binary(symbol, g, d1, d2):
    """Two methods of Var for g(s, t), written s symbol t,
     ↪  partials d1, d2.

    Python calls the first for u symbol w with u a Var, and the
 ↪  second
    for c symbol u with c a number.
    """
    def var_first(u, w):
        if isinstance(w, Var):
            s, t = u.value, w.value
            return record(f"{{}} {symbol} {{}}", g(s, t),
                          (u.k, d1(s, t)), (w.k, d2(s, t)))
        s = u.value
        return record(f"{{}} {symbol} {w:g}", g(s, w), (u.k, d1(s,
         ↪  w)))

    def number_first(u, c):
        t = u.value
```

```
        return record(f"{c:g} {symbol} {{}}", g(c, t), (u.k, d2(c,
         ↪ t)))

    return var_first, number_first


Var.__add__, Var.__radd__ = binary(
    "+", lambda s, t: s + t, lambda s, t: 1.0, lambda s, t: 1.0)
Var.__sub__, Var.__rsub__ = binary(
    "-", lambda s, t: s - t, lambda s, t: 1.0, lambda s, t: -1.0)
Var.__mul__, Var.__rmul__ = binary(
    "*", lambda s, t: s * t, lambda s, t: t, lambda s, t: s)
Var.__truediv__, Var.__rtruediv__ = binary(
    "/", lambda s, t: s / t, lambda s, t: 1 / t, lambda s, t: -s /
     ↪ t**2)


def elementary(name, g, dg):
    """The operation t -> g(t), with derivative dg, on a
     ↪ variable."""
    def apply(u):
        return record(f"{name} {{}}", g(u.value), (u.k,
         ↪ dg(u.value)))
    return apply


exp = elementary("exp", math.exp, math.exp)
log = elementary("log", math.log, lambda t: 1 / t)
sin = elementary("sin", math.sin, math.cos)
cos = elementary("cos", math.cos, lambda t: -math.sin(t))
relu = elementary("relu", lambda t: max(t, 0.0),
                  lambda t: 1.0 if t > 0 else 0.0)
```

The class `Tape` holds the list `lines`, with line k in entry k. Python numbers the entries of a list from 0, and entry 0 stays empty so that the numbering matches the book's. The class attribute `Tape.recording` names the recording tape, or is `None` when no tape records. The methods

`__enter__` and `__exit__` are the ones that the `with` statement calls at the start and the end of its block: the first makes the new tape the recording one, refusing if another tape is already recording, and the second stops it. Each line is a `Line` with three fields: `op`, a pattern for printing the operation in which `{}` stands for each argument; `value`, the value v_k; and `partials`, the pairs (i, d_{ki}). The function `record` appends a line to the recording tape and returns a `Var` holding the new line number and the value. The method `tape.input` records an input line, which has no pairs.

The operators come from the function `binary`. It turns an operation $g(s,t)$ of two numbers and its partial derivatives $\partial g/\partial s$ and $\partial g/\partial t$ into two methods. The first handles `u + w` with `u` a variable. If `w` is also a variable, the line has two arguments. If `w` is a number, the line has one, because constants live inside the operations: $t + 2$ is an operation of one variable (Section 3.1). The second method handles `2 + u`, with a number on the left, which Python passes to the method `__radd__` of `u`. The function `elementary` does the same for an operation $g(t)$ of one variable with derivative g', as `extend` does for dual numbers in Section 3.5.

The *primitive* operations, the ones with a derivative rule of their own, are the elementary operations of Section 3.1: addition, subtraction, multiplication and division of two numbers or of a number and a constant, powers with a constant exponent, and exp, log, sin and cos. To these the engine adds the ReLU, $\max(t, 0)$. Each primitive records one line, so the tape of a formula is the trace a reader would write for it by hand, as in Table 4.1. The other operations are *derived*: negation records `-1.0 * u`, a multiplication by a constant, and a square root is written `u ** 0.5`. A derived operation needs no derivative rule, because the tape records the primitive lines it is made of and the sweep differentiates those. Division could be derived in the same way, as `u * w ** -1`, at the price of a second line for every quotient; as a primitive it keeps the tape equal to the trace of Chapter 3.

The ReLU has no derivative at 0, and the engine uses 0 there, a choice Chapter 7 discusses. At points where no ReLU receives exactly 0, each

ReLU agrees with 0 or with t on an interval around its argument, so it is differentiable there and Theorem 4.1 applies.

```
def backward(tape, out):
    """Return bar, where bar[k] is the adjoint of line k for
     ↪ output out."""
    N = len(tape.lines) - 1
    bar = [0.0] * (N + 1)
    bar[out] = 1.0
    for k in range(N, 0, -1):
        for i, d in tape.lines[k].partials:
            bar[i] += bar[k] * d
    return bar


def gradient(f, a):
    """Return f(a) and the gradient of f at a, by one tape and one
     ↪ sweep."""
    with Tape() as tape:
        x = [tape.input(aj) for aj in a]
        out = f(*x)
    bar = backward(tape, out.k)
    return out.value, [bar[xj.k] for xj in x]


def print_tape(tape):
    """Print the tape as a table of lines, values and pairs (i,
     ↪ d_ki)."""
    print(f"{'line':<6}{'operation':<14}{'value':>9}   pairs (i,
     ↪ d_ki)")
    for k in range(1, len(tape.lines)):
        op, value, partials = tape.lines[k]
        formula = op.format(*(f"v{i}" for i, _ in partials))
        pairs = ", ".join(f"({i}, {d:.6g})" for i, d in partials)
        print(f"v{k:<5}{formula:<14}{value:>9.6g}
         ↪ {pairs}".rstrip())
```

The function `backward` takes a tape and the line number `out` of the output.

It makes a list `bar` of adjoints indexed by line number, with every entry 0 except `bar[out]`, which is 1. It then runs through the lines from the last, N, down to 1, and for each pair (i, d) of line k adds `bar[k] * d` to `bar[i]`. This is step 3 of Definition 4.2, with `bar` as the list of accumulators. Input lines have no pairs and add nothing. The function returns the whole list, so the adjoint of every line is available, inputs included.

The function `gradient` runs the whole cycle for a function of n numbers: it starts a tape, records the inputs, evaluates f on them, stops the tape, sweeps, and returns $f(\mathbf{a})$ together with the adjoints of the inputs. It plays the part of the forward-mode `gradient` of Section 3.5, with one sweep in place of n passes. The function `print_tape` prints one row for each line: the line, the operation with each argument written as v_i, the value, and the pairs.

Running the engine on Example 4.1 prints the tape and the adjoints:

```
with Tape() as tape:
    a = tape.input(2.0, "a")
    b = tape.input(3.0, "b")
    L = a * b + a
print_tape(tape)

bar = backward(tape, L.k)
print("adjoints:", bar[1:])
print("gradient:", (bar[a.k], bar[b.k]))
```

```
line  operation         value   pairs (i, d_ki)
v1    a                     2
v2    b                     3
v3    v1 * v2               6   (1, 3), (2, 2)
v4    v3 + v1               8   (3, 1), (1, 1)
adjoints: [4.0, 2.0, 1.0, 1.0]
gradient: (4.0, 2.0)
```

The printed tape is Table 4.1, row for row. The sweep gives the adjoints $(4, 2, 1, 1)$ of Example 4.1, and the gradient $\nabla L(2, 3) = (4, 2)$.

A product `x * x` records one line whose two pairs both name the line of `x`, each with $d = x$, and the `+=` in `backward` adds the two contributions to give $2x$. This is Equation 4.1 for the operation $t \mapsto t^2$, whose derivative is the sum of the partial derivatives of $t_1 t_2$ at $t_1 = t_2 = t$. The sweep runs over every line, including lines that the output does not depend on. Their adjoints stay 0, so their pairs add nothing (Exercise 4.3).

A small two-layer network, of the kind Chapter 7 studies, shows the engine on the kind of expression the rest of the book trains. From the inputs $x_1 = 0.5$ and $x_2 = -1.5$, a 2×2 matrix W of weights and weights c_1, c_2, it computes

$$h_i = \max(w_{i1}x_1 + w_{i2}x_2,\ 0) \quad (i = 1, 2), \qquad y = c_1 h_1 + c_2 h_2, \qquad \text{loss} = (y-1)^2,$$

the squared distance of y from a target of 1. The code records the inputs and the six weights as the first eight lines, evaluates the loss, and sweeps once for the derivative of the loss with respect to each weight.

```
with Tape() as tape:
    x = [tape.input(0.5), tape.input(-1.5)]
    W = [[tape.input(0.3), tape.input(-0.8)],
         [tape.input(1.1), tape.input(0.4)]]
    c = [tape.input(0.7), tape.input(-1.2)]
    h = [relu(W[i][0] * x[0] + W[i][1] * x[1]) for i in range(2)]
    y = c[0] * h[0] + c[1] * h[1]
    loss = (y - 1.0) ** 2
bar = backward(tape, loss.k)

print(f"loss = {loss.value:.6f}, lines on the tape:
 ↪  {len(tape.lines) - 1}")
print("dloss/dW =", [[round(bar[w.k], 6) for w in row] for row in
 ↪  W])
print("dloss/dc =", [round(bar[ci.k], 6) for ci in c])
```

```
loss = 0.003025, lines on the tape: 21
dloss/dW = [[-0.0385, 0.1155], [0.0, 0.0]]
dloss/dc = [-0.1485, 0.0]
```

For h_2 the argument of the maximum is $1.1 \cdot 0.5 + 0.4 \cdot (-1.5) = -0.05$, so $h_2 = 0$, the ReLU line has $d = 0$ and passes back adjoint 0, and the weights w_{21}, w_{22} and c_2 get zero derivative at this point. For h_1 it is 1.35, so $y = 0.7 \cdot 1.35 = 0.945$ and the loss is $0.055^2 = 0.003025$. The tape has 21 lines: 8 inputs, 4 for each h_i (two products, a sum and a ReLU), 3 for y, and 2 for the loss (a subtraction of a constant and a square).

The source of this book checks every gradient this engine produces against SymPy and against central differences, and on a chain of 1000 lines against the forward mode of Section 3.5. The engine is slow, since every scalar operation creates a Python object and a line, but its operations suffice for the model of Chapter 11: the program behind that model differentiates with the same operations, kept as linked nodes (Section 4.8) instead of a tape (Karpathy 2026b). Real systems apply the same sweep to arrays, with one line per matrix operation instead of one per number, which is the subject of Theorem 7.1.

4.8. Linked nodes and depth-first search

A tape is one way to keep the trace. Another keeps no list at all: each result holds references to its arguments and the local partial derivatives with respect to them, and nothing else records the computation. Call these objects *nodes*. Each node holds its value and links to its parents, the arguments of the operation that produced it, so the nodes and links form the computational graph of Definition 3.2. In Example 4.1 the node of L links to the nodes of v_3 and a, and the node of v_3 links to those of a and b. No table of lines exists in memory, and no node records when it was made.

To run the sweep, such an engine needs the nodes on which the output depends, listed so that every node comes after its parents; reading the list backwards then processes every node before its arguments. It must build the list from the links, starting at the output. In Example 4.1, start at

$L = v_4$. Before listing v_4, list its parents, v_3 and v_1. Before listing v_3, list its parents v_1 and v_2, which have no parents and are listed at once. So the list begins v_1, v_2, v_3. The other parent of v_4 is v_1, which is already listed, and finally v_4 is listed: the result is v_1, v_2, v_3, v_4.

Definition 4.4 (Depth-first topological sort)**.** Start with an empty list and no node marked. To *visit* a node u: if u is marked, do nothing; otherwise mark u, visit each parent of u in turn, and then append u to the list. The *depth-first order* from a node o is the list obtained by visiting o.

A node u is an *ancestor* of o if there is a path from u to o (possibly the empty path, so o is an ancestor of itself).

Theorem 4.5 (Depth-first search gives a topological order)**.** *Let o be a node of a DAG. The depth-first order from o contains every ancestor of o exactly once and no other node, and each node in it comes after all of its parents.*

Proof. A node is marked before its visit appends it, and a visit to a marked node does nothing, so each node is appended at most once. Each visit to an unmarked node marks it, so at most as many visits do any work as there are nodes, and the search ends. Visits happen only to o and, from inside the visit of a node, to that node's parents, so by induction every visited node is an ancestor of o. Conversely, if $u_0 \to u_1 \to ... \to u_r = o$ is a path, then by induction downwards from o each u_i is visited, since every visit to an unmarked node visits all its parents, and a marked node has already been visited. Every visited node is marked, and is appended when its first visit ends.

Now let p be a parent of u, with u in the list. During the first visit to u, the search visits p. If p is unmarked at that moment, the visit to p finishes, and appends p, before the visit to u appends u. If p is already marked, its first visit began earlier. If that visit has already finished, p is already in the list. Otherwise the visit to p is still in progress, so the visit to u was

started, directly or through a chain of visits, from inside it. Each visit in that chain is to a parent of the node before it, so there is a path from u to p. Together with the edge $p \to u$ it forms a cycle, which is impossible in a DAG. □

Each node is marked once and each edge is followed once, so the search does work proportional to the number of nodes plus the number of edges of the part of the graph above o. Tarjan (1972) showed how depth-first search leads to efficient graph algorithms.

The most direct translation of Definition 4.4 into Python is a function `visit` that calls itself once for each parent. But a call that has started and not yet finished occupies space until it returns, and Python allows only about a thousand such calls at once, so a chain of a thousand operations would exceed the limit. The function `topological_order` below therefore keeps its own list of the visits in progress. It adds and removes entries only at the end of that list, which programmers call a *stack*. The last entry is the visit being carried out, together with the parents it has yet to visit. At each step that visit takes its next parent and, if the parent is unmarked, marks it and starts a visit to it by adding an entry. When a visit has no parents left, its entry is removed and its node is appended to the order, as Definition 4.4 prescribes. The set `visited` holds the marked nodes. The function works on any objects that have an attribute `parents`; the small class `Node`, which holds a name and the parents and nothing else, is enough to try it on the graph of Example 4.1.

```
def topological_order(root):
    """Every node that root depends on, each after all of its
    ↪ parents."""
    order, visited = [], {root}
    stack = [(root, iter(root.parents))]   # the visits in
↪ progress
    while stack:
        node, parents = stack[-1]
        parent = next(parents, None)
```

```
        if parent is None:                  # every parent of node
        ↪  is done
            stack.pop()
            order.append(node)
        elif parent not in visited:         # start a visit to
        ↪  parent
            visited.add(parent)
            stack.append((parent, iter(parent.parents)))
    return order


class Node:
    """A node of a computational graph: a name and the parents."""

    def __init__(self, name, *parents):
        self.name, self.parents = name, parents
```

```
v1, v2 = Node("v1"), Node("v2")
v3 = Node("v3", v1, v2)
v4 = Node("v4", v3, v1)
print([u.name for u in topological_order(v4)])
```

```
['v1', 'v2', 'v3', 'v4']
```

With this list in hand, the sweep is Definition 4.2 with the positions in the list in place of line numbers. The engine takes the nodes from the last to the first, and each node adds its adjoint times each stored local partial derivative to the accumulator of the corresponding parent. The accumulators can be kept in the nodes themselves or in a table indexed by node.

This is the design of systems that build a graph of linked nodes as the program runs. PyTorch records with each result the function that produced it, an entry point into a graph of such functions that it builds afresh at every evaluation (PyTorch Contributors 2026). Two small examples in plain Python are micrograd (Karpathy 2020) and the program behind the

model of Chapter 11. Each wraps every number in an object that holds its accumulator and links to its arguments, and each sorts the nodes by the recursive form of Definition 4.4 before the sweep.

The two designs differ in four ways, summarised in Table 4.2.

Table 4.2.: Two ways to record the trace for reverse mode.

	tape	linked nodes
order for the sweep	the order of recording, free	rebuilt by depth-first search at every sweep
lines swept	every line on the tape	only the ancestors of the output
lifetime of the record	from start to discard, set by the program	as long as some node refers to it
several computations	one tape per computation, one recording at a time	separate graphs side by side, sharing nodes freely

The tape gets its order for nothing, but sweeps every line, including lines that the output does not depend on. Depth-first search costs work in proportion to the size of the graph above the output, about as much as the sweep itself, and in return skips every node that does not lead to the output. A tape lives until the program discards it, which is why the program must start and stop it; linked nodes live as long as something refers to them, so a node that leads nowhere disappears as soon as the program drops it, and the rest of the graph goes when the program drops the output. And because nodes carry their own links, two computations can build their graphs at the same time and share nodes, such as the parameters of a model, while the tape of this chapter records one computation at a time and keeps parameters as ordinary numbers between evaluations.

4.9. History

Wengert (1964) described forward mode as a program in 1964. For reverse mode, Linnainmaa (1976) expanded the rounding error accumulated by an algorithm in terms of the local rounding errors of its steps, a calculation of first-order sensitivities of the final result to each intermediate quantity; Baydin et al. (2018, sec. 3.3) note that this work is often cited as the first published description of reverse mode. Rumelhart et al. (1986) applied the method, under the name back-propagation, to training layered neural networks, and their paper brought it to wide attention. Researchers in several fields found the idea independently, and Baydin et al. (2018, sec. 3.3) trace its history through optimal control and the automatic differentiation community.

4.10. Notes and further reading

Griewank and Walther (2008) develop reverse mode in full, including its cost and the checkpointing methods that reduce its memory use. Baydin et al. (2018) survey automatic differentiation for machine learning, with worked forward and reverse sweeps on a small example and a review of implementations. Topological sorting goes back to Kahn (1962), and the method of Theorem 4.5 uses the depth-first search studied by Tarjan (1972).

4.11. Exercises

Exercise 4.1. Run the reverse sweep by hand on the trace of Table 3.1 at $(2, 0.5)$, listing the adjoint of every variable, and compare the result with Example 3.1.

Exercise 4.2. On a tape whose first line is an input `x`, the statement `y = x * x` records line 2 with the pairs $(1, x)$ and $(1, x)$.

(a) Show that `backward(tape, y.k)` sets `bar[1]` to $2x$.

(b) What would it set `bar[1]` to if `backward` used `=` instead of `+=`? What would it give for Example 4.1?

Exercise 4.3. Record, with $a = 2$ and $b = 3$,

```
with Tape() as tape:
    a = tape.input(2.0, "a")
    b = tape.input(3.0, "b")
    u = a * b
    w = exp(b)
    L = u + a
```

(a) Write out the tape as in Table 4.1, and run `backward(tape, L.k)` by hand. Show that the sweep processes line 4 but that the line adds nothing, and that the gradient is still $(4, 2)$.

(b) Run `backward(tape, u.k)`, with the output $u = ab$ on line 3. Which lines have adjoint 0, and which of them does the sweep still process? Compare the result with the first row of the Jacobian in Section 4.5.

(c) The depth-first search of Section 4.8 starts from the node of the output. Which nodes does it visit for L, and for u?

Exercise 4.4. Build the graph of Table 3.1 from the nodes of Section 4.8 as `v3 = Node("v3", v1)`, `v4 = Node("v4", v3, v2)`, `v5 = Node("v5", v1, v2)`, `v6 = Node("v6", v5)` and `v7 = Node("v7", v4, v6)`, with `v1 = Node("v1")` and `v2 = Node("v2")`. Find the list that `topological_order(v7)` returns, and check that it is a topological order.

Exercise 4.5. Let $f(x, y) = (xy,\ x + y,\ x^2)$ and $\mathbf{w} = (1, -1, 2)$. Compute $\mathbf{w}^\top J_f(1, 2)$ from the Jacobian, and again by a reverse sweep on a trace of f with the outputs seeded by $\mathbf{w}$.

Exercise 4.6. Let $f = f_3 \circ f_2 \circ f_1$ with $f_1 \colon \mathbb{R}^n \to \mathbb{R}^p$, $f_2 \colon \mathbb{R}^p \to \mathbb{R}^p$ and $f_3 \colon \mathbb{R}^p \to \mathbb{R}$, so that $J_f = J_3 J_2 J_1$ by Theorem 2.1. Multiplying an $a \times b$ matrix by a $b \times c$ matrix takes abc multiplications. Count the multiplications needed to form J_f as $J_3(J_2 J_1)$ and as $(J_3 J_2)J_1$, and evaluate both for $n = p = 1000$. Which order corresponds to forward mode, and which to reverse mode?

Exercise 4.7. Let $f(\mathbf{x}) = x_1 x_2 \cdots x_n$, computed by the trace $p_2 = x_1 x_2$, $p_k = p_{k-1} x_k$ for $k = 3, \dots, n$.

(a) Write out the reverse sweep and show that it computes $\nabla f(\mathbf{x})$ with $2(n - 1)$ multiplications.

(b) How many multiplications does it take to compute each partial derivative $\prod_{j \neq i} x_j$ separately, and how many does forward mode take?

Exercise 4.8. Let $u_0 = 2x$, $w_0 = 3x$, and for $i = 1, \dots, k$ let $u_i = u_{i-1} w_{i-1}$ and $w_i = u_{i-1} + w_{i-1}$. Let $L = u_k + w_k$.

(a) Show that the computational graph has 2^{k+1} paths from x to L.

(b) Show that step 3 of the reverse sweep updates an accumulator $4k + 4$ times, once for each edge of the graph.

Exercise 4.9. Reverse mode differentiates a chain of T steps with checkpoints every K steps, as in Section 4.6.

(a) For $T = 10\,000$, find the K that minimises $T/K + K$ and the number of states stored.

(b) Inside each segment of length K_1, store every K_2th state during the recomputation, and recompute the sub-segments from these. Show that the method stores at most about $T/K_1 + K_1/K_2 + K_2$ states at once, that the recomputation adds at most two forward passes, and that $K_1 = T^{2/3}$, $K_2 = T^{1/3}$ give about $3T^{1/3}$ states.

Exercise 4.10. Add a primitive operation `tanh` to the engine of Section 4.7, using $\frac{d}{dt}\tanh t = 1 - \tanh^2 t$. Prove this formula from $\tanh t = (e^t - e^{-t})/(e^t + e^{-t})$, and use the engine to find the gradient of $\tanh(xy)$ at $(0.5, -1)$. How many lines would the tape of $\tanh(xy)$ have if `tanh` were derived from `exp` by the formula above instead, with e^t and e^{-t} each computed once?

Part III.

Learning as optimisation

5. Gradient descent

A model computes its outputs from a list of adjustable numbers, its *parameters*, collected in a vector θ. A list of examples of inputs with the outputs the model should give, the *training data*, turns the parameters into a single number, the *loss* $L(\theta)$, which measures how far the model's outputs are from the desired ones. Fitting a line $\theta_1 + \theta_2 s$ through points (s_i, y_i) by least squares is the simplest case: the loss is $\frac{1}{2}\sum_i (\theta_1 + \theta_2 s_i - y_i)^2$ (Exercise 5.4). *Training* the model means finding parameters that make the loss small. Theorem 1.8 says which way is downhill from any point: the direction of $-\nabla f$. Gradient descent takes a step that way, recomputes the gradient, and repeats. This chapter asks how large the steps should be and how quickly the method gets close to a minimum.

The answers depend on two numbers that bound the curvature of the function (how fast its slope changes, measured by second derivatives), one from above and one from below. The upper bound limits the step size, and with it in hand every step is guaranteed to lower the function. The lower bound, when it is positive, makes the error shrink by a fixed factor at every step. Their ratio decides how many steps are needed, and on a quadratic function it can be read off from the eigenvalues of a matrix. The last section treats gradient descent as a numerical method for a differential equation.

Throughout this chapter and the next, $f\colon \mathbb{R}^n \to \mathbb{R}$ is the function being minimised. For a model it is the loss $L(\theta)$, but here the letter L is reserved for the upper curvature bound of Section 5.2, which is its standard name.

5.1. The iteration

Theorem 1.8 says that at a point $\mathbf{a}$ the function decreases fastest in the direction of $-\nabla f(\mathbf{a})$. It does not say how far to go. Take $f(x, y) = x^2 + 3y^2$ from Example 1.5, at $(1, 1)$, where $f = 4$ and $\nabla f = (2, 6)$. Moving by 0.1 times the negative gradient gives

$$(1, 1) - 0.1\,(2, 6) = (0.8, 0.4), \qquad f(0.8, 0.4) = 0.64 + 0.48 = 1.12,$$

a large decrease. At the new point the gradient is $(1.6, 2.4)$, and the same rule gives the next point. Repeating this is gradient descent. The multiplier 0.1 sets the length of each move, and the rest of the chapter is about how to choose it.

Definition 5.1 (Gradient descent)**.** Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be differentiable, let $\theta_0 \in \mathbb{R}^n$ and let $\eta > 0$. *Gradient descent* with *learning rate* (or *step size*) η produces the sequence

$$\theta_{k+1} = \theta_k - \eta\,\nabla f(\theta_k), \qquad k = 0, 1, 2, \ldots \tag{5.1}$$

The points θ_k are the *iterates*, and θ_0 is the *starting point*.

Each new point is the old one moved a distance $\eta\|\nabla f(\theta_k)\|$ in the direction of steepest descent. Despite the name "step size", η is not the length of the step: it is the ratio of the step to the gradient. The direction of steepest descent comes from the derivative, which describes f only close to θ_k, so the step must be short. Its length shrinks as the gradient shrinks. Near a minimum the gradient is close to $\mathbf{0}$, and the steps become short without any change to η. A fixed η is therefore a natural first choice, and it is what this chapter studies.

Example 5.1. Continue Example 1.5 with $f(x, y) = x^2 + 3y^2$ and $\theta_0 = (1, 1)$. Since $\nabla f(x, y) = (2x, 6y)$, the iteration acts on each coordinate separately:

$$x_{k+1} = (1 - 2\eta)\,x_k, \qquad y_{k+1} = (1 - 6\eta)\,y_k.$$

Each is a first-order recurrence relation, with solution $x_k = (1-2\eta)^k$ and $y_k = (1-6\eta)^k$. With $\eta = 0.1$ the iterates are

$$(1,1), \quad (0.8,0.4), \quad (0.64,0.16), \quad (0.512,0.064), \ \ldots$$

and they approach the minimum at the origin. With $\eta = 0.4$ the factors are 0.2 and -1.4, so $y_k = (-1.4)^k$ changes sign at every step and grows without bound: $y_5 = -5.37824$. The step that works well for x is too long for y, the direction in which f curves more steeply.

The rest of the chapter explains this behaviour in general: why the safe step size depends on the steepest curvature, and what the shallowest curvature does to the speed.

5.2. Smoothness

In Example 5.1 the step $\eta = 0.4$ was too long because f curves steeply in the y-direction: there the gradient changes quickly from point to point, and a step chosen from the gradient at one point overshoots. The rate at which the gradient can change is the number that limits the step. For $f(x,y) = x^2 + 3y^2$, moving from $(1,1)$ to $(1,1.1)$ changes the gradient from $(2,6)$ to $(2,6.6)$, a change of 0.6 for a move of length 0.1: a ratio of 6. Moving instead to $(1.1,1)$ changes it to $(2.2,6)$, a ratio of 2. Since $\nabla f(\mathbf{x}) - \nabla f(\mathbf{y}) = (2(x_1 - y_1), 6(x_2 - y_2))$, no move gives a ratio above 6.

Definition 5.2 (L-smooth function)**.** Let $L > 0$. A differentiable function $f\colon \mathbb{R}^n \to \mathbb{R}$ is *L-smooth* if its gradient is *Lipschitz continuous with constant* L, which means

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \le L\,\|\mathbf{x} - \mathbf{y}\| \quad \text{for all } \mathbf{x}, \mathbf{y} \in \mathbb{R}^n.$$

In words: between any two points, the gradient changes by at most L times the distance between them. So $x^2 + 3y^2$ is 6-smooth. For a function of one variable the condition says that the slope f' changes by at most $L|x - y|$ between x and y, and by the mean value theorem that holds whenever $|f''| \le L$ everywhere. So L bounds how quickly the slope of the graph can change.

A bound on the gradient itself, $\|\nabla f\| \le G$, might look like the natural hypothesis, but it limits only how far a step goes, not whether it overshoots. The function $f(x) = \sqrt{x^2 + 10^{-4}}$, a smoothed version of $|x|$, has $|f'(x)| = |x|/\sqrt{x^2 + 10^{-4}} < 1$ everywhere. From $x_0 = 0.01$, where $f'(x_0) = 1/\sqrt{2} \approx 0.707$, a step with $\eta = 0.1$ lands at $x_1 \approx -0.0607$, and f rises from 0.0141 to 0.0615. The slope changes from about -1 to about 1 within a short interval around 0: $f''(0) = 100$, so f is 100-smooth, and $\eta = 0.1$ is ten times $1/L = 0.01$, the learning rate that the results below treat as safe. The function x^2 has the unbounded derivative $2x$, but it is 2-smooth, and the same $\eta = 0.1$ multiplies x by 0.8 and lowers f at every step. A step uses the gradient at its starting point, so what decides whether it is safe is how much the gradient changes along it, and that is what L bounds.

For several variables the second derivative is the Hessian matrix $\nabla^2 f$ (Definition 2.3), and the test for smoothness uses its eigenvalues. The proof restricts f to a line, a device used throughout this chapter.

Lemma 5.1. *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be differentiable, and let $\mathbf{x}, \mathbf{d} \in \mathbb{R}^n$. The function $\varphi(t) = f(\mathbf{x} + t\mathbf{d})$ of one variable is differentiable, with $\varphi'(t) = \nabla f(\mathbf{x} + t\mathbf{d}) \cdot \mathbf{d}$.*

Proof. If $\mathbf{d} = \mathbf{0}$, φ is constant and both sides are 0. Otherwise put $\mathbf{u} = \mathbf{d}/\|\mathbf{d}\|$ and $\mathbf{a} = \mathbf{x} + t\mathbf{d}$. With $s = h\|\mathbf{d}\|$,

$$\frac{\varphi(t + h) - \varphi(t)}{h} = \|\mathbf{d}\| \, \frac{f(\mathbf{a} + s\mathbf{u}) - f(\mathbf{a})}{s},$$

and as $h \to 0$, $s \to 0$ and the right side tends to $\|\mathbf{d}\| \, \nabla f(\mathbf{a}) \cdot \mathbf{u} = \nabla f(\mathbf{a}) \cdot \mathbf{d}$ by Theorem 1.7. □

A twice continuously differentiable function (one whose second partial derivatives exist and are continuous) has a symmetric Hessian by Theorem 2.5. By Theorem 2.9, a real symmetric matrix A has an orthonormal basis of eigenvectors $\mathbf{u}_1, \dots, \mathbf{u}_n$, with real eigenvalues $\lambda_1, \dots, \lambda_n$. The coefficients of any vector in that basis are dot products, $\mathbf{v} = \sum_i c_i \mathbf{u}_i$ with $c_i = \mathbf{u}_i \cdot \mathbf{v}$, and $\|\mathbf{v}\|^2 = \sum_i c_i^2$. Corollary 2.4 uses this to bound $\mathbf{v}^\top A \mathbf{v}$; the same idea bounds the length of $A\mathbf{v}$.

Lemma 5.2. *Let A be a real symmetric matrix whose eigenvalues all lie in the interval $[-M, M]$. Then $\|A\mathbf{v}\| \le M\|\mathbf{v}\|$ for every vector $\mathbf{v}$.*

Proof. Write $\mathbf{v} = \sum_i c_i \mathbf{u}_i$ in an orthonormal basis of eigenvectors. Then $A\mathbf{v} = \sum_i \lambda_i c_i \mathbf{u}_i$, so $\|A\mathbf{v}\|^2 = \sum_i \lambda_i^2 c_i^2 \le M^2 \sum_i c_i^2 = M^2 \|\mathbf{v}\|^2$. □

Integrals of vector-valued functions are taken component by component. One inequality about them is needed.

Lemma 5.3. *If $\mathbf{w} \colon [0,1] \to \mathbb{R}^n$ has continuous components, then $\left\| \int_0^1 \mathbf{w}(t)\, dt \right\| \le \int_0^1 \|\mathbf{w}(t)\|\, dt$.*

Proof. Let $\mathbf{W} = \int_0^1 \mathbf{w}(t)\, dt$. If $\mathbf{W} = \mathbf{0}$ there is nothing to prove. Otherwise, by linearity of the integral and Theorem 1.1,

$$\|\mathbf{W}\|^2 = \mathbf{W} \cdot \int_0^1 \mathbf{w}(t)\, dt = \int_0^1 \mathbf{W} \cdot \mathbf{w}(t)\, dt \le \|\mathbf{W}\| \int_0^1 \|\mathbf{w}(t)\|\, dt,$$

and dividing by $\|\mathbf{W}\|$ gives the result. □

Theorem 5.1 (A test for smoothness)**.** *Let $f \colon \mathbb{R}^n \to \mathbb{R}$ be twice continuously differentiable, let $L > 0$, and suppose that at every point $\mathbf{x}$ all the eigenvalues of $\nabla^2 f(\mathbf{x})$ lie in $[-L, L]$. Then f is L-smooth.*

Proof. Fix $\mathbf{x}$ and $\mathbf{y}$ and put $\mathbf{d} = \mathbf{y} - \mathbf{x}$. The partial derivative $\partial f/\partial x_i$ has continuous partial derivatives, so it is differentiable (Theorem 1.6), and its gradient is column i of the Hessian, which equals row i because the Hessian is symmetric (Theorem 2.5). By Lemma 5.1, $\psi_i(t) = \frac{\partial f}{\partial x_i}(\mathbf{x} + t\mathbf{d})$ has the continuous derivative $\psi_i'(t) = \big(\nabla^2 f(\mathbf{x} + t\mathbf{d})\,\mathbf{d}\big)_i$. The fundamental theorem of calculus, applied to each component, gives

$$\nabla f(\mathbf{y}) - \nabla f(\mathbf{x}) = \int_0^1 \nabla^2 f(\mathbf{x} + t\mathbf{d})\,\mathbf{d}\; dt\,:$$

the change in the gradient is the sum of the changes along the segment. By Lemma 5.3 and Lemma 5.2, $\|\nabla f(\mathbf{y}) - \nabla f(\mathbf{x})\| \le \int_0^1 L\|\mathbf{d}\|\, dt = L\|\mathbf{y} - \mathbf{x}\|$. □

Example 5.2 (Quadratic functions)**.** Let A be a symmetric $n \times n$ matrix, $\mathbf{b} \in \mathbb{R}^n$ and

$$f(\theta) = \tfrac{1}{2}\theta^\top A\theta - \mathbf{b}^\top\theta = \frac{1}{2}\sum_{i,j} a_{ij}\theta_i\theta_j - \sum_i b_i\theta_i.$$

By Example 1.8 and Example 1.6, $\nabla f(\theta) = A\theta - \mathbf{b}$, and the Hessian is A at every point: entry a_{ij} is the rate at which $\partial f/\partial\theta_j$ changes as θ_i changes, the same everywhere. So $\nabla f(\mathbf{x}) - \nabla f(\mathbf{y}) = A(\mathbf{x} - \mathbf{y})$, and by Lemma 5.2 f is L-smooth with L the largest absolute value of an eigenvalue of A. No smaller constant works: if $A\mathbf{u} = \lambda\mathbf{u}$ with $|\lambda| = L$, then $\|\nabla f(\mathbf{u}) - \nabla f(\mathbf{0})\| = L\|\mathbf{u}\|$.

The function $x^2 + 3y^2$ of Example 5.1 is of this form with $A = \operatorname{diag}(2, 6)$ and $\mathbf{b} = \mathbf{0}$. The diagonal entries are the curvatures in the x- and y-directions, and the zero off-diagonal entries say that $\partial f/\partial x$ does not depend on y, nor $\partial f/\partial y$ on x. So it is 6-smooth, matching the ratio found above.

Example 5.3. The *softplus* function $f(x) = \ln(1 + e^x)$ appears in the loss of a model that predicts one of two outcomes. Its derivatives are

$$f'(x) = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}}, \qquad f''(x) = f'(x)(1 - f'(x)).$$

Since $0 < f'(x) < 1$ and $p(1-p) \le \frac{1}{4}$ for every real p (because $\frac{1}{4} - p(1-p) = (p - \frac{1}{2})^2$), we have $0 < f''(x) \le \frac{1}{4}$, with equality at $x = 0$. By Theorem 5.1 with $n = 1$, f is $\frac{1}{4}$-smooth.

5.3. The descent lemma

To guarantee that a step lowers f, we need an upper bound on f at the new point in terms of what is known at the old one. The tangent plane at $\mathbf{x}$, the graph of $\mathbf{y} \mapsto f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x})$, approximates f near $\mathbf{x}$, and smoothness bounds how far the graph of f can rise above it. For $f(x, y) = x^2 + 3y^2$ at $\mathbf{x} = (1, 1)$, take $\mathbf{y} = (0.9, 0.7)$. The tangent plane predicts $4 + (2, 6) \cdot (-0.1, -0.3) = 2$, the true value is $f(0.9, 0.7) = 0.81 + 1.47 = 2.28$, and the excess 0.28 is less than $\frac{6}{2}\|(-0.1, -0.3)\|^2 = 0.3$. The lemma says that this always happens.

Lemma 5.4 (Descent lemma)**.** *Let $f \colon \mathbb{R}^n \to \mathbb{R}$ be L-smooth. For all $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^n$,*

$$f(\mathbf{y}) \le f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x}) + \frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2. \tag{5.2}$$

The right side is the tangent plane plus the term $\frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2$, whose graph is a paraboloid (a bowl) with its lowest point at $\mathbf{x}$. So the graph of f never rises above the tangent plane by more than $\frac{L}{2}$ times the squared distance from $\mathbf{x}$.

Proof. Put $\mathbf{d} = \mathbf{y} - \mathbf{x}$ and $\varphi(t) = f(\mathbf{x} + t\mathbf{d})$. By Lemma 5.1, $\varphi'(t) = \nabla f(\mathbf{x} + t\mathbf{d}) \cdot \mathbf{d}$, which is continuous in t because ∇f is Lipschitz continuous. By the fundamental theorem of calculus,

$$f(\mathbf{y}) - f(\mathbf{x}) - \nabla f(\mathbf{x}) \cdot \mathbf{d} = \int_0^1 (\varphi'(t) - \varphi'(0))\, dt = \int_0^1 (\nabla f(\mathbf{x} + t\mathbf{d}) - \nabla f(\mathbf{x})) \cdot \mathbf{d}\, dt.$$

By Theorem 1.1 and smoothness, the integrand is at most $\|\nabla f(\mathbf{x} + t\mathbf{d}) - \nabla f(\mathbf{x})\|\,\|\mathbf{d}\| \le Lt\|\mathbf{d}\|^2$, and $\int_0^1 Lt\|\mathbf{d}\|^2\, dt = \frac{L}{2}\|\mathbf{d}\|^2$. □

The same proof bounds the integrand below by $-Lt\|\mathbf{d}\|^2$, so the graph of f also stays above the downward parabola $f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot \mathbf{d} - \frac{L}{2}\|\mathbf{d}\|^2$. Only the upper bound is needed here. Applied to a gradient step, it shows how much each step must gain.

Corollary 5.1. *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be L-smooth and $\mathbf{x}^+ = \mathbf{x} - \eta\nabla f(\mathbf{x})$ with $\eta > 0$. Then*

$$f(\mathbf{x}^+) \le f(\mathbf{x}) - \eta\Big(1 - \frac{L\eta}{2}\Big)\|\nabla f(\mathbf{x})\|^2.$$

In particular, if $0 < \eta \le 1/L$ then $f(\mathbf{x}^+) \le f(\mathbf{x}) - \frac{\eta}{2}\|\nabla f(\mathbf{x})\|^2$, and with $\eta = 1/L$ each step lowers f by at least $\|\nabla f(\mathbf{x})\|^2/(2L)$.

Proof. Put $\mathbf{y} = \mathbf{x}^+$ in Equation 5.2: $\nabla f(\mathbf{x}) \cdot (\mathbf{x}^+ - \mathbf{x}) = -\eta\|\nabla f(\mathbf{x})\|^2$ and $\|\mathbf{x}^+ - \mathbf{x}\|^2 = \eta^2\|\nabla f(\mathbf{x})\|^2$. If $\eta \le 1/L$ then $1 - L\eta/2 \ge 1/2$. □

Any $\eta < 2/L$ makes the bound a strict decrease whenever the gradient is not zero, and $\eta = 1/L$ gives the largest guaranteed decrease, because $\eta(1 - L\eta/2)$ is largest there.

Example 5.4. For $f(x, y) = x^2 + 3y^2$, which is 6-smooth, take $\eta = 1/6$ at $(1, 1)$. The gradient is $(2, 6)$, with $\|\nabla f\|^2 = 40$, so Corollary 5.1 promises a decrease of at least $40/12 = 10/3$. The step goes to $(1, 1) - \frac{1}{6}(2, 6) = (\frac{2}{3}, 0)$, where $f = \frac{4}{9}$. The actual decrease is $4 - \frac{4}{9} = \frac{32}{9} \approx 3.56$.

5.4. Functions that are not convex

The loss of a neural network is not convex (Section 5.5 defines the term), and has many critical points, points where the gradient is zero (Definition 2.4). Some are minima, others maxima or saddle points (Theorem 2.10). Gradient descent cannot be expected to find the lowest point of such a function, but smoothness alone guarantees that it finds points where the gradient is small.

Example 5.5. Let $f(\theta) = \cos\theta$ on $\mathbb{R}$. Then $f'(\theta) = -\sin\theta$ and $|f''(\theta)| = |\cos\theta| \le 1$, so f is 1-smooth. Gradient descent with $\eta = 1$ is $\theta_{k+1} = \theta_k + \sin\theta_k$. From $\theta_0 = 1$, to six significant figures,

$$\theta_1 = 1.84147, \quad \theta_2 = 2.80506, \quad \theta_3 = 3.13528, \quad \theta_4 = 3.14159,$$

approaching the minimum at π. The squared gradients $\sin^2\theta_k$ for $k = 0, 1, 2, 3$ are 0.708, 0.929, 0.109 and 0.0000399. From $\theta_0 = 0$, where cos has a maximum, the gradient is zero and $\theta_k = 0$ for every k.

Theorem 5.2 (Gradient descent on smooth functions)**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be L-smooth and bounded below, with $f^* = \inf f$ its greatest lower bound. Run gradient descent from any θ_0 with $0 < \eta \le 1/L$. Then $f(\theta_k)$ never increases, and for every $k \ge 1$,*

$$\min_{0\le j<k} \|\nabla f(\theta_j)\|^2 \le \frac{1}{k}\sum_{j=0}^{k-1} \|\nabla f(\theta_j)\|^2 \le \frac{2(f(\theta_0) - f^*)}{\eta k}.$$

With $\eta = 1/L$ the right side is $2L(f(\theta_0) - f^*)/k$. Also $\nabla f(\theta_k) \to \mathbf{0}$ as $k \to \infty$.

Proof. By Corollary 5.1, $\frac{\eta}{2}\|\nabla f(\theta_j)\|^2 \le f(\theta_j) - f(\theta_{j+1})$ for each j, so f never increases. Adding these inequalities for $j = 0, \dots, k-1$, the right sides telescope:

$$\frac{\eta}{2}\sum_{j=0}^{k-1} \|\nabla f(\theta_j)\|^2 \le f(\theta_0) - f(\theta_k) \le f(\theta_0) - f^*.$$

Divide by $\eta k/2$. The minimum of k numbers is at most their mean. The partial sums of the series $\sum_j \|\nabla f(\theta_j)\|^2$ are bounded and its terms are not negative, so it converges and its terms tend to 0. □

In words: the average of the first k squared gradients falls at least as fast as $1/k$, so some early iterate already has a small gradient. In Example 5.5,

$f^* = -1$, and the theorem bounds the smallest of the first k squared gradients by $2(\cos 1 + 1)/k \approx 3.08/k$. The theorem does not say that θ_k converges, or that it approaches a minimum. A critical point might be a maximum or a saddle point, and if θ_0 is such a point gradient descent never moves, as the start $\theta_0 = 0$ shows.

5.5. Convex functions

In Example 5.5 a point with zero gradient was a maximum. A shape condition rules this out. For $f(x) = x^2$, take the chord joining the points of the graph at $x = -1$ and $x = 3$. At the midpoint $x = 1$ the chord has height $(1 + 9)/2 = 5$, above $f(1) = 1$, and the graph lies below the chord all the way between. For cos on $[-\pi, \pi]$ the chord joining the ends is the line at height -1, but $\cos 0 = 1$ lies above it: the graph rises over the chord, and the maximum at 0 is the top of that rise. Functions whose graphs never rise above a chord are the subject of this section. For them a zero gradient does mean a minimum, and the function values along gradient descent converge.

Definition 5.3 (Convex function)**.** A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is *convex* if

$$f(t\mathbf{x} + (1-t)\mathbf{y}) \le tf(\mathbf{x}) + (1-t)f(\mathbf{y})$$

for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and all $t \in [0, 1]$.

As t runs from 0 to 1, the point $t\mathbf{x} + (1-t)\mathbf{y}$ runs along the segment from $\mathbf{y}$ to $\mathbf{x}$, and the right side runs along the chord. The inequality says that the graph over the segment lies on or below the chord.

For differentiable functions an equivalent condition uses tangent planes instead of chords. For $f(x) = x^2$ the tangent line at $x = 1$ is $1 + 2(y - 1)$, and $y^2 - (1 + 2(y - 1)) = (y - 1)^2 \ge 0$: the graph lies above the tangent line everywhere.

Theorem 5.3 (First-order characterisation of convexity)**.** *A differentiable function $f\colon \mathbb{R}^n \to \mathbb{R}$ is convex if and only if*

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x}) \quad \textit{for all } \mathbf{x}, \mathbf{y} \in \mathbb{R}^n. \tag{5.3}$$

Proof. Suppose f is convex, and fix $\mathbf{x}$ and $\mathbf{y}$. For $0 < t \leq 1$, convexity with the points $\mathbf{y}$ and $\mathbf{x}$ gives $f(\mathbf{x} + t(\mathbf{y} - \mathbf{x})) \leq f(\mathbf{x}) + t(f(\mathbf{y}) - f(\mathbf{x}))$, so

$$\frac{f(\mathbf{x} + t(\mathbf{y} - \mathbf{x})) - f(\mathbf{x})}{t} \leq f(\mathbf{y}) - f(\mathbf{x}).$$

As $t \to 0^+$ the left side tends to $\nabla f(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x})$ by Lemma 5.1, which gives Equation 5.3.

Conversely, suppose Equation 5.3 holds. Fix $\mathbf{x}$, $\mathbf{y}$ and $t \in [0, 1]$, and let $\mathbf{z} = t\mathbf{x} + (1 - t)\mathbf{y}$. Applying Equation 5.3 at $\mathbf{z}$ twice,

$$f(\mathbf{x}) \geq f(\mathbf{z}) + \nabla f(\mathbf{z}) \cdot (\mathbf{x} - \mathbf{z}), \qquad f(\mathbf{y}) \geq f(\mathbf{z}) + \nabla f(\mathbf{z}) \cdot (\mathbf{y} - \mathbf{z}).$$

Multiply the first by t and the second by $1 - t$ and add. Since $t(\mathbf{x} - \mathbf{z}) + (1 - t)(\mathbf{y} - \mathbf{z}) = \mathbf{z} - \mathbf{z} = \mathbf{0}$, the gradient terms cancel and leave $tf(\mathbf{x}) + (1 - t)f(\mathbf{y}) \geq f(\mathbf{z})$. □

A *minimiser* of f is a point θ^* with $f(\theta^*) \leq f(\mathbf{y})$ for every $\mathbf{y}$.

Corollary 5.2. *If $f\colon \mathbb{R}^n \to \mathbb{R}$ is convex and differentiable and $\nabla f(\theta^*) = \mathbf{0}$, then θ^* is a minimiser of f.*

Proof. Put $\mathbf{x} = \theta^*$ in Equation 5.3. □

Conversely, at any minimiser of a differentiable function the gradient is zero (Theorem 2.7). The next theorem gives a practical test for convexity, and a stronger property that Section 5.6 uses.

Theorem 5.4 (Curvature bounded below)**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be twice continuously differentiable, let $\mu \geq 0$, and suppose that at every point all the eigenvalues of $\nabla^2 f(\mathbf{x})$ are at least μ. Then for all $\mathbf{x}$ and $\mathbf{y}$,*

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x}) + \frac{\mu}{2}\|\mathbf{y} - \mathbf{x}\|^2.$$

In particular (taking $\mu = 0$), a function whose Hessian is positive semidefinite at every point (Definition 2.5), that is, has no negative eigenvalues, is convex.

Proof. Put $\mathbf{d} = \mathbf{y} - \mathbf{x}$. By Taylor's theorem in the form Equation 2.6, there is a $\tau \in (0, 1)$ with

$$f(\mathbf{y}) = f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot \mathbf{d} + \tfrac{1}{2}\,\mathbf{d}^\top \nabla^2 f(\mathbf{x} + \tau\mathbf{d})\,\mathbf{d}.$$

The Hessian is symmetric (Theorem 2.5) with all eigenvalues at least μ, so by Corollary 2.4 the last term is at least $\frac{\mu}{2}\|\mathbf{d}\|^2$. With $\mu = 0$ the inequality is Equation 5.3, so f is convex by Theorem 5.3. □

So the quadratic of Example 5.2 is convex when A is positive semidefinite, and the function $\ln(1 + e^x)$ of Example 5.3 is convex because $f'' > 0$. That function also shows that a convex function need not have a minimiser: it is positive and tends to 0 as $x \to -\infty$, but never reaches 0. The next theorem assumes a minimiser exists.

Theorem 5.5 (Gradient descent on convex functions)**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be convex and L-smooth, with a minimiser θ^*, and let $f^* = f(\theta^*)$. Run gradient descent with $0 < \eta \leq 1/L$. Then the distance $\|\theta_k - \theta^*\|$ never increases, and for every $k \geq 1$,*

$$f(\theta_k) - f^* \leq \frac{\|\theta_0 - \theta^*\|^2}{2\eta k}.$$

Proof. Fix j, write $\mathbf{g} = \nabla f(\theta_j)$, and expand the square of the distance after one step:

$$\|\theta_{j+1} - \theta^*\|^2 = \|\theta_j - \theta^* - \eta\mathbf{g}\|^2 = \|\theta_j - \theta^*\|^2 - 2\eta\,\mathbf{g}\cdot(\theta_j - \theta^*) + \eta^2\|\mathbf{g}\|^2.$$

Rearranged,

$$\mathbf{g}\cdot(\theta_j - \theta^*) - \frac{\eta}{2}\|\mathbf{g}\|^2 = \frac{1}{2\eta}\Big(\|\theta_j - \theta^*\|^2 - \|\theta_{j+1} - \theta^*\|^2\Big). \tag{5.4}$$

By Corollary 5.1, $f(\theta_{j+1}) \le f(\theta_j) - \frac{\eta}{2}\|\mathbf{g}\|^2$, and by Equation 5.3 with $\mathbf{x} = \theta_j$ and $\mathbf{y} = \theta^*$, $f(\theta_j) \le f^* + \mathbf{g}\cdot(\theta_j - \theta^*)$. Together with Equation 5.4 these give

$$0 \le f(\theta_{j+1}) - f^* \le \frac{1}{2\eta}\Big(\|\theta_j - \theta^*\|^2 - \|\theta_{j+1} - \theta^*\|^2\Big),$$

so the distance never increases. Adding these inequalities for $j = 0, \dots, k-1$, the right sides telescope:

$$\sum_{j=1}^{k}\big(f(\theta_j) - f^*\big) \le \frac{1}{2\eta}\Big(\|\theta_0 - \theta^*\|^2 - \|\theta_k - \theta^*\|^2\Big) \le \frac{\|\theta_0 - \theta^*\|^2}{2\eta}.$$

The values $f(\theta_j)$ never increase (Theorem 5.2), so each term on the left is at least $f(\theta_k) - f^*$, and the left side is at least $k(f(\theta_k) - f^*)$. □

The error in f falls like $1/k$: to make it ten times smaller takes about ten times as many steps.

Example 5.6. The *Huber function*

$$h(x) = \begin{cases} \frac{1}{2}x^2, & |x| \le 1, \\ |x| - \frac{1}{2}, & |x| > 1, \end{cases} \qquad h'(x) = \begin{cases} x, & |x| \le 1, \\ 1, & x > 1, \\ -1, & x < -1, \end{cases}$$

is used as a loss that grows slowly for large errors. The two pieces and their derivatives agree at $x = \pm 1$, so h is differentiable. Its derivative

clips x to the interval $[-1, 1]$, which never moves two numbers further apart, so h is 1-smooth. The derivative never decreases, so for $y > x$, $h(y) - h(x) - h'(x)(y - x) = \int_x^y (h'(s) - h'(x))\, ds \geq 0$, and similarly for $y < x$. By Theorem 5.3, h is convex, with minimiser 0.

Gradient descent with $\eta = 1$ from $x_0 = 10$ moves one unit at a time while $|x_k| > 1$: $x_k = 10 - k$ for $k \leq 9$, and then $x_{10} = 1 - h'(1) = 0$. So $h(x_k) = 9.5 - k$ for $k \leq 9$. The bound of Theorem 5.5 is $|x_0 - 0|^2/(2k) = 50/k$; at $k = 9$ it is about 5.56, and the true value is 0.5. The theorem covers every convex 1-smooth function, and most are better behaved than the worst case.

5.6. Strong convexity

The $1/k$ rate of Theorem 5.5 is slow, and in Example 5.1 gradient descent did much better: every coordinate shrank by a fixed factor at every step. The reason is that $x^2 + 3y^2$ curves upwards in every direction, by at least a fixed amount. Its graph lies above every tangent plane by a definite margin: for any $\mathbf{x}$ and $\mathbf{y}$,

$$f(\mathbf{y}) - f(\mathbf{x}) - \nabla f(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x}) = (y_1 - x_1)^2 + 3(y_2 - x_2)^2 \geq \|\mathbf{y} - \mathbf{x}\|^2,$$

so the gap between the graph and the tangent plane grows at least like the squared distance. The definition writes the margin as $\frac{\mu}{2}\|\mathbf{y} - \mathbf{x}\|^2$, so here $\mu = 2$, the smaller diagonal entry of the Hessian $\operatorname{diag}(2, 6)$.

The Huber function of Example 5.6 has no such margin. For $x > 1$ its graph is a straight line, which coincides with its own tangent: at $x = 2$ and $y = 3$, $h(3) - h(2) - h'(2)(3 - 2) = 2.5 - 1.5 - 1 = 0$. There gradient descent covered only one unit per step.

Definition 5.4 (Strongly convex function)**.** Let $\mu > 0$. A differentiable function $f\colon \mathbb{R}^n \to \mathbb{R}$ is *μ-strongly convex* if

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x}) + \frac{\mu}{2}\|\mathbf{y} - \mathbf{x}\|^2 \quad \text{for all } \mathbf{x}, \mathbf{y} \in \mathbb{R}^n.$$

Compared with Equation 5.3, the graph must clear each tangent plane by a bowl of curvature μ. A strongly convex function is therefore convex, by Theorem 5.3. By Theorem 5.4, a twice continuously differentiable function is μ-strongly convex when every eigenvalue of its Hessian is at least μ at every point. The quadratic $\frac{1}{2}\theta^\top A\theta - \mathbf{b}^\top\theta$ with A symmetric is μ-strongly convex when the smallest eigenvalue of A is $\mu > 0$, and $x^2 + 3y^2$ is 2-strongly convex and 6-smooth.

Lemma 5.5. *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be μ-strongly convex with a minimiser θ^*, and $f^* = f(\theta^*)$. For every $\mathbf{x}$,*

$$\frac{\mu}{2}\|\mathbf{x} - \theta^*\|^2 \le f(\mathbf{x}) - f^* \le \frac{1}{2\mu}\|\nabla f(\mathbf{x})\|^2.$$

In particular, θ^* is the only minimiser.

Proof. For the left inequality, put θ^* in place of $\mathbf{x}$ and $\mathbf{x}$ in place of $\mathbf{y}$ in the definition, and use $\nabla f(\theta^*) = \mathbf{0}$ (Theorem 2.7). For the right inequality, fix $\mathbf{x}$, write $\mathbf{g} = \nabla f(\mathbf{x})$, and complete the square:

$$f(\mathbf{x}) + \mathbf{g}\cdot(\mathbf{y}-\mathbf{x}) + \frac{\mu}{2}\|\mathbf{y}-\mathbf{x}\|^2 = f(\mathbf{x}) - \frac{\|\mathbf{g}\|^2}{2\mu} + \frac{\mu}{2}\Big\|\mathbf{y}-\mathbf{x}+\frac{\mathbf{g}}{\mu}\Big\|^2 \ge f(\mathbf{x}) - \frac{\|\mathbf{g}\|^2}{2\mu}.$$

By the definition with $\mathbf{y} = \theta^*$, f^* is at least the left side, hence at least $f(\mathbf{x}) - \|\mathbf{g}\|^2/(2\mu)$. If $\mathbf{x}$ is another minimiser, the left inequality gives $\|\mathbf{x} - \theta^*\|^2 \le 0$. □

The right inequality says that the gradient cannot be small unless f is close to its minimum. That fails for cos, where the gradient is zero at every maximum, and for the Huber function, whose derivative is 1 at every $x > 1$ while the gap $h(x) - h(0) = x - \frac{1}{2}$ grows without bound. It is the extra ingredient in the next theorem. By Corollary 5.1, a step with $\eta \le 1/L$ lowers f by at least $\frac{\eta}{2}\|\nabla f\|^2$, and by the right inequality this is at least $\eta\mu\,(f - f^*)$. So every step removes at least the fixed fraction $\eta\mu$ of the remaining gap $f - f^*$, and the gap decays at least as fast as the solution $(1-\eta\mu)^k u_0$ of the first-order recurrence $u_{k+1} = (1-\eta\mu)u_k$.

Theorem 5.6 (Gradient descent on strongly convex functions)**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be L-smooth and μ-strongly convex, with a minimiser θ^* and $f^* = f(\theta^*)$. Run gradient descent with $0 < \eta \le 1/L$. Then $0 \le 1-\eta\mu < 1$, and for every k,*

$$f(\theta_k) - f^* \le (1-\eta\mu)^k (f(\theta_0) - f^*), \qquad \|\theta_k - \theta^*\|^2 \le \frac{2}{\mu}(1-\eta\mu)^k (f(\theta_0) - f^*).$$

With $\eta = 1/L$ the factor is $1 - 1/\kappa$, where $\kappa = L/\mu$.

Proof. Exercise 5.6 shows that $\mu \le L$, so $0 < \eta\mu \le \mu/L \le 1$. By Corollary 5.1 and Lemma 5.5,

$$f(\theta_{k+1}) - f^* \le f(\theta_k) - f^* - \frac{\eta}{2}\|\nabla f(\theta_k)\|^2 \le f(\theta_k) - f^* - \eta\mu(f(\theta_k) - f^*).$$

So each step multiplies $f - f^*$ by at most $1 - \eta\mu$, and induction on k gives the first inequality. The second follows from the first and the left inequality of Lemma 5.5. □

An error bound of the form Cr^k with $0 \le r < 1$ is called *linear convergence* with rate r. The name describes the logarithm of the bound, $\log_{10} C - k \log_{10}(1/r)$, which falls linearly in k: each step adds about $\log_{10}(1/r)$ correct decimal digits. The $1/k$ bound of Theorem 5.5 is slower, since each extra correct digit multiplies the number of steps by ten.

The rate $1 - \eta\mu$ is smallest at the longest allowed step, $\eta = 1/L$. That step suits the steepest direction, where the curvature can be as large as L. In a direction where the curvature is only μ it removes just the fraction μ/L of the gap. So the ratio of the two bounds sets the speed.

Definition 5.5 (Condition number)**.** Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be L-smooth and μ-strongly convex, where L is the smallest constant and μ the largest constant for which these hold. The *condition number* of f is $\kappa = L/\mu$.

Both constants exist because the inequalities in Definition 5.2 and Definition 5.4 are not strict, and $\kappa \geq 1$ by Exercise 5.6. A condition number close to 1 means that f curves by about the same amount in every direction, and its level curves are close to circles. A large condition number means a long, narrow valley. Theorem 5.6 holds for any valid pair of constants, and the best pair gives the best bound. Since $1 - x \leq e^{-x}$, after k steps with $\eta = 1/L$ the gap $f - f^*$ has shrunk by at least $e^{-k/\kappa}$, and $k = \kappa \ln(1/\varepsilon)$ steps reduce it by a factor ε. A function whose curvature varies a lot between directions needs many steps.

Example 5.7. For $f(x, y) = x^2 + 3y^2$, $\mu = 2$ and $L = 6$, so $\kappa = 3$, and with $\eta = 1/6$ the theorem promises that f falls by a factor of at least $1 - 1/3 = 2/3$ per step. The iteration is $x_{k+1} = \frac{2}{3}x_k$, $y_{k+1} = 0$, so from $(1, 1)$, $f(\theta_k) = (\frac{2}{3})^{2k} = (\frac{4}{9})^k$ for $k \geq 1$: a factor of $\frac{4}{9}$ per step, better than the guaranteed $\frac{2}{3}$.

5.7. Quadratic functions and the condition number

Near a minimum, a twice differentiable function is close to its second-order Taylor polynomial (Equation 2.7), a quadratic. On a quadratic, gradient descent can be solved exactly. Example 5.1 already did this for $x^2 + 3y^2$: each coordinate obeyed its own first-order recurrence, with factors $1 - 2\eta$ and $1 - 6\eta$, where 2 and 6 are the eigenvalues of the Hessian. For a general quadratic the coordinates are mixed together, but in the basis of eigenvectors of the Hessian they separate in the same way.

So let A be a symmetric $n \times n$ matrix whose eigenvalues all lie in an interval $[\mu, L]$ with $\mu > 0$, where μ and L are the smallest and largest eigenvalues; such a matrix is positive definite (Definition 2.5). By Theorem 2.9, A has an orthonormal basis of eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_n$, with $A\mathbf{u}_i = \lambda_i \mathbf{u}_i$. Let $\mathbf{b} \in \mathbb{R}^n$ and

$$f(\theta) = \tfrac{1}{2}\theta^\top A\theta - \mathbf{b}^\top \theta. \tag{5.5}$$

By Example 5.2 and Theorem 5.4, f is L-smooth and μ-strongly convex. These are the best constants. No smaller L works, by Example 5.2. No larger μ works, because $f(\mathbf{x}+\mathbf{d}) - f(\mathbf{x}) - \nabla f(\mathbf{x}) \cdot \mathbf{d} = \frac{1}{2}\mathbf{d}^\top A\mathbf{d}$, which is exactly $\frac{\mu}{2}\|\mathbf{d}\|^2$ when $\mathbf{d}$ is an eigenvector for the eigenvalue μ. So the condition number of f is $\kappa = L/\mu$, the ratio of the largest to the smallest eigenvalue of A; it is also called the condition number of the matrix A. The matrix A is invertible, because 0 is not an eigenvalue, so $\theta^* = A^{-1}\mathbf{b}$ is the only point where $\nabla f(\theta) = A\theta - \mathbf{b}$ is zero, and by Corollary 5.2 it is the minimiser. Since $\mathbf{b} = A\theta^*$, the gradient is $A(\theta - \theta^*)$.

Theorem 5.7 (Gradient descent on a quadratic)**.** *Let A be a real symmetric $n \times n$ matrix with smallest eigenvalue $\mu > 0$ and largest eigenvalue L, with orthonormal eigenvectors $\mathbf{u}_1, \dots, \mathbf{u}_n$ and eigenvalues $\lambda_1, \dots, \lambda_n$. Let $\mathbf{b} \in \mathbb{R}^n$, let f be the quadratic Equation 5.5, with minimiser $\theta^* = A^{-1}\mathbf{b}$, and run gradient descent with step $\eta > 0$ from any θ_0. Let $\mathbf{e}_k = \theta_k - \theta^*$ be the error. Then:*

1. *$\mathbf{e}_{k+1} = (I - \eta A)\,\mathbf{e}_k$, and if $\mathbf{e}_0 = \sum_i c_i \mathbf{u}_i$ then $\mathbf{e}_k = \sum_i (1-\eta\lambda_i)^k c_i\,\mathbf{u}_i$.*
2. *$\|\mathbf{e}_k\| \le \rho(\eta)^k \|\mathbf{e}_0\|$, where $\rho(\eta) = \max_i |1 - \eta\lambda_i| = \max(|1-\eta\mu|, |1-\eta L|)$, and equality holds for some non-zero $\mathbf{e}_0$.*
3. *$\theta_k \to \theta^*$ for every θ_0 if and only if $\eta < 2/L$.*
4. *$\rho(\eta)$ is smallest at $\eta = 2/(\mu + L)$, where*

$$\rho = \frac{L-\mu}{L+\mu} = \frac{\kappa - 1}{\kappa + 1}, \qquad \kappa = \frac{L}{\mu}.$$

Proof.

(1) $\mathbf{e}_{k+1} = \mathbf{e}_k - \eta A(\theta_k - \theta^*) = (I - \eta A)\mathbf{e}_k$. Since $(I - \eta A)\mathbf{u}_i = (1 - \eta\lambda_i)\mathbf{u}_i$, each application of $I - \eta A$ multiplies the coefficient of $\mathbf{u}_i$ by $1 - \eta\lambda_i$.

(2) Each λ_i lies in $[\mu, L]$, so $1 - \eta L \le 1 - \eta\lambda_i \le 1 - \eta\mu$, and $|1 - \eta\lambda_i|$ is at most the larger of $|1 - \eta\mu|$ and $|1 - \eta L|$; both are attained, since μ and L are eigenvalues. By (1), $\|\mathbf{e}_k\|^2 = \sum_i (1 - \eta\lambda_i)^{2k} c_i^2 \le \rho^{2k} \sum_i c_i^2$. If $|1 - \eta\lambda_j| = \rho$, then $\mathbf{e}_0 = \mathbf{u}_j$ gives equality.

(3) If $\eta < 2/L$ then $-1 < 1 - \eta L \le 1 - \eta\lambda_i \le 1 - \eta\mu < 1$ for every i, so $\rho < 1$ and $\mathbf{e}_k \to \mathbf{0}$ by (2). If $\eta \ge 2/L$ then $1 - \eta L \le -1$, and the starting error $\mathbf{e}_0 = \mathbf{u}_j$, an eigenvector with eigenvalue L, gives $\|\mathbf{e}_k\| = |1 - \eta L|^k \ge 1$ for every k.

(4) Let $a = 1 - \eta\mu$ and $b = 1 - \eta L$, so that $b \le a$. Then $\max(|a|, |b|) = \max(a, -a, b, -b) = \max(a, -b)$, because $b \le a$ and $-a \le -b$. So $\rho(\eta) = \max(1 - \eta\mu,\ \eta L - 1)$. The first term decreases and the second increases with η, and they are equal when $\eta = 2/(\mu + L)$, where both are $(L - \mu)/(L + \mu)$. For a smaller η the first term is larger than this, and for a larger η the second is. Dividing the numerator and denominator by μ gives $(\kappa - 1)/(\kappa + 1)$.

□

Part 1 says that one step of gradient descent multiplies the error by the matrix $I - \eta A$, so $\mathbf{e}_k = (I - \eta A)^k \mathbf{e}_0$: a first-order recurrence, with a matrix in place of a number. Entry (i, j) of $I - \eta A$ is $\delta_{ij} - \eta a_{ij}$, where δ_{ij} is 1 if $i = j$ and 0 otherwise. It is the amount by which coordinate j of the old error contributes to coordinate i of the new one. For $x^2 + 3y^2$, $A = \operatorname{diag}(2, 6)$ and $I - \eta A = \operatorname{diag}(1 - 2\eta, 1 - 6\eta)$: the off-diagonal entries are zero, so the coordinates never mix, and each is multiplied by its own factor, as in Example 5.1. For a general symmetric A the off-diagonal entries mix the coordinates, but $I - \eta A$ has the same eigenvectors as A, with eigenvalues $1 - \eta\lambda_i$. In the basis $\mathbf{u}_1, \dots, \mathbf{u}_n$ it acts as a diagonal matrix again: the eigenvalue $1 - \eta\lambda_i$ is the factor by which one step multiplies the component of the error along $\mathbf{u}_i$. The number $\rho(\eta)$ is the largest absolute value of an eigenvalue of $I - \eta A$, which is called the *spectral radius* of $I - \eta A$. A component whose factor is negative changes sign at every step.

With $\eta = 1/L$, which Theorem 5.6 uses, $\rho = 1 - 1/\kappa$, and the gap $f - f^* = \frac{1}{2}\mathbf{e}_k^\top A \mathbf{e}_k$ shrinks by at least $(1 - 1/\kappa)^2$ per step, better than the factor $1 - 1/\kappa$ that the general theorem guarantees. For large κ the best rate $(\kappa - 1)/(\kappa + 1) = 1 - 2/(\kappa + 1)$ is not much better. The number of

steps needed to divide the error by 10^6 is about $\frac{\kappa}{2}\ln 10^6$, so it grows in proportion to the condition number.

Example 5.8 (The zigzag)**.** Return to $f(x,y) = x^2+3y^2$ from Example 1.5. Here $A = \operatorname{diag}(2,6)$, $\mathbf{b} = \mathbf{0}$, $\theta^* = \mathbf{0}$, $\mu = 2$, $L = 6$ and $\kappa = 3$. Gradient descent converges exactly when $0 < \eta < \frac{1}{3}$, which matches Example 5.1: $\eta = 0.4$ is too large. The best fixed step is $\eta = 2/8 = \frac{1}{4}$, with $\rho = \frac{3-1}{3+1} = \frac{1}{2}$. Starting from $(1,1)$, the factors are $1 - \frac{2}{4} = \frac{1}{2}$ and $1 - \frac{6}{4} = -\frac{1}{2}$, so

$$\theta_k = \left(\left(\tfrac{1}{2}\right)^k,\ \left(-\tfrac{1}{2}\right)^k\right): \quad (1,1),\ (\tfrac{1}{2},-\tfrac{1}{2}),\ (\tfrac{1}{4},\tfrac{1}{4}),\ (\tfrac{1}{8},-\tfrac{1}{8}),\ \ldots$$

The iterates bounce between the lines $y = x$ and $y = -x$, crossing the valley floor ($y = 0$) at every step. The first step has direction $-(1,3)/\sqrt{10}$, the steepest-descent direction of Example 1.5, and it overshoots in y. Consecutive steps make an angle whose cosine is -0.8, about 143°. The distance to the minimum after k steps is $\sqrt{2}\,(\frac{1}{2})^k$, but the path walked has total length $\sum_k \frac{1}{4}\|\nabla f(\theta_k)\| = \sqrt{10}$, more than twice the straight-line distance $\sqrt{2}$.

A narrower valley makes this worse. For $x^2 + 25y^2$, $\mu = 2$, $L = 50$ and $\kappa = 25$. The best step is $\eta = \frac{2}{52} = \frac{1}{26}$, the factors are $\frac{12}{13}$ and $-\frac{12}{13}$, and $\theta_k = ((\frac{12}{13})^k, (-\frac{12}{13})^k)$. Dividing the error by 10^6 takes 173 steps, against 20 for $\kappa = 3$. Figure 5.1 shows both runs.

The zigzag comes from a single step size serving every direction. In the steep direction the step overshoots and the error changes sign; in the shallow direction it moves only part of the way. No fixed η suits both when κ is large. Chapter 6 returns to this problem with momentum, and with methods that choose a different step for each coordinate.

5.8. Gradient flow

Run gradient descent on $x^2 + 3y^2$ from $(1,1)$ with smaller and smaller steps, always for $1/\eta$ steps, so that the total of the step sizes is 1. With

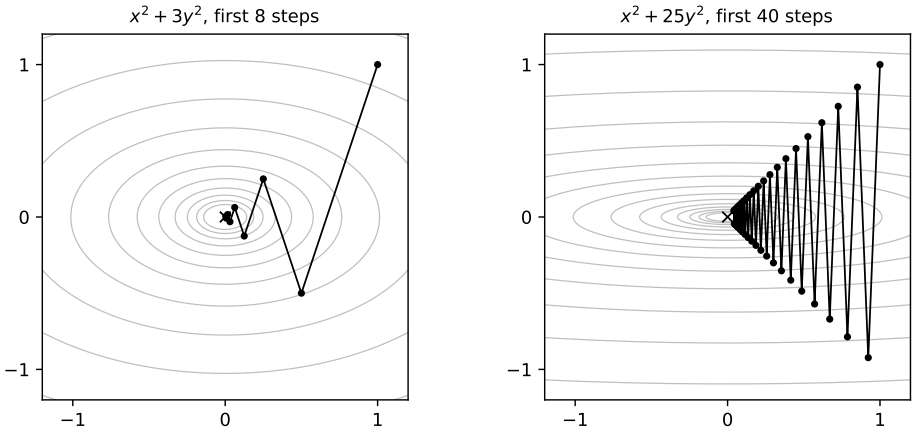

Figure 5.1.: Gradient descent with the best fixed step on $x^2 + 3y^2$ (left, $\kappa = 3$, $\eta = 1/4$) and on $x^2 + 25y^2$ (right, $\kappa = 25$, $\eta = 1/26$), starting from $(1, 1)$. The grey curves are level curves. Both runs cross the valley floor at every step, and the narrower valley needs many more steps.

$\eta = 0.1$ the last point is $(0.8^{10}, 0.4^{10}) \approx (0.1074, 0.000105)$; with $\eta = 0.01$ it is $(0.98^{100}, 0.94^{100}) \approx (0.1326, 0.00205)$; with $\eta = 0.001$ it is $(0.998^{1000}, 0.994^{1000}) \approx (0.1351, 0.00243)$. The points settle down, towards $(e^{-2}, e^{-6}) \approx (0.1353, 0.00248)$. Think of η as a small interval of time, and of θ_k as the position at time $k\eta$. Rearranged, Equation 5.1 reads

$$\frac{\theta_{k+1} - \theta_k}{\eta} = -\nabla f(\theta_k) :$$

the left side is the average velocity over one interval, a difference quotient, so the position changes at rate $-\nabla f$. As $\eta \to 0$ the difference quotient becomes a derivative, and the iterates trace out the solution of a differential equation.

Definition 5.6 (Gradient flow)**.** Let $f \colon \mathbb{R}^n \to \mathbb{R}$ be differentiable. A *gradient flow* of f is a differentiable path $\theta \colon [0, \infty) \to \mathbb{R}^n$ with

$$\frac{d\theta}{dt} = -\nabla f(\theta(t)). \tag{5.6}$$

The path moves at every instant in the direction of steepest descent, with speed equal to the length of the gradient. This is a system of n first-order differential equations, one for each coordinate. When ∇f is Lipschitz continuous, it has exactly one solution for each starting point $\theta(0)$, by the existence and uniqueness theorem for such systems, which the book does not prove (see Stoer and Bulirsch (2002, ch. 7)). The function decreases along the solution.

Theorem 5.8. *Let $f \colon \mathbb{R}^n \to \mathbb{R}$ be differentiable and θ a gradient flow of f. Then*

$$\frac{d}{dt} f(\theta(t)) = -\big\|\nabla f(\theta(t))\big\|^2 \leq 0,$$

so $f(\theta(t))$ never increases, and it strictly decreases on any interval where the gradient is not zero. If f is also μ-strongly convex with minimiser θ^* and $f^* = f(\theta^*)$, then

$$f(\theta(t)) - f^* \leq e^{-2\mu t}\big(f(\theta(0)) - f^*\big).$$

Proof. By the chain rule (Theorem 2.1), the derivative of $t \mapsto f(\theta(t))$ is $\nabla f(\theta(t)) \cdot \theta'(t)$, which is $-\|\nabla f(\theta(t))\|^2$ by Equation 5.6. A function whose derivative is never positive never increases, and one with a negative derivative strictly decreases.

For the second part, let $u(t) = f(\theta(t)) - f^*$. By Lemma 5.5, $u'(t) = -\|\nabla f\|^2 \le -2\mu u(t)$. Multiply by the integrating factor $e^{2\mu t}$:

$$\frac{d}{dt}(e^{2\mu t}u(t)) = e^{2\mu t}(u'(t) + 2\mu u(t)) \le 0,$$

so $e^{2\mu t}u(t) \le u(0)$. □

Euler's method approximates the solution of $\theta' = F(\theta)$ by stepping along the tangent: with step h, $\theta_{k+1} = \theta_k + hF(\theta_k)$, and θ_k approximates $\theta(kh)$. With $F = -\nabla f$ and $h = \eta$ this is Equation 5.1. Gradient descent is Euler's method for the gradient flow, and the learning rate is the time step.

On the quadratic Equation 5.5 the flow can be solved exactly. The error $\mathbf{e}(t) = \theta(t) - \theta^*$ satisfies $\mathbf{e}' = -A\mathbf{e}$, and $\mathbf{e}(t) = \sum_i e^{-\lambda_i t} c_i \mathbf{u}_i$ is the solution, since differentiating multiplies the coefficient of $\mathbf{u}_i$ by $-\lambda_i$. Gradient descent replaces $e^{-\lambda_i \eta k}$, the value at time $t = \eta k$, by $(1 - \eta\lambda_i)^k$. When $\eta\lambda_i$ is small the two are close, because $1 - x \approx e^{-x}$ for small x. When $\eta\lambda_i > 1$ the factor $1 - \eta\lambda_i$ is negative, and Euler's method overshoots a solution that never changes sign; when $\eta\lambda_i \ge 2$ it diverges, although the flow decays. The zigzag of Example 5.8 is this overshoot in the direction with the largest eigenvalue. The condition $\eta < 2/L$ of Theorem 5.7 is the condition $|1 - \eta L| < 1$ under which Euler's method for the equation $y' = -Ly$ produces a sequence that decays, as the solution $y(t) = y(0)e^{-Lt}$ does; numerical analysts call it the *stability* condition for that equation.

Example 5.9. For $x^2 + 3y^2$ from $(1, 1)$, the flow is $\theta(t) = (e^{-2t}, e^{-6t})$, which follows the curve $y = x^3$ into the origin without crossing the x-axis. Gradient descent with $\eta = \frac{1}{4}$ gives $y_k = (-\frac{1}{2})^k$ where the flow has $y(\frac{k}{4}) = e^{-3k/2}$: the flow's y-coordinate stays positive, and the iterates'

changes sign at every step. The smaller steps at the start of this section track the flow, and their last points approach its value (e^{-2}, e^{-6}) at time 1.

5.9. Notes and further reading

Nesterov (2018, ch. 2) develops the theory of smooth convex optimisation in detail. It proves that no method that uses only gradients can beat a rate of order $1/k^2$ on all convex L-smooth functions, and constructs accelerated methods that achieve it, so the $1/k$ rate of Theorem 5.5 is not the best possible. Boyd and Vandenberghe (2004, ch. 3) covers convex functions, including the first-order and second-order conditions, and Boyd and Vandenberghe (2004, sec. 9.3) analyses gradient descent with line searches, with worked examples showing the effect of the condition number. Nocedal and Wright (2006, ch. 3) treats line searches, which choose a step at each iteration instead of fixing η (Exercise 5.3 is the simplest case), and the convergence rate of steepest descent on quadratics. Deisenroth et al. (2020, ch. 7) introduces these methods from the point of view of machine learning.

5.10. Exercises

Exercise 5.1. Let $f(x, y) = x^2 + 3y^2$, $\theta_0 = (1, 1)$ and $\eta > 0$.

(a) For which η does gradient descent converge to $(0, 0)$?

(b) For which η does y_k change sign at every step?

(c) Find θ_3 for $\eta = 0.3$, and describe what happens when $\eta = \frac{1}{3}$.

Exercise 5.2. Let $f(x) = \ln(1 + e^x)$, as in Example 5.3.

(a) Show that no constant smaller than $\frac{1}{4}$ makes f L-smooth.

(b) Run gradient descent with $\eta = 4$ from $x_0 = 0$. Find x_1, and show that x_k decreases at every step and that $x_k \to -\infty$.

Exercise 5.3. *Exact line search* chooses each step length to minimise f along the gradient direction: $\theta_{k+1} = \theta_k - t_k \nabla f(\theta_k)$, where $t_k > 0$ minimises $\varphi(t) = f(\theta_k - t\nabla f(\theta_k))$.

(a) For $f(x, y) = x^2 + 3y^2$ and $\theta_0 = (1, 1)$, find t_0 and θ_1.

(b) Show that for any differentiable f, $\nabla f(\theta_{k+1}) \cdot \nabla f(\theta_k) = 0$: consecutive steps are perpendicular.

Exercise 5.4. Let X be an $m \times n$ matrix, $\mathbf{y} \in \mathbb{R}^m$ and $f(\theta) = \frac{1}{2}\|X\theta - \mathbf{y}\|^2$.

(a) Show that $\nabla f(\theta) = X^\top(X\theta - \mathbf{y})$ and that the Hessian is $X^\top X$.

(b) Show that every eigenvalue of $X^\top X$ is non-negative. Deduce that f is convex and L-smooth, with L the largest eigenvalue of $X^\top X$.

(c) Fitting a line $\theta_1 + \theta_2 s$ to data at $s = 0, 1, 2$ gives $X = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}^\top$. Find the eigenvalues of $X^\top X$ and the condition number of f.

Exercise 5.5. Let f be differentiable and $\mu > 0$. Show that f is μ-strongly convex if and only if $g(\mathbf{x}) = f(\mathbf{x}) - \frac{\mu}{2}\|\mathbf{x}\|^2$ is convex.

Exercise 5.6. Let $f \colon \mathbb{R}^n \to \mathbb{R}$ be L-smooth and μ-strongly convex. Show that $\mu \le L$.

Exercise 5.7. Let $A = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$, $\mathbf{b} = (1, 1)$ and $f(\theta) = \frac{1}{2}\theta^\top A\theta - \mathbf{b}^\top\theta$.

(a) Find the eigenvalues and unit eigenvectors of A, the minimiser θ^*, and the step sizes for which gradient descent converges from every starting point.

(b) Find the best fixed step η and its rate ρ.

(c) With that η and $\theta_0 = \mathbf{0}$, find θ_1 and θ_2. Explain why the error shrinks by exactly ρ at every step.

Exercise 5.8. Let $\lambda > 0$ and $f(\theta) = \frac{\lambda}{2}\theta^2$.

(a) Solve the gradient flow with $\theta(0) = \theta_0$, and find the gradient descent iterates θ_k.

(b) Fix $t > 0$ and take k steps of size $\eta = t/k$. Show that $\theta_k \to \theta(t)$ as $k \to \infty$.

(c) For $\lambda = 6$, $\theta_0 = 1$ and $t = 1$, compare $\theta(1)$ with θ_k for $k = 6$, 60 and 600.

6. Stochastic and adaptive methods

A model learns from its training data (Chapter 5), a list of N examples, and its loss is the average of the losses on the individual examples:

$$f(\theta) = \frac{1}{N} \sum_{i=1}^{N} f_i(\theta), \tag{6.1}$$

where f_i measures the error on example i. Its gradient is the average of the N gradients ∇f_i, and computing it exactly means a pass through all the training data for every step. The model in Chapter 11 trains on a list of names, and instead takes each step using the gradient of the loss on a single name. That gradient is random, because the choice of name is, but it is right on average. The first half of this chapter makes "right on average" precise and shows that gradient descent still works with such estimates.

The second half builds the method the model actually uses. Momentum smooths the zigzag of Example 5.8 by remembering past steps. Exponential moving averages keep a running mean of recent gradients and of their squares, and Adam combines the two to give each parameter its own step size. A learning-rate schedule then shrinks the steps as training goes on.

As in Chapter 5, f is the function being minimised and L is a smoothness constant.

6.1. Probability on a finite set

Suppose the training data have three examples, and at the current parameters their gradients are the numbers -1, -2 and -6. The full gradient is their average, -3. Pick one example at random, each with probability $\frac{1}{3}$, and use its gradient instead. The result is random, but averaging its three possible values with their probabilities as weights gives $\frac{1}{3}(-1) + \frac{1}{3}(-2) + \frac{1}{3}(-6) = -3$, the full gradient. That weighted average is the *expectation* of the random gradient.

Now pick two examples, one after the other, each pick independent of the first, and average their gradients. There are 9 equally likely ordered pairs, but the averages are not equally likely: the value -1.5 comes from the pairs $(1, 2)$ and $(2, 1)$ and has probability $\frac{2}{9}$, while -1 comes only from $(1, 1)$ and has probability $\frac{1}{9}$. So the definitions below allow outcomes of unequal probability. They also allow vector values, since gradients are vectors.

Definition 6.1 (Finite probability space and expectation)**.** A *finite probability space* is a finite set Ω of *outcomes* together with numbers $P(\omega) \geq 0$, one for each $\omega \in \Omega$, that add up to 1. An *event* is a subset $E \subseteq \Omega$, with probability $P(E) = \sum_{\omega \in E} P(\omega)$. A *random variable* is a function $X \colon \Omega \to \mathbb{R}$, and a *random vector* is a function $\mathbf{X} \colon \Omega \to \mathbb{R}^n$. For a possible value $\mathbf{x}$, $P(\mathbf{X} = \mathbf{x})$ is the probability of the event $\{\omega : \mathbf{X}(\omega) = \mathbf{x}\}$. The *expectation* of a random vector (or random variable) $\mathbf{X}$ is

$$\mathbb{E}[\mathbf{X}] = \sum_{\omega \in \Omega} \mathbf{X}(\omega)\, P(\omega).$$

A random variable is a quantity whose value depends on the outcome, such as the gradient of the example picked. Writing it as a function on Ω makes "its value at the outcome ω" precise. The expectation is the average of the values of $\mathbf{X}$, each weighted by the probability of the outcome that produces it; its components are the expectations of the components of $\mathbf{X}$. When

all outcomes are equally likely, $P(\omega) = 1/|\Omega|$, $P(E) = |E|/|\Omega|$ is a matter of counting, and $\mathbb{E}[\mathbf{X}]$ is the ordinary mean of the values $\mathbf{X}(\omega)$. In the example, Ω is the set of 9 ordered pairs, each with $P = \frac{1}{9}$, and the average of the two gradients is a random variable on Ω.

Theorem 6.1 (Linearity of expectation)**.** *For random vectors* $\mathbf{X}$*,* $\mathbf{Y}$ *on the same space, real numbers* a*,* b *and a fixed vector* $\mathbf{v}$*,*

$$\mathbb{E}[a\mathbf{X} + b\mathbf{Y}] = a\,\mathbb{E}[\mathbf{X}] + b\,\mathbb{E}[\mathbf{Y}], \qquad \mathbb{E}[\mathbf{v}] = \mathbf{v}, \qquad \mathbb{E}[\mathbf{v} \cdot \mathbf{X}] = \mathbf{v} \cdot \mathbb{E}[\mathbf{X}].$$

Also $\mathbb{E}[\mathbf{X}] = \sum_{\mathbf{x}} \mathbf{x}\, P(\mathbf{X} = \mathbf{x})$*, the sum running over the values of* $\mathbf{X}$*. If* $X \le Y$ *at every outcome then* $\mathbb{E}[X] \le \mathbb{E}[Y]$*.*

Proof. The first three statements follow from the definition, because sums can be split and constants taken outside. For the fourth, group the outcomes according to the value of $\mathbf{X}$: the outcomes with $\mathbf{X}(\omega) = \mathbf{x}$ contribute $\mathbf{x} \sum_{\mathbf{X}(\omega)=\mathbf{x}} P(\omega) = \mathbf{x}\, P(\mathbf{X} = \mathbf{x})$. The last holds term by term, since $P(\omega) \ge 0$. □

In the example, let X be the average of the two picked gradients. Its six values have unequal probabilities:

value x	-1	-1.5	-2	-3.5	-4	-6
$P(X = x)$	$\frac{1}{9}$	$\frac{2}{9}$	$\frac{1}{9}$	$\frac{2}{9}$	$\frac{2}{9}$	$\frac{1}{9}$

The formula over values gives

$$\mathbb{E}[X] = \frac{-1 - 3 - 2 - 7 - 8 - 6}{9} = -\frac{27}{9} = -3.$$

Linearity gives the same answer at once: $X = \frac{1}{2}(X_1 + X_2)$, where X_1 and X_2 are the gradients of the first and the second pick, each with expectation

-3, so $\mathbb{E}[X] = \frac{1}{2}(-3-3) = -3$. The average of two picks is right on average, just as a single pick is.

Two picks made one after the other, each from all three examples, do not influence each other. In the example, the probability that the first pick is example 1 and the second is example 2 is $\frac{1}{9} = \frac{1}{3} \cdot \frac{1}{3}$, the product of the separate probabilities. That product rule is the definition of independence.

Definition 6.2 (Independence)**.** Random vectors $\mathbf{X}$ and $\mathbf{Y}$ on the same space are *independent* if

$$P(\mathbf{X} = \mathbf{x} \text{ and } \mathbf{Y} = \mathbf{y}) = P(\mathbf{X} = \mathbf{x})\, P(\mathbf{Y} = \mathbf{y})$$

for all values $\mathbf{x}$ and $\mathbf{y}$. Random vectors $\mathbf{X}_1, \ldots, \mathbf{X}_B$ on the same space are *independent*, or *mutually independent*, if

$$P(\mathbf{X}_1 = \mathbf{x}_1, \ldots, \mathbf{X}_B = \mathbf{x}_B) = P(\mathbf{X}_1 = \mathbf{x}_1) \cdots P(\mathbf{X}_B = \mathbf{x}_B)$$

for all values $\mathbf{x}_1, \ldots, \mathbf{x}_B$, where the commas on the left mean "and". They are *pairwise independent* if $\mathbf{X}_i$ and $\mathbf{X}_j$ are independent whenever $i \neq j$.

In words: knowing the value of $\mathbf{X}$ does not change the probabilities of the values of $\mathbf{Y}$. Mutual independence implies pairwise independence: summing the product rule over all the values of the vectors other than $\mathbf{X}_i$ and $\mathbf{X}_j$ leaves the product rule for that pair. Pairwise independence is all that the variance calculations below need.

Lemma 6.1. *If $\mathbf{X}$ and $\mathbf{Y}$ are independent, then so are $g(\mathbf{X})$ and $h(\mathbf{Y})$, for any functions g and h.*

Proof. The event $g(\mathbf{X}) = \mathbf{a}$ and $h(\mathbf{Y}) = \mathbf{b}$ is made up of the events $\mathbf{X} = \mathbf{x}$ and $\mathbf{Y} = \mathbf{y}$ with $g(\mathbf{x}) = \mathbf{a}$ and $h(\mathbf{y}) = \mathbf{b}$, which do not overlap. So

$$P(g(\mathbf{X}) = \mathbf{a},\, h(\mathbf{Y}) = \mathbf{b}) = \sum_{g(\mathbf{x})=\mathbf{a}} \sum_{h(\mathbf{y})=\mathbf{b}} P(\mathbf{X} = \mathbf{x})\, P(\mathbf{Y} = \mathbf{y}) = P\big(g(\mathbf{X}) = \mathbf{a}\big)\, P\big(h(\mathbf{Y}) = \mathbf{b}\big).$$

□

Theorem 6.2. *If* $\mathbf{X}$ *and* $\mathbf{Y}$ *are independent random vectors in* $\mathbb{R}^n$*, then* $\mathbb{E}[\mathbf{X} \cdot \mathbf{Y}] = \mathbb{E}[\mathbf{X}] \cdot \mathbb{E}[\mathbf{Y}]$.

Proof. Group the outcomes by the pair of values $(\mathbf{x}, \mathbf{y})$, as in Theorem 6.1, and use independence:

$$\mathbb{E}[\mathbf{X}\cdot\mathbf{Y}] = \sum_{\mathbf{x},\mathbf{y}} (\mathbf{x}\cdot\mathbf{y})\, P(\mathbf{X}=\mathbf{x})\, P(\mathbf{Y}=\mathbf{y}) = \Big(\sum_{\mathbf{x}} \mathbf{x}\, P(\mathbf{X}=\mathbf{x})\Big)\cdot\Big(\sum_{\mathbf{y}} \mathbf{y}\, P(\mathbf{Y}=\mathbf{y})\Big).$$

□

A random estimate that is right on average can still be far from its mean. The single-pick gradient above takes the values -1, -2 and -6, which differ from the mean -3 by 2, 1 and -3; the average of the squared differences is $\frac{4+1+9}{3} = \frac{14}{3}$. This mean squared deviation measures the spread of the estimate.

Definition 6.3 (Variance)**.** The *variance* of a random vector $\mathbf{X}$ is $\operatorname{Var}[\mathbf{X}] = \mathbb{E}\big[\|\mathbf{X} - \mathbb{E}[\mathbf{X}]\|^2\big]$, the mean squared distance from the mean. For a random variable it is $\mathbb{E}\big[(X - \mathbb{E}[X])^2\big]$. Its square root $\sqrt{\operatorname{Var}[\mathbf{X}]}$ is the *standard deviation*.

The mean absolute deviation $\mathbb{E}\big[\|\mathbf{X} - \mathbb{E}[\mathbf{X}]\|\big]$ would also measure spread, but squares can be expanded, and that makes the variance of a sum of independent random vectors the sum of their variances (Theorem 6.3). The standard deviation has the same units as $\mathbf{X}$; for the single pick above it is $\sqrt{14/3} \approx 2.16$.

Lemma 6.2. *For a random vector* $\mathbf{X}$*, a real number* c *and a fixed vector* $\mathbf{v}$*,*

$$\operatorname{Var}[\mathbf{X}] = \mathbb{E}\big[\|\mathbf{X}\|^2\big] - \big\|\mathbb{E}[\mathbf{X}]\big\|^2, \qquad \operatorname{Var}[c\mathbf{X} + \mathbf{v}] = c^2 \operatorname{Var}[\mathbf{X}].$$

Proof. Write $\mathbf{m} = \mathbb{E}[\mathbf{X}]$. Expanding and using Theorem 6.1, $\mathbb{E}[\|\mathbf{X}\|^2 - 2\mathbf{m}\cdot\mathbf{X} + \|\mathbf{m}\|^2] = \mathbb{E}[\|\mathbf{X}\|^2] - 2\|\mathbf{m}\|^2 + \|\mathbf{m}\|^2$. For the second formula, $c\mathbf{X} + \mathbf{v} - \mathbb{E}[c\mathbf{X} + \mathbf{v}] = c(\mathbf{X} - \mathbf{m})$. □

Theorem 6.3. *If $\mathbf{X}_1, \dots, \mathbf{X}_B$ are pairwise independent random vectors in $\mathbb{R}^n$ on the same finite probability space, then*

$$\operatorname{Var}[\mathbf{X}_1 + \cdots + \mathbf{X}_B] = \operatorname{Var}[\mathbf{X}_1] + \cdots + \operatorname{Var}[\mathbf{X}_B].$$

Proof. Let $\mathbf{Y}_j = \mathbf{X}_j - \mathbb{E}[\mathbf{X}_j]$, so $\mathbb{E}[\mathbf{Y}_j] = \mathbf{0}$, and the sum minus its mean is $\sum_j \mathbf{Y}_j$. Expanding the square,

$$\Big\|\sum_j \mathbf{Y}_j\Big\|^2 = \sum_j \|\mathbf{Y}_j\|^2 + \sum_{i\neq j} \mathbf{Y}_i \cdot \mathbf{Y}_j.$$

For $i \neq j$, $\mathbf{Y}_i$ and $\mathbf{Y}_j$ are independent by Lemma 6.1, so by Theorem 6.2 $\mathbb{E}[\mathbf{Y}_i \cdot \mathbf{Y}_j] = \mathbf{0}\cdot\mathbf{0} = 0$. Taking expectations leaves $\sum_j \mathbb{E}[\|\mathbf{Y}_j\|^2]$. □

Example 6.1. Let X be the score of a fair die. Then $\mathbb{E}[X] = \frac{1+\cdots+6}{6} = \frac{7}{2}$ and $\mathbb{E}[X^2] = \frac{1+4+\cdots+36}{6} = \frac{91}{6}$, so $\operatorname{Var}[X] = \frac{91}{6} - \frac{49}{4} = \frac{35}{12}$ by Lemma 6.2. The mean of B independent throws has variance $\frac{35}{12B}$, by Theorem 6.3 and Lemma 6.2 with $c = 1/B$: averaging shrinks the spread.

6.2. Minibatch gradients

Return to the average loss Equation 6.1. By linearity of the gradient, $\nabla f = \frac{1}{N}\sum_i \nabla f_i$. Choosing an example *uniformly at random*, each with probability $1/N$, and using its gradient gives an estimate of ∇f; averaging the gradients of several random examples gives a better one. In the example that opened Section 6.1, a single pick has variance $\frac{14}{3}$, and the average

of two independent picks takes nine equally likely values whose variance turns out to be $\frac{7}{3}$, half as much (Example 6.2 lists them). Write

$$\sigma^2(\theta) = \frac{1}{N}\sum_{i=1}^{N}\|\nabla f_i(\theta) - \nabla f(\theta)\|^2$$

for the spread of the individual gradients around their mean: the variance of the gradient of one uniformly chosen example. A random list of examples used for one gradient estimate is a *minibatch*, and the average of their gradients is the *minibatch gradient*.

Theorem 6.4 (Minibatch gradients)**.** *Let $f = \frac{1}{N}\sum_{i=1}^{N} f_i$ with each $f_i\colon \mathbb{R}^n \to \mathbb{R}$ differentiable, fix θ, and let $\sigma^2(\theta)$ be the spread defined above. Fix a* batch size *$B \geq 1$. Let the outcomes be the N^B lists $(i_1, \dots, i_B)$ of examples, all equally likely, let I_j be the jth entry of the list, and let*

$$\mathbf{G} = \frac{1}{B}\sum_{j=1}^{B}\nabla f_{I_j}(\theta).$$

Then $\mathbb{E}[\mathbf{G}] = \nabla f(\theta)$ and $\operatorname{Var}[\mathbf{G}] = \sigma^2(\theta)/B$.

Proof. For each j and each example a, the lists with $i_j = a$ number N^{B-1}, so $P(I_j = a) = N^{B-1}/N^B = 1/N$: each entry is a uniformly chosen example. So by Theorem 6.1, $\mathbb{E}[\nabla f_{I_j}(\theta)] = \frac{1}{N}\sum_a \nabla f_a(\theta) = \nabla f(\theta)$, and its variance is $\frac{1}{N}\sum_a \|\nabla f_a - \nabla f\|^2 = \sigma^2$. Averaging B of these gives $\mathbb{E}[\mathbf{G}] = \nabla f(\theta)$.

For $i \neq j$, the lists with $i_i = a$ and $i_j = b$ number N^{B-2}, so $P(I_i = a \text{ and } I_j = b) = 1/N^2 = P(I_i = a)\,P(I_j = b)$. The entries are pairwise independent, and so are the gradients $\nabla f_{I_j}(\theta)$, by Lemma 6.1. By Theorem 6.3 the sum of the B gradients has variance $B\sigma^2$, and by Lemma 6.2 dividing by B divides the variance by B^2. □

The *bias* of a random estimate is its expectation minus the quantity it estimates, and an estimate with bias zero is *unbiased*. So the minibatch

gradient is an unbiased estimate of ∇f. A batch of B examples costs B gradient computations and cuts the variance by a factor of B, so the typical error, measured by the standard deviation, falls only like $1/\sqrt{B}$.

Example 6.2. Take $N = 3$ examples with targets $y = 1, 2, 6$, one parameter θ, and $f_i(\theta) = \frac{1}{2}(\theta - y_i)^2$. Then $f(\theta) = \frac{1}{2}(\theta - 3)^2 + \frac{7}{3}$, with minimum $f^* = \frac{7}{3}$ at $\theta = 3$, the mean of the targets. The individual gradients are $f_i'(\theta) = \theta - y_i$, and they differ from $f'(\theta) = \theta - 3$ by $3 - y_i = 2, 1, -3$, so $\sigma^2 = \frac{4+1+9}{3} = \frac{14}{3}$ at every θ.

At $\theta = 0$, a batch of $B = 2$ is one of 9 equally likely ordered pairs, and gives the estimate $-(y_a + y_b)/2$:

pair	(1,1)	(1,2)	(1,6)	(2,1)	(2,2)	(2,6)	(6,1)	(6,2)	(6,6)
G	-1	-1.5	-3.5	-1.5	-2	-4	-3.5	-4	-6

The mean of the nine values is $-27/9 = -3 = f'(0)$, and their variance is $21/9 = \frac{7}{3} = \sigma^2/2$, as Theorem 6.4 says.

The model in Chapter 11 uses batches of size 1: at step k it takes the next name from a list that was shuffled once at the start. Every order of the list is equally likely, so the name in any given position is equally likely to be each of the N names, and the gradient at each step is unbiased. Consecutive names are drawn without replacement, which makes them slightly dependent; Exercise 6.2 shows that this only lowers the variance of a batch.

6.3. Stochastic gradient descent

Return to Example 6.2 at $\theta = 0$, where $f'(0) = -3$. A gradient descent step with $\eta = 0.1$ goes to 0.3. A step that uses the gradient of one randomly

chosen example instead goes to 0.1, 0.2 or 0.6, each with probability $\frac{1}{3}$; on average it goes to 0.3, the same place. Repeating such random steps is stochastic gradient descent, "stochastic" being the standard word for random. Each step is a fresh random choice, independent of the earlier ones, so the definition has to describe a sequence of draws.

Definition 6.4 (Stochastic gradient descent)**.** Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be differentiable. A *stochastic gradient* for f is a finite set Ξ with probabilities $p(\xi) > 0$ adding up to 1, and a function $\mathbf{g}(\theta, \xi) \in \mathbb{R}^n$, such that for every θ

$$\sum_{\xi \in \Xi} p(\xi)\, \mathbf{g}(\theta, \xi) = \nabla f(\theta).$$

Stochastic gradient descent (SGD) with learning rates $\eta_0, \eta_1, \dots > 0$ starts from a fixed θ_0 and sets

$$\theta_{k+1} = \theta_k - \eta_k\, \mathbf{g}(\theta_k, \xi_k),$$

where $\xi_0, \xi_1, \dots$ are drawn independently with probabilities p. For K steps the outcomes are the lists $(\xi_0, \dots, \xi_{K-1})$, with $P(\xi_0, \dots, \xi_{K-1}) = p(\xi_0) \cdots p(\xi_{K-1})$.

The condition says that at every point θ, $\mathbf{g}(\theta, \xi)$ is an unbiased estimate of $\nabla f(\theta)$: ξ records the random choice, such as which example was picked, and $\mathbf{g}$ is the gradient that choice produces. Minibatches fit this definition: Ξ is the set of lists of B examples, all equally likely, and $\mathbf{g}$ is the batch average of Theorem 6.4. The iterate θ_k is a random vector that depends only on the first k draws $\xi_0, \dots, \xi_{k-1}$. The proof below averages over one draw at a time, using the following lemma.

Lemma 6.3. *Let Ξ be a finite set with probabilities $p(\xi) > 0$ adding up to 1, let $K \geq 1$, and give each list $(\xi_0, \dots, \xi_{K-1})$ of elements of Ξ the probability $p(\xi_0) \cdots p(\xi_{K-1})$, as in Definition 6.4. For $0 \leq k < K$, let F be a real function of the first $k+1$ entries $\xi_0, \dots, \xi_k$, and let*

$$\bar{F}(\xi_0, \dots, \xi_{k-1}) = \sum_{\xi \in \Xi} p(\xi)\, F(\xi_0, \dots, \xi_{k-1}, \xi)$$

be its average over the last of them. Then $\mathbb{E}[F] = \mathbb{E}[\bar{F}]$.

Proof. By definition, $\mathbb{E}[F] = \sum p(\xi_0)\cdots p(\xi_{K-1})\, F(\xi_0, \dots, \xi_k)$, the sum running over all lists. Sum first over ξ_{K-1}, then ξ_{K-2}, and so on down to ξ_{k+1}: F does not depend on them, and each $\sum_\xi p(\xi)$ is 1. What remains is

$$\sum_{\xi_0,\dots,\xi_{k-1}} p(\xi_0)\cdots p(\xi_{k-1}) \sum_{\xi_k} p(\xi_k)\, F(\xi_0, \dots, \xi_k) = \sum_{\xi_0,\dots,\xi_{k-1}} p(\xi_0)\cdots p(\xi_{k-1})\, \bar{F}(\xi_0, \dots, \xi_{k-1}),$$

which is $\mathbb{E}[\bar{F}]$ by the same argument. □

Theorem 6.5 (Convergence of SGD)**.** *Let* $f\colon \mathbb{R}^n \to \mathbb{R}$ *be* L*-smooth and bounded below, with* $f^* = \inf f$*, and let* $\mathbf{g}$ *be a stochastic gradient for* f *whose variance is at most* σ^2 *at every point:*

$$\sum_\xi p(\xi) \left\|\mathbf{g}(\theta, \xi) - \nabla f(\theta)\right\|^2 \le \sigma^2 \quad \text{for all } \theta.$$

Run SGD with a constant learning rate $0 < \eta \le 1/L$*. Then for every* $K \ge 1$*,*

$$\frac{1}{K}\sum_{k=0}^{K-1} \mathbb{E}[\|\nabla f(\theta_k)\|^2] \le \frac{2(f(\theta_0) - f^*)}{\eta K} + \eta L \sigma^2.$$

Proof. Fix k and the first k draws, so that θ_k is a fixed vector. Write $\mathbf{d} = \nabla f(\theta_k)$ and $\mathbf{G}(\xi) = \mathbf{g}(\theta_k, \xi)$. By Lemma 5.4 with $\mathbf{y} = \theta_k - \eta\mathbf{G}(\xi)$,

$$f(\theta_k - \eta\mathbf{G}(\xi)) \le f(\theta_k) - \eta\, \mathbf{d}\cdot\mathbf{G}(\xi) + \frac{L\eta^2}{2}\|\mathbf{G}(\xi)\|^2.$$

Average over ξ with weights $p(\xi)$. The average of $\mathbf{G}(\xi)$ is $\mathbf{d}$, and by Lemma 6.2 on the space Ξ the average of $\|\mathbf{G}(\xi)\|^2$ is $\|\mathbf{d}\|^2$ plus the variance, at most $\|\mathbf{d}\|^2 + \sigma^2$. So the average of $f(\theta_{k+1})$ over the draw ξ_k is at most

$$f(\theta_k) - \eta\Big(1 - \frac{L\eta}{2}\Big)\|\mathbf{d}\|^2 + \frac{L\eta^2\sigma^2}{2} \le f(\theta_k) - \frac{\eta}{2}\|\nabla f(\theta_k)\|^2 + \frac{L\eta^2\sigma^2}{2},$$

using $\eta \le 1/L$. This holds for every choice of the first k draws. Take expectations: by Lemma 6.3 and Theorem 6.1,

$$\mathbb{E}[f(\theta_{k+1})] \le \mathbb{E}[f(\theta_k)] - \frac{\eta}{2}\mathbb{E}[\|\nabla f(\theta_k)\|^2] + \frac{L\eta^2\sigma^2}{2}. \tag{6.2}$$

Add these inequalities for $k = 0, \dots, K-1$. The terms $\mathbb{E}[f(\theta_k)]$ telescope, and $\mathbb{E}[f(\theta_K)] \ge f^*$, so

$$\frac{\eta}{2}\sum_{k=0}^{K-1}\mathbb{E}[\|\nabla f(\theta_k)\|^2] \le f(\theta_0) - f^* + \frac{KL\eta^2\sigma^2}{2}.$$

Divide by $\eta K/2$. □

With $\sigma = 0$ this is Theorem 5.2. The noise adds the term $\eta L\sigma^2$, which does not shrink as K grows: with a fixed learning rate the random steps never die away, and the bound only says that the average squared gradient ends up of order $\eta L\sigma^2$, a level called the *noise floor*. A smaller η lowers the noise floor but slows the first term. Given a target $\varepsilon > 0$, the choice $\eta = \min(1/L, \varepsilon/(2L\sigma^2))$ makes the second term at most $\varepsilon/2$, and then $K \ge 4(f(\theta_0) - f^*)/(\eta\varepsilon)$ steps make the first term at most $\varepsilon/2$ too. When σ is large this needs a number of steps of order $1/\varepsilon^2$, against $1/\varepsilon$ for full gradients, but each step costs B example gradients instead of N.

Example 6.3. Run SGD with batch size 1 on Example 6.2: $L = 1$, $\sigma^2 = \frac{14}{3}$, $\theta_0 = 0$ and $\eta = 0.1$, so $f(\theta_0) - f^* = \frac{9}{2}$. For $K = 100$ steps the bound of Theorem 6.5 is

$$\frac{2 \cdot \frac{9}{2}}{0.1 \cdot 100} + 0.1 \cdot \frac{14}{3} = 0.9 + 0.467 \approx 1.37.$$

Here $\|\nabla f(\theta_k)\|^2 = (\theta_k - 3)^2$, and Exercise 6.3 computes its expectation exactly: the average over the first 100 steps is 0.706, and as $k \to \infty$ the expectation tends to $\eta\sigma^2/(2-\eta) \approx 0.246$, which is about half of the term $\eta L\sigma^2 \approx 0.467$ in the bound.

6.4. Momentum

In Example 5.8 consecutive steps point in nearly opposite directions across the valley and in the same direction along it. Adding a fraction of the previous step to the current one cancels part of the first and builds up the second. With $\eta = \frac{1}{4}$ on $x^2 + 3y^2$, the first two steps from $(1, 1)$ are $(-0.5, -1.5)$ and $(-0.25, 0.75)$. Adding half of the first to the second gives $(-0.25, 0.75) + \frac{1}{2}(-0.5, -1.5) = (-0.5, 0)$: the movement across the valley cancels, and the movement along it doubles. The previous step plays the part of a velocity, which is why the method below is named after a heavy ball rolling downhill.

Definition 6.5 (Heavy-ball momentum)**.** Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be differentiable, $\eta > 0$ and $0 \le \beta < 1$. *Gradient descent with momentum* (the *heavy-ball method*) sets $\theta_{-1} = \theta_0$ and

$$\theta_{k+1} = \theta_k - \eta \nabla f(\theta_k) + \beta\,(\theta_k - \theta_{k-1}), \qquad k = 0, 1, 2, \dots \tag{6.3}$$

The step $\theta_{k+1} - \theta_k$ is the gradient step $-\eta \nabla f(\theta_k)$ plus β times the previous step; the number β is the *momentum coefficient*. Setting $\theta_{-1} = \theta_0$ makes the first step a plain gradient step, and with $\beta = 0$ every step is. Each new point depends on the two before it, so on a quadratic the iteration is a second-order linear recurrence, and the usual method for those, through the characteristic equation, applies. Start with one variable: $f(\theta) = \frac{\lambda}{2}\theta^2$ with $\lambda > 0$, so $f'(\theta) = \lambda\theta$ and Equation 6.3 becomes

$$\theta_{k+1} - (1 + \beta - \eta\lambda)\,\theta_k + \beta\,\theta_{k-1} = 0. \tag{6.4}$$

Its characteristic equation is

$$p(r) = r^2 - s\,r + \beta = 0, \qquad s = 1 + \beta - \eta\lambda.$$

Recall the form of the solutions. If the roots $r_1 \ne r_2$ are real, $\theta_k = Ar_1^k + Br_2^k$. If there is a repeated root r, $\theta_k = (A + Bk)r^k$. If the roots are

complex, they are $\rho e^{\pm i\phi}$ and $\theta_k = \rho^k(A\cos k\phi + B\sin k\phi)$. The constants A and B come from θ_{-1} and θ_0. In every case the size of θ_k is governed by the largest modulus of a root: if it is less than 1 then $\theta_k \to 0$, because $k|r|^k \to 0$ when $|r| < 1$.

Theorem 6.6 (Momentum on a one-dimensional quadratic)**.** *Let $\lambda > 0$, $\eta > 0$ and $0 \le \beta < 1$.*

1. *Every solution of Equation 6.4 tends to 0 if and only if $\eta\lambda < 2(1+\beta)$.*
2. *If $(1-\sqrt{\beta})^2 \le \eta\lambda \le (1+\sqrt{\beta})^2$, both roots have modulus $\sqrt{\beta}$, and every solution satisfies $|\theta_k| \le (C+Dk)\,\beta^{k/2}$ for constants C, D that depend on θ_{-1} and θ_0.*

Proof. The product of the roots is β and their sum is s. Also $p(1) = \eta\lambda > 0$ and $p(-1) = 2(1+\beta) - \eta\lambda$.

(2) The roots are not real and distinct exactly when $s^2 \le 4\beta$, that is, $|1+\beta-\eta\lambda| \le 2\sqrt{\beta}$, which rearranges to $(1-\sqrt{\beta})^2 \le \eta\lambda \le (1+\sqrt{\beta})^2$. Complex roots are conjugate, r and $\bar{r}$, so $|r|^2 = r\bar{r} = \beta$. A repeated root is $s/2 = \pm\sqrt{\beta}$. In both cases the forms above give the bound, with $C = |A|$ and $D = |B|$ for a repeated root, and $C = |A| + |B|$, $D = 0$ for complex roots.

(3) Suppose $\eta\lambda < 2(1+\beta)$. If the roots are complex, they have modulus $\sqrt{\beta} < 1$. If they are real, $r_1 \le r_2$, then p is negative or zero exactly on $[r_1, r_2]$, and since $p(1) > 0$ and $p(-1) > 0$, neither 1 nor -1 lies in that interval. So both roots lie in one of $(-\infty, -1)$, $(-1, 1)$ and $(1, \infty)$. In the first and last cases $r_1 r_2 > 1$, but $r_1 r_2 = \beta < 1$. So both roots lie in $(-1, 1)$, and every solution tends to 0.

Conversely, suppose $\eta\lambda \ge 2(1+\beta)$. Then $p(-1) \le 0$, and since $p(r) \to \infty$ as $r \to -\infty$, p has a real root $r \le -1$. The sequence $\theta_k = r^k$ (with $\theta_{-1} = 1/r$, $\theta_0 = 1$) is a solution with $|\theta_k| \ge 1$ for every k. □

With $\beta = 0$ the condition is $\eta\lambda < 2$, as for gradient descent. Momentum allows steps up to twice as long before the iteration diverges, and inside the band of part 2 its rate $\sqrt{\beta}$ does not depend on λ at all. That is what helps on a function with many curvatures at once.

Theorem 6.7 (Tuned momentum on a quadratic)**.** *Let A be a real symmetric $n \times n$ matrix with all its eigenvalues in $[\mu, L]$, where $0 < \mu \le L$, and orthonormal eigenvectors $\mathbf{u}_1, \dots, \mathbf{u}_n$. Let $\mathbf{b} \in \mathbb{R}^n$, let f be the quadratic Equation 5.5, with minimiser $\theta^* = A^{-1}\mathbf{b}$, and let $\kappa = L/\mu$. Run Equation 6.3 with*

$$\eta = \frac{4}{(\sqrt{L} + \sqrt{\mu})^2}, \qquad \beta = \Big(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\Big)^2.$$

Then each coefficient $c_{k,i} = \mathbf{u}_i \cdot (\theta_k - \theta^*)$ of the error satisfies $|c_{k,i}| \le (C_i + D_i k)\,\rho^k$, for constants C_i and D_i that depend on θ_0, with

$$\rho = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} < \frac{\kappa - 1}{\kappa + 1} \quad \text{when } \kappa > 1.$$

Proof. Let $\mathbf{e}_k = \theta_k - \theta^*$. Since $\nabla f(\theta_k) = A\mathbf{e}_k$, subtracting θ^* from Equation 6.3 gives $\mathbf{e}_{k+1} = \mathbf{e}_k - \eta A\mathbf{e}_k + \beta(\mathbf{e}_k - \mathbf{e}_{k-1})$. Taking the dot product with $\mathbf{u}_i$, and using $\mathbf{u}_i \cdot A\mathbf{e}_k = (A\mathbf{u}_i) \cdot \mathbf{e}_k = \lambda_i c_{k,i}$ (as A is symmetric), each $c_{k,i}$ satisfies Equation 6.4 with $\lambda = \lambda_i$. Now $1 - \sqrt{\beta} = 2/(\sqrt{\kappa} + 1)$ and $1 + \sqrt{\beta} = 2\sqrt{\kappa}/(\sqrt{\kappa} + 1)$, so

$$(1 - \sqrt{\beta})^2 = \frac{4}{(\sqrt{\kappa} + 1)^2} = \eta\mu \le \eta\lambda_i \le \eta L = \frac{4\kappa}{(\sqrt{\kappa} + 1)^2} = (1 + \sqrt{\beta})^2,$$

and part 2 of Theorem 6.6 applies with $\sqrt{\beta} = \rho$. For the comparison, $(\sqrt{\kappa} + 1)^2 = \kappa + 1 + 2\sqrt{\kappa} > \kappa + 1$, so

$$\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} = \frac{\kappa - 1}{(\sqrt{\kappa} + 1)^2} < \frac{\kappa - 1}{\kappa + 1}.$$

□

For $\kappa = 25$, gradient descent at its best step shrinks the error by $\frac{12}{13} \approx 0.923$ per step and momentum by $\frac{4}{6} \approx 0.667$; dividing the error by 10^6 takes about 173 steps against about 35. For large κ the number of steps grows like $\sqrt{\kappa}$ instead of κ.

Example 6.4. For $f(x, y) = x^2 + 3y^2$, $\mu = 2$, $L = 6$ and $\kappa = 3$, the parameters of Theorem 6.7 are

$$\eta = \frac{4}{(\sqrt{6}+\sqrt{2})^2} = \frac{1}{2+\sqrt{3}} = 2-\sqrt{3} \approx 0.268, \qquad \beta = (2-\sqrt{3})^2 \approx 0.0718,$$

and $\rho = 2 - \sqrt{3} \approx 0.268$, against $\frac{1}{2}$ for gradient descent. Both eigenvalues sit at the ends of the band, so both characteristic equations have a repeated root: $2-\sqrt{3}$ for the x-coordinate ($\lambda = 2$) and $-(2-\sqrt{3})$ for the y-coordinate ($\lambda = 6$). Solving for the constants from $\theta_{-1} = \theta_0 = 1$ gives

$$x_k = (1 + (\sqrt{3}-1)k)(2-\sqrt{3})^k, \qquad y_k = (1 + (3-\sqrt{3})k)(-(2-\sqrt{3}))^k.$$

The y-coordinate still changes sign at every step, but the zigzag dies away faster: after 10 steps the distance to the minimum is about 3.05×10^{-5}, against $\sqrt{2} \cdot 2^{-10} \approx 1.38 \times 10^{-3}$ for gradient descent with $\eta = \frac{1}{4}$. Figure 6.1 compares the two methods on this function and on $x^2 + 25y^2$.

These parameters need μ and L, which are rarely known, and the guarantee is for quadratics only. Lessard et al. (2016, sec. 4.6) construct a one-dimensional function, μ-strongly convex and L-smooth with $\kappa = 25$, on which the heavy-ball method with exactly these parameters never converges from some starting points: its iterates settle into a cycle. In practice β is chosen by experiment, often close to 1.

6.5. Exponential moving averages

Momentum keeps a memory of past gradients. Unrolling the velocity form of Equation 6.3 (Exercise 6.4) shows that the step is a weighted sum of all

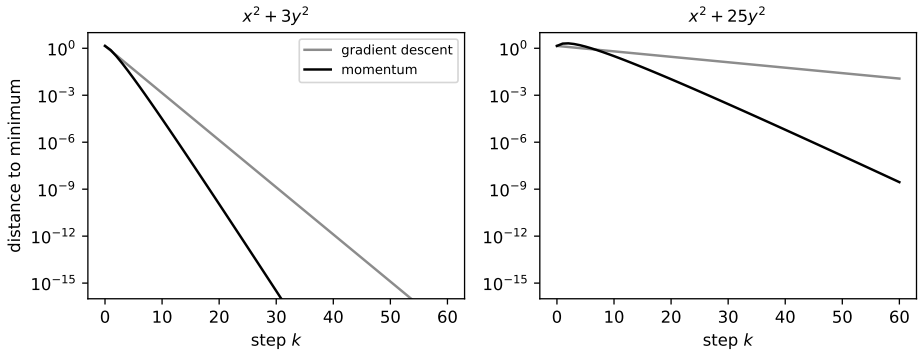

Figure 6.1.: Distance to the minimum against the step number, on a logarithmic scale, for gradient descent with its best fixed step and for momentum with the parameters of Theorem 6.7, starting from $(1, 1)$. Left: $x^2 + 3y^2$ ($\kappa = 3$). Right: $x^2 + 25y^2$ ($\kappa = 25$). The distance under momentum can rise at first, because of the factor $C + Dk$ in Theorem 6.6.

past gradients, with weights that decay geometrically. Adam makes that average explicit.

Suppose the gradient of one parameter is 4 at every step, and keep a running average that moves half way towards each new value, starting from 0. It goes 2, 3, 3.5, 3.75: it approaches 4 but lags behind, because the starting value 0 still has weight. The shortfall is predictable. After k steps the average is $(1 - 0.5^k) \cdot 4$, and dividing by $1 - 0.5^k$ recovers 4 exactly.

Definition 6.6 (Exponential moving average)**.** Let $0 \le \beta < 1$ and let $g_1, g_2, \ldots$ be numbers (or vectors). Their *exponential moving average* is

$$m_0 = 0, \qquad m_k = \beta\, m_{k-1} + (1 - \beta)\, g_k \quad (k \ge 1).$$

Each new value moves the average a fraction $1 - \beta$ of the way towards the latest g_k. With β close to 1 the average changes slowly and remembers many past values.

Theorem 6.8 (Closed form)**.** *For every $k \ge 0$,*

$$m_k = (1 - \beta) \sum_{j=1}^{k} \beta^{k-j} g_j.$$

Proof. By induction on k. For $k = 0$ both sides are 0. If the formula holds for $k - 1$, then

$$m_k = \beta(1 - \beta) \sum_{j=1}^{k-1} \beta^{k-1-j} g_j + (1 - \beta) g_k = (1 - \beta) \Big(\sum_{j=1}^{k-1} \beta^{k-j} g_j + \beta^0 g_k \Big).$$

□

So g_k has weight $1 - \beta$, g_{k-1} has weight $(1 - \beta)\beta$, and each older value has β times the weight of the next. Starting from $m_0 = 0$ makes the average too small at first, and the next theorem measures by how much.

Theorem 6.9 (Bias of the moving average)**.** *Let $k \geq 1$.*

1. *The weights in Theorem 6.8 add up to $1 - \beta^k$.*
2. *If $g_j = g$ for every j, then $m_k = (1 - \beta^k)\, g$.*
3. *The* bias-corrected *average $\widehat{m}_k = m_k/(1 - \beta^k)$ is a weighted average of $g_1, \dots, g_k$ with weights that add up to 1. If the g_j are random variables on a finite probability space with $\mathbb{E}[g_j] = g$ for every j, then $\mathbb{E}[\widehat{m}_k] = g$.*

Proof.

(1) $(1-\beta)\sum_{j=1}^{k} \beta^{k-j} = (1-\beta)(1+\beta+\cdots+\beta^{k-1}) = 1-\beta^k$, the sum of a geometric series. (2) follows from (1) and Theorem 6.8. (3) Dividing the weights by their sum makes them add up to 1. By Theorem 6.1, $\mathbb{E}[\widehat{m}_k] = \frac{1-\beta}{1-\beta^k} \sum_j \beta^{k-j}\, \mathbb{E}[g_j] = g$ by (1).

□

In the language of Section 6.2: if every g_j has expectation g, then $\mathbb{E}[m_k] = (1 - \beta^k)g$, so the uncorrected average has bias $-\beta^k g$. The bias shrinks as k grows, but slowly when β is close to 1, and dividing by $1 - \beta^k$ removes it at every step.

Example 6.5. With $\beta = 0.85$, the first average is $m_1 = 0.15\, g_1$, and the correction divides by $1 - 0.85 = 0.15$ to recover g_1. With $\beta = 0.99$ the shortfall lasts much longer: after 10 constant values, $m_{10} = (1-0.99^{10})g \approx 0.0956\, g$, less than a tenth of the true value. The weights halve every $\ln 2 / \ln(1/\beta)$ steps, which is about 4.3 steps for $\beta = 0.85$ and 69 steps for $\beta = 0.99$.

6.6. Adam

Adam (Kingma and Ba 2014) keeps two exponential moving averages for every parameter: one of its gradient and one of the square of its gradient. It divides the first by the square root of the second, so that each parameter's step is measured against the typical size of its own gradient. (The name abbreviates "adaptive moment estimation": the expectations of a random variable and of its square are its first two *moments*.)

At $(1, 1)$ on $x^2 + 3y^2$ the gradient is $(2, 6)$. Gradient descent moves the second coordinate three times as far as the first, which is what produced the zigzag. Dividing each component by its own size gives $(1, 1)$, and a step against that points from $(1, 1)$ straight at the minimum. Adam does this division with averages: each component of the gradient is divided by the *root mean square* of its recent values, the square root of the mean of their squares. The root mean square of 3 and -4, for example, is $\sqrt{(9 + 16)/2} \approx 3.54$, and that of 2 and -2 is 2.

Definition 6.7 (Adam)**.** Let $0 \le \beta_1 < 1$, $0 \le \beta_2 < 1$, $\epsilon > 0$, and learning rates $\eta_1, \eta_2, \cdots > 0$. Start from θ_0 with $\mathbf{m}_0 = \mathbf{v}_0 = \mathbf{0}$. For $k = 1, 2, ...$, let $\mathbf{g}_k$ be a gradient (or stochastic gradient) at θ_{k-1}, and for every coordinate i set

$$m_{k,i} = \beta_1 m_{k-1,i} + (1 - \beta_1)\, g_{k,i}, \quad \hat{m}_{k,i} = \frac{m_{k,i}}{1 - \beta_1^k},$$

$$v_{k,i} = \beta_2 v_{k-1,i} + (1 - \beta_2)\, g_{k,i}^2, \qquad \hat{v}_{k,i} = \frac{v_{k,i}}{1 - \beta_2^k},$$

$$\theta_{k,i} = \theta_{k-1,i} - \eta_k \frac{\hat{m}_{k,i}}{\sqrt{\hat{v}_{k,i}} + \epsilon}.$$

In words: $\mathbf{m}_k$ is the exponential moving average of the gradients and $\mathbf{v}_k$ that of their squared components, the hats mark the bias corrections of Theorem 6.9, and each coordinate moves by $-\eta_k$ times the ratio of the corrected mean to the corrected root mean square. The small number ϵ

keeps the denominator positive when a coordinate's gradients have all been zero.

The model of Chapter 11 uses $\beta_1 = 0.85$, $\beta_2 = 0.99$, $\epsilon = 10^{-8}$, and 1000 steps with learning rate $\eta_k = 0.01(1 - (k-1)/1000)$, which decreases linearly from 0.01 at the first step to 10^{-5} at the last. Each step uses the gradient of the loss on one name.

By Theorem 6.9, $\hat{\mathbf{m}}_k$ and $\hat{\mathbf{v}}_k$ are weighted averages of the recent gradients and squared gradients, with weights adding up to 1. Without the corrections both would be too small at the start, by different factors: the uncorrected ratio $m_{k,i}/\sqrt{v_{k,i}}$ is the corrected one times $(1-\beta_1^k)/\sqrt{1-\beta_2^k}$, which for the model's values is $0.15/0.1 = 1.5$ at the first step and drifts towards 1 later. In each coordinate the ratio $\hat{m}_{k,i}/\sqrt{\hat{v}_{k,i}}$ is a mean of recent gradients divided by their root mean square. It has no units: multiplying every gradient by $c > 0$ leaves it unchanged (Exercise 6.7). If a coordinate's gradient is steady, the ratio is close to ± 1 and the step is close to η_k in size whatever the gradient's scale. If the gradient keeps changing sign, the mean is small compared with the root mean square and the step is small. Theorem 1.1 bounds the ratio in every case.

Theorem 6.10 (Size of an Adam step)**.** *Run Adam (Definition 6.7) with $0 \le \beta_1 < 1$, $0 < \beta_2 < 1$, $\beta_1^2 < \beta_2$, $\epsilon > 0$ and learning rates $\eta_k > 0$, on any vectors $\mathbf{g}_1, \mathbf{g}_2, \ldots$ in place of the gradients. Then for every $k \ge 1$ and every coordinate i,*

$$|\theta_{k,i} - \theta_{k-1,i}| \le \eta_k \, \frac{1-\beta_1}{\sqrt{(1-\beta_2)(1-\beta_1^2/\beta_2)}}.$$

Proof. Fix i and drop it from the notation. If $g_1 = \cdots = g_k = 0$ the step is 0. Otherwise $v_k > 0$. Put $\gamma = \beta_1^2/\beta_2 < 1$. By Theorem 6.8, and writing $\beta_1^{k-j} g_j = \gamma^{(k-j)/2} \cdot \beta_2^{(k-j)/2} g_j$, Theorem 1.1 in $\mathbb{R}^k$ gives

$$|m_k| = (1-\beta_1)\Big|\sum_{j=1}^{k} \beta_1^{k-j} g_j\Big| \le (1-\beta_1)\Big(\sum_{j=1}^{k} \gamma^{k-j}\Big)^{1/2}\Big(\sum_{j=1}^{k} \beta_2^{k-j} g_j^2\Big)^{1/2} = (1-\beta_1)\sqrt{\frac{1-\gamma^k}{1-\gamma}}\sqrt{\frac{v_k}{1-\beta_2}}.$$

Dividing by $1-\beta_1^k$ and by $\sqrt{\hat{v}_k} = \sqrt{v_k/(1-\beta_2^k)}$,

$$\frac{|\hat{m}_k|}{\sqrt{\hat{v}_k}} \le \frac{1-\beta_1}{\sqrt{(1-\beta_2)(1-\gamma)}} \cdot \frac{\sqrt{(1-\gamma^k)(1-\beta_2^k)}}{1-\beta_1^k}.$$

Write $a = \gamma^k$ and $b = \beta_2^k$, so that $ab = \beta_1^{2k}$. Since $a + b \ge 2\sqrt{ab}$, $(1-a)(1-b) = 1-(a+b)+ab \le 1-2\sqrt{ab}+ab = (1-\beta_1^k)^2$, so the last fraction is at most 1. Finally $|\theta_k - \theta_{k-1}| = \eta_k|\hat{m}_k|/(\sqrt{\hat{v}_k}+\epsilon) \le \eta_k|\hat{m}_k|/\sqrt{\hat{v}_k}$. □

For the model's values, $\beta_1^2/\beta_2 = 0.7225/0.99$ and the bound is

$$\frac{0.15}{\sqrt{0.01\,(1-0.7225/0.99)}} \approx 2.886,$$

so no parameter ever moves by more than $2.89\,\eta_k$ in one step, however large or small its gradients are. The bound cannot be improved much: equality holds in Theorem 1.1 when $g_j = (\beta_1/\beta_2)^{k-j}$, gradients that grow steadily, and then the ratio $\hat{m}_k/\sqrt{\hat{v}_k}$ tends to the bound as k grows. Plain gradient descent moves each parameter by η times its partial derivative, which can be anything.

Example 6.6. Run Adam with the model's settings on $f(x,y) = x^2 + 3y^2$ from $(1,1)$, with exact gradients. The first gradient is $(2,6)$, and $\hat{\mathbf{m}}_1 = (2,6)$, $\hat{\mathbf{v}}_1 = (4,36)$, so the first step is $0.01\,(2/(2+\epsilon), 6/(6+\epsilon)) \approx (0.01, 0.01)$. It points straight at the minimum, along $-(1,1)/\sqrt{2}$, where steepest descent pointed along $-(1,3)/\sqrt{10}$ (Example 1.5). The two coordinates see gradients $2x$ and $6y$, and dividing each by its own scale removes the factor 3 that made the valley narrow. While $x_k = y_k$ the gradients stay in the ratio $1:3$, and the iterates stay on the line $y = x$ (up to the tiny effect of ϵ). After 100 steps they are at about $(0.252, 0.252)$, after 200 at $(0.0213, 0.0213)$, and after 1000 steps both coordinates are below 10^{-27}.

6.7. Learning-rate schedules

Theorem 6.5 shows a trade-off: a large learning rate makes fast progress at first, and a small one lowers the noise floor $\eta L\sigma^2$. A *learning-rate schedule* gets some of both by letting η_k depend on k, large early and small late. The proof of Theorem 6.5 works step by step, so it extends directly.

In Example 6.3, with $\eta = 0.1$ the expected squared gradient $\mathbb{E}[(\theta_k - 3)^2]$ is already 0.246 after 50 steps, and stays there: that is its limit $\eta\sigma^2/(2-\eta)$. With $\eta = 0.01$ it is still 3.31 after 50 steps, but after 1000 steps it reaches 0.0235, ten times lower. A schedule aims to have both.

Theorem 6.11 (SGD with a schedule)**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be L-smooth and bounded below, with $f^* = \inf f$, and let $\mathbf{g}$ be a stochastic gradient for f (Definition 6.4) with variance at most σ^2 at every point. Run SGD with learning rates $0 < \eta_k \le 1/L$. Then for every $K \ge 1$,*

$$\sum_{k=0}^{K-1} \eta_k \, \mathbb{E}\big[\|\nabla f(\theta_k)\|^2\big] \le 2\big(f(\theta_0) - f^*\big) + L\sigma^2 \sum_{k=0}^{K-1} \eta_k^2 .$$

Proof. The proof of Theorem 6.5, with η_k in place of η at step k, gives Equation 6.2 with η_k. Multiply it by 2, add for $k = 0, \dots, K-1$, and use $\mathbb{E}[f(\theta_K)] \ge f^*$. □

Dividing by $\sum_k \eta_k$ bounds a weighted average of the expected squared gradients by

$$\frac{2\big(f(\theta_0) - f^*\big) + L\sigma^2 \sum_k \eta_k^2}{\sum_k \eta_k} .$$

If $\sum_k \eta_k$ grows without bound while $\sum_k \eta_k^2$ stays bounded, this tends to 0 as $K \to \infty$: the noise floor disappears. These are the conditions that Robbins and Monro (1951) introduced for *stochastic approximation*, the study of methods that find a root or a minimum from noisy measurements,

and $\eta_k = \eta_0/(k+1)$ satisfies them, since the harmonic series diverges and $\sum 1/(k+1)^2$ converges.

A run of fixed length K can use a *linear decay* $\eta_k = \eta_0(1 - k/K)$ for $k = 0, \dots, K-1$, which falls from η_0 to η_0/K. Then

$$\sum_{k=0}^{K-1} \eta_k = \eta_0 \, \frac{K+1}{2}, \qquad \sum_{k=0}^{K-1} \eta_k^2 = \eta_0^2 \, \frac{(K+1)(2K+1)}{6K}.$$

This is the schedule of the model, with $\eta_0 = 0.01$ and $K = 1000$ (its step $k+1$ in the numbering of Definition 6.7). The sums are 5.005 and about 0.0334, against 10 and 0.1 for the constant rate 0.01. In the bound above, the noise term $L\sigma^2 \sum \eta_k^2 / \sum \eta_k$ falls from $0.01\,L\sigma^2$ to about $0.0067\,L\sigma^2$, and the first term doubles. The theorems of this chapter cover SGD, not Adam; the model's use of a decaying schedule with Adam rests on experience more than on a theorem.

6.8. Notes and further reading

Bottou et al. (2018) survey stochastic gradient methods for machine learning. Their sec. 4 proves convergence results of the kind given here under weaker assumptions; Theorem 6.5 is a special case of their thm. 4.8, and they also treat strongly convex functions and decreasing learning rates. Stochastic approximation began with Robbins and Monro (1951). The heavy-ball method is due to Polyak (1964); Goh (2017) explains its behaviour on quadratics with interactive diagrams, and Lessard et al. (2016) show why its guarantees do not extend beyond quadratics. Nesterov (2018, ch. 2) describes accelerated methods that do achieve the $\sqrt{\kappa}$ rate on every smooth strongly convex function.

Adam is from Kingma and Ba (2014), whose sec. 2.1 gives an informal bound on the step: $\eta(1-\beta_1)/\sqrt{1-\beta_2}$, which is 1.5η for the model's settings. That value is the limit when a single gradient is non-zero; Theorem 6.10 shows that steadily growing gradients can exceed it, up to 2.89η. Reddi et

al. (2018) give a convex problem on which Adam does not converge and identify a gap in the original convergence proof.

6.9. Exercises

Exercise 6.1. Two fair dice are thrown independently, and S is the total score. Use Theorem 6.1 and Theorem 6.3 to find $\mathbb{E}[S]$ and $\mathrm{Var}[S]$, and check the answers by listing the 36 outcomes.

Exercise 6.2. In Example 6.2, choose a batch of $B = 2$ *different* examples: one of the 6 ordered pairs (a, b) with $a \neq b$, all equally likely.

(a) At $\theta = 0$, find the expectation and the variance of $G = -(y_a + y_b)/2$, and compare with $\frac{7}{3}$.

(b) In general, let $\mathbf{d}_1, \dots, \mathbf{d}_N$ be vectors with $\sum_a \mathbf{d}_a = \mathbf{0}$ and $\sigma^2 = \frac{1}{N}\sum_a \|\mathbf{d}_a\|^2$, and let (I, J) be a uniformly chosen ordered pair of different indices. Show that $\mathbb{E}[\mathbf{d}_I \cdot \mathbf{d}_J] = -\sigma^2/(N-1)$.

(c) Deduce that the mean of B gradients chosen without replacement has variance $\frac{\sigma^2}{B} \cdot \frac{N-B}{N-1}$.

Exercise 6.3. In Example 6.3, let $e_k = \theta_k - 3$.

(a) Show that $e_{k+1} = (1-\eta)e_k + \eta(y_{I_k} - 3)$, where I_k is the example drawn at step k.

(b) Show that $\mathbb{E}[e_{k+1}^2] = (1-\eta)^2\,\mathbb{E}[e_k^2] + \eta^2\sigma^2$.

(c) Solve this recurrence, and show that $\mathbb{E}[e_k^2] \to \eta\sigma^2/(2-\eta)$ when $0 < \eta < 2$.

Exercise 6.4. Let $\mathbf{w}_0 = \mathbf{0}$ and, for $k \geq 0$, $\mathbf{w}_{k+1} = \beta\mathbf{w}_k - \eta\nabla f(\theta_k)$ and $\theta_{k+1} = \theta_k + \mathbf{w}_{k+1}$.

(a) Show that this produces the same points as Equation 6.3.

(b) Show that $\mathbf{w}_k = -\eta \sum_{j=0}^{k-1} \beta^{k-1-j} \nabla f(\theta_j)$, and compare with Theorem 6.8.

Exercise 6.5. Let $\lambda = 1$ and $\beta = \frac{1}{4}$ in Equation 6.4.

(a) For which $\eta > 0$ does every solution tend to 0?

(b) For which η are the roots complex? Find them, and their modulus, when $\eta = 1$.

Exercise 6.6. Let $\beta = 0.9$.

(a) What is the smallest k with $1 - \beta^k \geq 0.99$?

(b) Show that the oldest half of the weights in Theorem 6.8, those on $g_1, \dots, g_{k/2}$ for even k, add up to $\beta^{k/2}(1 - \beta^{k/2})$.

Exercise 6.7.

(a) Let $\epsilon = 0$ and $c > 0$. Show that running Adam on cf instead of f, from the same starting point and with exact gradients, gives the same iterates (as long as no $\hat{v}_{k,i}$ is zero, so that every step is defined).

(b) Show that if a coordinate's gradient is the same non-zero number g at every step, its Adam step is $-\eta_k \, g/(|g| + \epsilon)$ at every step.

Exercise 6.8. Prove the two sums for linear decay in Section 6.7, and use them to show that with linear decay the bound after Theorem 6.11 is

$$\frac{4(f(\theta_0) - f^*)}{\eta_0(K+1)} + \frac{L\sigma^2\eta_0(2K+1)}{3K}.$$

Part IV.

Neural networks

7. Neural networks

Gradient descent (Chapter 5) adjusts a vector of parameters to make a number, the loss, small. It needs a family of functions to adjust: a rule that turns an input vector and a parameter vector into an output vector, flexible enough to fit the data and simple enough to differentiate. A neural network is such a family. It is a composition of simple maps: an affine map, then a fixed function applied to each component, then another affine map, and so on. Each piece has a derivative we already know from Chapter 1, and the chain rule of Chapter 2 combines them. This chapter writes that combination out for the multilayer perceptron, which also appears inside the language model that Chapter 11 builds (a transformer), and finds that the gradient of the loss comes from one backward sweep of matrix–vector products. That sweep is reverse mode (Chapter 4) organised by layers.

Two questions about the networks themselves come next. The first is what a network can represent: a network with a single intermediate layer can approximate any continuous function of one variable on a closed interval as closely as we like, and the proof builds the network explicitly. The second is where training starts. The parameters are drawn at random, and their spread decides whether the numbers passing through the network keep their size or grow or shrink from layer to layer.

7.1. Neurons and layers

The simplest family of maps from $\mathbb{R}^n$ to $\mathbb{R}^m$ with adjustable numbers in it is the family of affine maps $\mathbf{x} \mapsto W\mathbf{x} + \mathbf{b}$ (Example 1.6), whose

adjustable numbers are the entries of the $m \times n$ matrix W and of the vector $\mathbf{b} \in \mathbb{R}^m$. Affine maps alone are too rigid. An affine function of one variable has a straight-line graph, so no choice of its two numbers fits even $|x|$ (two of the neurons defined below do, by Exercise 7.6). Composing affine maps does not help: composing two of them gives another, since $W_2(W_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 = (W_2W_1)\mathbf{x} + (W_2\mathbf{b}_1 + \mathbf{b}_2)$, so a stack of affine maps can do no more than a single one. Inserting a fixed function of one variable between them, applied to each component, breaks this collapse. The simplest choice used in practice is $t \mapsto \max(t, 0)$, which keeps positive numbers and replaces negative ones by 0.

Here is the smallest piece of such a network. Fix the vector $\mathbf{w} = (1, -1)^\top$ and the number $b = 0$, and send an input $\mathbf{x} \in \mathbb{R}^2$ to $\max(\mathbf{w} \cdot \mathbf{x} + b, 0)$. On $\mathbf{x} = (2, 1)^\top$ this gives $\mathbf{w} \cdot \mathbf{x} + b = 1$ and then $\max(1, 0) = 1$. On $\mathbf{x} = (1, 2)^\top$ it gives -1 and then 0: the map responds to inputs on one side of the line $x_1 = x_2$ and ignores the rest. A second map of the same kind, with $\mathbf{w} = (\frac{1}{2}, 1)^\top$ and $b = -1$, gives $\frac{1}{2} + 2 - 1 = \frac{3}{2}$ on $(1, 2)^\top$ and outputs $\frac{3}{2}$. Applied to the same input, the two together compute

$$\begin{pmatrix} 1 & -1 \\ \frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \begin{pmatrix} 0 \\ -1 \end{pmatrix} = \begin{pmatrix} -1 \\ \frac{3}{2} \end{pmatrix} \quad \longmapsto \quad \begin{pmatrix} 0 \\ \frac{3}{2} \end{pmatrix},$$

an affine map whose matrix has one row for each of the two maps, followed by $\max(\cdot, 0)$ on each component.

In general a *neuron* (or *unit*) takes a vector $\mathbf{x} \in \mathbb{R}^n$ and computes $\sigma(\mathbf{w} \cdot \mathbf{x} + b)$, where the vector $\mathbf{w} \in \mathbb{R}^n$ holds its *weights*, the number b is its *bias*, and $\sigma \colon \mathbb{R} \to \mathbb{R}$ is a fixed function, the *activation function*. In the example both neurons have $\sigma(t) = \max(t, 0)$. Placing m neurons side by side, all reading the same input, and stacking their weight vectors as the rows of a matrix gives a layer.

Definition 7.1 (Layer)**.** A *layer* with weight matrix W $(m \times n)$, bias $\mathbf{b} \in \mathbb{R}^m$ and activation function σ is the map $F \colon \mathbb{R}^n \to \mathbb{R}^m$,

$$F(\mathbf{x}) = \sigma(W\mathbf{x} + \mathbf{b}),$$

where σ acts on each component. The vector $\mathbf{z} = W\mathbf{x} + \mathbf{b}$ is the layer's *pre-activation* and $\sigma(\mathbf{z})$ its *activation*.

Row i of W holds the weights of neuron i, column j holds the weights with which the m neurons read input j, entry w_{ij} is the weight with which neuron i reads input j, and b_i is the bias of neuron i. The number m of neurons is the *width* of the layer. The activation function in the example has a name.

Definition 7.2 (Rectified linear unit)**.** The *rectified linear unit* (ReLU) is the function $\operatorname{relu}\colon \mathbb{R} \to \mathbb{R}$,

$$\operatorname{relu}(x) = \max(x, 0) = \begin{cases} x & x \geq 0, \\ 0 & x < 0. \end{cases}$$

For $x > 0$, relu agrees with the identity on an interval around x, so its derivative there is 1; for $x < 0$ it agrees with 0 near x, so its derivative is 0. At 0 its graph has a corner. The difference quotient $(\operatorname{relu}(h) - \operatorname{relu}(0))/h$ is 1 for $h > 0$ and 0 for $h < 0$, so it has no limit as $h \to 0$, and relu is not differentiable at 0. It is continuous everywhere, and $|\operatorname{relu}(x) - \operatorname{relu}(y)| \leq |x - y|$ for all x and y. Section 7.3 returns to the corner. A neuron whose pre-activation is negative outputs 0 and is called *inactive*; the first neuron in the example is inactive on the input $(1, 2)^\top$.

Other common activation functions are the logistic function $1/(1 + e^{-x})$ and $\tanh x$, which are differentiable everywhere. The ReLU is cheaper to evaluate and its derivative is 0 or 1; the book's model uses it.

A layer is the affine map $\mathbf{x} \mapsto W\mathbf{x} + \mathbf{b}$, whose Jacobian is W at every point (Example 1.6), followed by the componentwise map $\mathbf{z} \mapsto \sigma(\mathbf{z})$, whose Jacobian at $\mathbf{z}$ is the diagonal matrix with entries $\sigma'(z_1), \dots, \sigma'(z_m)$ (Example 1.7). The chain rule (Theorem 2.1) says that the Jacobian of a composition is the product of the Jacobians, $J_{g \circ f}(\mathbf{a}) = J_g(f(\mathbf{a}))\, J_f(\mathbf{a})$,

when f is differentiable at $\mathbf{a}$ and g at $f(\mathbf{a})$. So if σ is differentiable at every component of $\mathbf{z}$, the Jacobian of the layer with respect to its input is

$$J_F(\mathbf{x}) = \operatorname{diag}(\sigma'(z_1), \dots, \sigma'(z_m))\, W. \tag{7.1}$$

Entry (i, j) is $\sigma'(z_i)w_{ij}$, the rate at which the output of neuron i changes per unit increment of input j. Row i is row i of W scaled by $\sigma'(z_i)$, the rate at which neuron i responds to its own pre-activation, and column j lists how every neuron responds to input j. In the example, at $\mathbf{x} = (1, 2)^\top$, $\mathbf{z} = (-1, \frac{3}{2})$ and $J_F = \operatorname{diag}(0, 1)\, W = \begin{pmatrix} 0 & 0 \\ \frac{1}{2} & 1 \end{pmatrix}$: the inactive neuron does not respond to small changes of the input at all. Example 1.7 assumed σ differentiable everywhere, but its argument used differentiability only at the components z_i, and the chain rule needs differentiability only at the point where it is applied.

A network feeds the output of one layer into the next. Feed the output $(0, \frac{3}{2})^\top$ of the example layer into one more neuron, with weights $(2, -1)$, bias $\frac{1}{2}$ and no activation function: it computes $2 \cdot 0 - \frac{3}{2} + \frac{1}{2} = -1$. That composition of two layers is a small multilayer perceptron.

Definition 7.3 (Multilayer perceptron)**.** Let $n_0, n_1, \dots, n_D$ be positive integers and σ an activation function. A *multilayer perceptron* (MLP) of depth D with widths $n_0, \dots, n_D$ has weight matrices W_l of size $n_l \times n_{l-1}$ and biases $\mathbf{b}_l \in \mathbb{R}^{n_l}$, for $l = 1, \dots, D$. On input $\mathbf{x} \in \mathbb{R}^{n_0}$ it computes

$$\mathbf{a}_0 = \mathbf{x}, \qquad \mathbf{z}_l = W_l \mathbf{a}_{l-1} + \mathbf{b}_l \quad (l = 1, \dots, D), \qquad \mathbf{a}_l = \sigma(\mathbf{z}_l) \quad (l = 1, \dots, D-1), \tag{7.2}$$

and its output is $\mathbf{z}_D$. Layers 1 to $D-1$ are the *hidden* layers. The parameters (Chapter 5) of the network are all the entries of $W_1, \mathbf{b}_1, \dots, W_D, \mathbf{b}_D$, collected in one vector $\theta \in \mathbb{R}^p$ with $p = \sum_{l=1}^{D} n_l(n_{l-1} + 1)$.

In words: $\mathbf{z}_l$ is the vector of pre-activations of layer l and $\mathbf{a}_l$ the vector of its outputs; each layer applies its affine map to the previous layer's outputs and then, except for the last layer, the activation function. The depth D

counts the layers that carry parameters, n_l is the width of layer l, and n_0 is the number of inputs. Layer l contributes $n_l n_{l-1}$ weights and n_l biases to the count p.

The last layer has no activation function: its output is a vector of real numbers that a loss function reads. The training data of Chapter 5 are here a finite list of *training examples*, pairs $(\mathbf{x}, \mathbf{y})$ of an input and the output we would like the network to give for it, called the *target*. A *loss function* $\ell\colon \mathbb{R}^{n_D} \to \mathbb{R}$ measures how far the output is from the target, and training means searching for parameters that make the average loss over the training examples small, by gradient descent or one of its variants (Chapter 6). When the target is a vector of real numbers (a *regression* problem) the loss might be $\ell(\mathbf{z}) = \frac{1}{2}\|\mathbf{z} - \mathbf{y}\|^2$. When the target is one of finitely many classes (a *classification* problem, such as predicting the next letter of a word) it is the cross-entropy loss of Chapter 8. Either way, for one training example $(\mathbf{x}, \mathbf{y})$ the loss is

$$L(\theta) = \ell(\mathbf{z}_D),$$

where ℓ is a fixed function that depends on $\mathbf{y}$, and $\mathbf{z}_D$ depends on θ through Equation 7.2.

The MLP inside the book's final model (Chapter 11) has depth 2, widths 16, 64 and 16, and no biases, so it has $64 \times 16 + 16 \times 64 = 2048$ parameters.

Example 7.1. The two-layer computation above is an MLP of depth 2 with widths 2, 2 and 1 and ReLU activation, with

$$W_1 = \begin{pmatrix} 1 & -1 \\ \frac{1}{2} & 1 \end{pmatrix}, \quad \mathbf{b}_1 = \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \quad W_2 = \begin{pmatrix} 2 & -1 \end{pmatrix}, \quad b_2 = \tfrac{1}{2},$$

and $2 \cdot 3 + 1 \cdot 3 = 9$ parameters. On input $\mathbf{x} = (1, 2)^\top$ it computes $\mathbf{z}_1 = (-1, \frac{3}{2})^\top$, $\mathbf{a}_1 = (0, \frac{3}{2})^\top$ and $z_2 = -1$. With target $y = 1$ and loss $\ell(z) = \frac{1}{2}(z - y)^2$, the loss is $L = \frac{1}{2}(-2)^2 = 2$.

7.2. Backpropagation

A step of gradient descent replaces θ by $\theta - \eta \nabla L(\theta)$ for a small learning rate $\eta > 0$, so training needs the partial derivative of L with respect to every parameter. The quantities that organise the computation are the partial derivatives of the loss with respect to the pre-activations. We work them out in Example 7.1, from the output backwards.

- The loss is $\frac{1}{2}(z_2 - 1)^2$, so an increment h in z_2 changes L by about $(z_2 - 1)h = -2h$: $\partial L / \partial z_2 = -2$.
- An increment h in the second hidden pre-activation $(\mathbf{z}_1)_2 = \frac{3}{2}$ passes through the ReLU unchanged, since $\frac{3}{2} > 0$, and changes z_2 by $-h$, through the weight -1. So it changes L by about $(-2)(-h) = 2h$, and $\partial L / \partial (\mathbf{z}_1)_2 = 2$.
- A small increment in the first hidden pre-activation $(\mathbf{z}_1)_1 = -1$ leaves it negative, so the ReLU still outputs 0 and L does not change: $\partial L / \partial (\mathbf{z}_1)_1 = 0$.
- The weight $(W_1)_{21}$ multiplies $x_1 = 1$ into $(\mathbf{z}_1)_2$, so an increment h in it changes $(\mathbf{z}_1)_2$ by $1 \cdot h$ and L by about $2h$: $\partial L / \partial (W_1)_{21} = 2 \cdot 1 = 2$.

Each derivative with respect to a pre-activation came from the one in the layer above, multiplied by a weight and by the derivative of the ReLU, and each derivative with respect to a weight is the derivative for the pre-activation it feeds times the input it multiplies. The theorem states this for every MLP.

Theorem 7.1 (Backpropagation)**.** *Consider an MLP as in Definition 7.3, a fixed input* $\mathbf{x}$ *and a loss* $L(\theta) = \ell(\mathbf{z}_D)$*. Fix* θ*, and suppose that* ℓ *is differentiable at* $\mathbf{z}_D$ *and that* σ *is differentiable at every component of* $\mathbf{z}_1, \dots, \mathbf{z}_{D-1}$*. Define vectors* $\delta_l \in \mathbb{R}^{n_l}$ *by*

$$\delta_D = \nabla \ell(\mathbf{z}_D), \qquad \delta_l = (W_{l+1}^\top \delta_{l+1}) \odot \sigma'(\mathbf{z}_l) \quad (l = D-1, \dots, 1), \tag{7.3}$$

where $\odot$ *multiplies vectors component by component and* $\sigma'(\mathbf{z}_l)$ *is the vector of the numbers* $\sigma'((\mathbf{z}_l)_i)$*. Then:*

(a) *δ_l is the gradient of the loss with respect to $\mathbf{z}_l$: if g_l is the map from $\mathbf{z}_l$ to the loss given by layers $l+1, \dots, D$ followed by ℓ, then g_l is differentiable at $\mathbf{z}_l$ and $\nabla g_l(\mathbf{z}_l) = \delta_l$;*

(b) *L is differentiable at θ, and for each l,*

$$\frac{\partial L}{\partial W_l} = \delta_l \, \mathbf{a}_{l-1}^\top, \qquad \frac{\partial L}{\partial \mathbf{b}_l} = \delta_l, \tag{7.4}$$

where $\partial L/\partial W_l$ is the $n_l \times n_{l-1}$ matrix whose (i,j) entry is $\partial L/\partial (W_l)_{ij}$, and $\partial L/\partial \mathbf{b}_l$ is the vector of the $\partial L/\partial (\mathbf{b}_l)_i$.

In words, $(\delta_l)_i$ measures how strongly the loss responds to the pre-activation of neuron i in layer l. To get it, Equation 7.3 collects the values δ_{l+1} of the layer above through the transposed weights, so that neuron i receives $\sum_k (W_{l+1})_{ki}(\delta_{l+1})_k$, one term for each neuron k that reads it, and then multiplies by $\sigma'((\mathbf{z}_l)_i)$. In the matrix $\partial L/\partial W_l$, row i belongs to neuron i of layer l, column j to input j of that layer, and entry (i,j) is the product $(\delta_l)_i (\mathbf{a}_{l-1})_j$: the partial derivative of the loss with respect to the neuron's pre-activation, times the input that the weight multiplies.

Proof.

(a) By downward induction on l. For $l = D$, $g_D = \ell$, which is differentiable at $\mathbf{z}_D$ with gradient δ_D by definition. Suppose the claim holds for $l+1$, where $l \le D-1$. The map from $\mathbf{z}_l$ to $\mathbf{z}_{l+1}$ is $h(\mathbf{u}) = W_{l+1}\sigma(\mathbf{u}) + \mathbf{b}_{l+1}$, and $g_l = g_{l+1} \circ h$. By Equation 7.1 and Example 1.6, h is differentiable at $\mathbf{z}_l$ with Jacobian $W_{l+1}\operatorname{diag}(\sigma'(\mathbf{z}_l))$. By the induction hypothesis g_{l+1} is differentiable at $h(\mathbf{z}_l) = \mathbf{z}_{l+1}$ with Jacobian $\delta_{l+1}^\top$, a single row. By Theorem 2.1, g_l is differentiable at $\mathbf{z}_l$ with Jacobian

$$J_{g_l}(\mathbf{z}_l) = \delta_{l+1}^\top W_{l+1} \operatorname{diag}(\sigma'(\mathbf{z}_l)).$$

The gradient is the transpose of this row, $\operatorname{diag}(\sigma'(\mathbf{z}_l)) W_{l+1}^\top \delta_{l+1}$, and multiplying a vector by a diagonal matrix multiplies its components by the diagonal entries. This is Equation 7.3.

(b) First, L is differentiable at θ. Write the forward pass as a chain of maps that carry the parameters along. The map $(\theta, \mathbf{a}_{l-1}) \mapsto (\theta, \mathbf{z}_l)$ has components that are polynomials in its inputs, so it is differentiable everywhere by Theorem 1.6. The map $(\theta, \mathbf{z}_l) \mapsto (\theta, \sigma(\mathbf{z}_l))$ is differentiable at the current point by the argument of Example 1.7, because σ is differentiable at each component of $\mathbf{z}_l$. Finally ℓ is differentiable at $\mathbf{z}_D$. Starting from $(\theta, \mathbf{x})$, the composition of these maps is L, so Theorem 2.1 shows that L is differentiable at θ.

By Theorem 1.5 the partial derivatives of L exist, and we may compute each one along a line. Fix l, i and j, and change only the entry $(W_l)_{ij}$, to $(W_l)_{ij} + t$. The vector $\mathbf{a}_{l-1}$ depends only on $\mathbf{x}$ and the parameters of layers 1 to $l-1$, so it does not change, and $\mathbf{z}_l$ becomes $\mathbf{z}_l + t\,(\mathbf{a}_{l-1})_j\,\mathbf{e}_i$. The layers above l do not change, so the loss becomes $g_l(\mathbf{z}_l + t\,(\mathbf{a}_{l-1})_j\,\mathbf{e}_i)$. By (a) and Theorem 2.1, its derivative at $t = 0$ is $\delta_l \cdot (\mathbf{a}_{l-1})_j \mathbf{e}_i = (\delta_l)_i (\mathbf{a}_{l-1})_j$, which is entry (i, j) of $\delta_l \mathbf{a}_{l-1}^\top$. Changing $(\mathbf{b}_l)_i$ by t changes $\mathbf{z}_l$ by $t\mathbf{e}_i$, and the same argument gives $(\delta_l)_i$. □

The name backpropagation, which Chapter 4 gives to reverse mode applied to a network, describes the order of work. The forward pass computes and stores every $\mathbf{z}_l$ and $\mathbf{a}_l$. The backward pass starts from δ_D, applies Equation 7.3 to get $\delta_{D-1}, \ldots, \delta_1$ in turn, and at each layer reads off the parameter gradients from Equation 7.4. Definition 4.1 calls the partial derivative $\bar{v} = \partial L / \partial v$ of the final output with respect to an intermediate quantity v the adjoint of v, and Theorem 4.1 shows that reverse accumulation, one sweep from the output back to the inputs, computes every adjoint of an acyclic computational graph. The vector δ_l is the vector of adjoints $\bar{\mathbf{z}}_l$, and Equation 7.3 is reverse accumulation applied to the graph of the network, with the scalar nodes of each layer grouped into one vector. The transpose $W_{l+1}^\top$ appears because reverse mode multiplies by Jacobians from the left: a row vector times W_{l+1} is, after transposing, $W_{l+1}^\top$ times a column.

Cost. Count multiplications, since each comes with roughly one addition. The forward pass spends $n_l n_{l-1}$ multiplications on $W_l \mathbf{a}_{l-1}$, so $F = \sum_{l=1}^{D} n_l n_{l-1}$ in all, plus lower-order terms for the biases and activations. The backward pass spends $n_l n_{l-1}$ on each outer product $\delta_l \mathbf{a}_{l-1}^\top$, $n_{l+1} n_l$ on each product $W_{l+1}^\top \delta_{l+1}$ for $l = 1, \dots, D-1$, and n_l on each $\odot$. That is

$$2F - n_1 n_0 + \sum_{l=1}^{D-1} n_l$$

multiplications, at most twice the forward pass plus a term linear in the widths. So one forward and one backward pass together cost about three forward passes, however many parameters there are, as Theorem 4.3 predicts: there, a gradient costs at most a constant multiple of the cost of evaluating the function. The price is memory: the backward pass reads every stored $\mathbf{a}_{l-1}$ and $\mathbf{z}_l$.

The theorem translates line by line into code. The function below uses ReLU, whose derivative it takes to be 1 for positive inputs and 0 otherwise (Section 7.3 explains the choice at 0), and the loss $\ell(\mathbf{z}) = \frac{1}{2}\|\mathbf{z} - \mathbf{y}\|^2$, whose gradient is $\mathbf{z} - \mathbf{y}$.

```
import numpy as np

def mlp_backprop(Ws, bs, x, y):
    """Loss and gradients of a ReLU MLP with squared-error
    ↪ loss."""
    a, zs, acts = x, [], [x]
    for l, (W, b) in enumerate(zip(Ws, bs)):
        z = W @ a + b
        zs.append(z)
        a = np.maximum(z, 0) if l < len(Ws) - 1 else z
        acts.append(a)
    loss = 0.5 * np.sum((zs[-1] - y) ** 2)
    delta = zs[-1] - y          # delta_D, the gradient of the
↪   loss
    dWs, dbs = [None] * len(Ws), [None] * len(Ws)
```

```
    for l in reversed(range(len(Ws))):
        dWs[l] = np.outer(delta, acts[l])   # delta_l a_{l-1}^T
        dbs[l] = delta
        if l > 0:
            delta = (Ws[l].T @ delta) * (zs[l - 1] > 0)
    return loss, dWs, dbs
```

Example 7.2. Continue Example 7.1. Here $\ell(z) = \frac{1}{2}(z-1)^2$, so $\delta_2 = z_2 - 1 = -2$. Then

$$\delta_1 = (W_2^\top \delta_2) \odot \sigma'(\mathbf{z}_1) = \begin{pmatrix} -4 \\ 2 \end{pmatrix} \odot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \end{pmatrix},$$

because $\mathrm{relu}'(-1) = 0$ and $\mathrm{relu}'(3/2) = 1$. By Equation 7.4,

$$\frac{\partial L}{\partial W_2} = \delta_2 \mathbf{a}_1^\top = \begin{pmatrix} 0 & -3 \end{pmatrix}, \quad \frac{\partial L}{\partial b_2} = -2, \quad \frac{\partial L}{\partial W_1} = \delta_1 \mathbf{x}^\top = \begin{pmatrix} 0 & 0 \\ 2 & 4 \end{pmatrix}, \quad \frac{\partial L}{\partial \mathbf{b}_1} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}.$$

The inactive first neuron passes no gradient back: its own weights and bias have partial derivative 0 at this input. Central differences with step 10^{-6} agree with all nine numbers to within 10^{-9}.

7.3. What automatic differentiation computes at the corner

Theorem 7.1 needs σ to be differentiable at every pre-activation. For the ReLU that means no component of any $\mathbf{z}_l$ with $l < D$ is exactly 0. An automatic differentiation *engine*, a program that carries out the reverse accumulation of Chapter 4 on the operations of another program, does not test this. It stores, for each elementary function, a rule for the local derivative, and for the ReLU the rule fixes a value at 0. TensorFlow (Bolte

and Pauwels 2020, sec. 1) uses the rule below, and so do the engine of Section 4.7 and the program behind the book's final model:

$$\mathrm{relu}'(x) := \begin{cases} 1 & x > 0, \\ 0 & x \leq 0. \end{cases}$$

Call the value assigned to $\mathrm{relu}'(0)$ g; here $g = 0$. The engine then runs reverse accumulation with g in place of the missing derivative. What can we say about the result?

Corollary 7.1. *Let an MLP with ReLU activation, an input* $\mathbf{x}$*, a loss* ℓ *differentiable at* $\mathbf{z}_D$*, and parameters* θ *be given. If no component of* $\mathbf{z}_1, \ldots, \mathbf{z}_{D-1}$ *is* 0*, then* L *is differentiable at* θ*, and backpropagation returns* $\nabla L(\theta)$ *whatever value* g *it assigns to* $\mathrm{relu}'(0)$*.*

Proof. relu is differentiable at every non-zero number, so Theorem 7.1 applies and its formulas give the gradient. They evaluate relu' only at components of the $\mathbf{z}_l$, none of which is 0, so g is never used. □

So the value g matters only at parameter values where some pre-activation is exactly 0 for the input at hand. There the engine still returns a vector, but that vector need not be the gradient, even when the gradient exists.

Example 7.3. The function $f(x) = \mathrm{relu}(x) - \mathrm{relu}(-x)$ equals x for every x (check $x \geq 0$ and $x < 0$ separately), so $f'(0) = 1$. It is a network of depth 2 with one input, two hidden units with input weights 1 and -1, and output weights 1 and -1. At $x = 0$ both pre-activations are 0. An engine that assigns g to $\mathrm{relu}'(0)$ applies the chain rule to this program and returns

$$1 \cdot g \cdot 1 + (-1) \cdot g \cdot (-1) = 2g,$$

which is 0 for $g = 0$. The engine reports a derivative of 0 for the identity function.

The example is small but the behaviour is general: an engine differentiates the program, not the function, and two programs for the same function can give different answers at a corner. Bolte and Pauwels (2020, sec. 1) show the same with TensorFlow, and prove [Proposition 3] that for a wide class of programs, including ReLU networks, the engine's output equals the gradient at every interior point of each of finitely many regions whose closures cover the whole space. For an MLP and a fixed input, the sign patterns of the pre-activations give such regions directly: on the set of parameters where every pre-activation has a prescribed non-zero sign, Corollary 7.1 applies at every point. The boundaries of these regions lie in the set where some pre-activation is 0, which is exactly what Corollary 7.1 excludes. The book does not need the general theorem. Corollary 7.1 is a statement about the point at which we compute, and in a given forward pass we can check directly whether any pre-activation is exactly 0.

Exact zeros are rare when the weights are drawn at random from a continuous distribution, as Section 7.5 shows after setting up the probability needed to say so. In floating-point arithmetic a sum of products of non-zero numbers comes out as exactly 0 only through exact cancellation. Once training has moved the weights, the book makes no probabilistic claim and relies on Corollary 7.1 at each step.

Exact zeros do occur systematically in one situation. If every unit of a hidden layer is inactive, then $\mathbf{a}_{l-1} = \mathbf{0}$, and in a network without biases, such as the book's model, every component of $\mathbf{z}_l = W_l \mathbf{a}_{l-1}$ is exactly 0. The same happens when an input vector is $\mathbf{0}$. Then Corollary 7.1 gives no guarantee, and the engine returns whatever its choice of g produces.

7.4. Universal approximation in one variable

What functions can a ReLU network represent? A network of depth 2 with one input, N hidden units and one output computes

$$x \mapsto c + \sum_{k=0}^{N-1} d_k \operatorname{relu}(w_k x + b_k).$$

Each term with $w_k \neq 0$ is 0 on one side of the point $-b_k/w_k$ and affine on the other, so the sum is continuous and *piecewise linear*: affine on each of finitely many intervals. This section shows that such functions can approximate every continuous function on a closed interval, and that the approximation can be written down explicitly.

The plan has two steps. First, mark some points on the graph of the function and join neighbouring points by straight lines. For x^2 on $[0, 1]$, the broken line through $(0, 0)$, $(\frac{1}{2}, \frac{1}{4})$ and $(1, 1)$ stays within $\frac{1}{16}$ of x^2, and the broken line through the eleven points with $x = 0, 0.1, \ldots, 1$ stays within $\frac{1}{400}$ (Example 7.5). Second, write the broken line as a network. The first step needs a property of continuous functions on closed intervals, which says that the points can be chosen to make the error as small as we like.

Lemma 7.1 (Cantor's theorem on uniform continuity)**.** *Let $f\colon [a, b] \to \mathbb{R}$ be continuous. For every $\varepsilon > 0$ there is a $\delta > 0$ such that $|f(x) - f(y)| < \varepsilon$ for all x and y in $[a, b]$ with $|x - y| < \delta$.*

A function with the property in the conclusion is *uniformly continuous* on $[a, b]$. Continuity at each point gives a δ for each point; the lemma says that one δ works for the whole interval. The book does not prove it; a proof is in Zorich (2015, sec. 4.2.2, Theorem 4), and it uses the completeness of the real numbers. It fails on intervals that are not closed: $1/x$ is continuous on $(0, 1]$, but for every $\delta < 1$ the points $x = \delta$ and $y = \delta/2$ are closer than δ while $|1/x - 1/y| = 1/\delta > 1$, so no δ works for $\varepsilon = 1$.

Lemma 7.2 (Piecewise-linear interpolation)**.** *Let $f\colon [a,b] \to \mathbb{R}$ be continuous, let $a = t_0 < t_1 < \cdots < t_N = b$, and let $\hat{f}$, the* piecewise-linear interpolant *of f at these points, be the function that equals f at each t_k and is affine on each interval $[t_{k-1}, t_k]$. If $\varepsilon > 0$ and $|f(x) - f(y)| < \varepsilon$ whenever $|x - y| \le \max_k(t_k - t_{k-1})$, then $|f(x) - \hat{f}(x)| < \varepsilon$ for every x in $[a,b]$.*

Proof. Let $x \in [t_{k-1}, t_k]$, and write $x = (1-\lambda)t_{k-1} + \lambda t_k$ with $0 \le \lambda \le 1$. Since $\hat{f}$ is affine on this interval, $\hat{f}(x) = (1-\lambda)f(t_{k-1}) + \lambda f(t_k)$, and so

$$f(x) - \hat{f}(x) = (1-\lambda)\big(f(x) - f(t_{k-1})\big) + \lambda\big(f(x) - f(t_k)\big).$$

Both $|x - t_{k-1}|$ and $|x - t_k|$ are at most $t_k - t_{k-1}$, so both brackets are less than ε in absolute value. The weights $1-\lambda$ and λ are non-negative with sum 1, so $|f(x) - \hat{f}(x)| < (1-\lambda)\varepsilon + \lambda\varepsilon = \varepsilon$. □

The second ingredient converts any continuous piecewise-linear function into a ReLU network. Consider $\mathrm{relu}(x) - \mathrm{relu}(x-1)$. For $x \le 0$ both terms are 0. For $0 \le x \le 1$ only the first is non-zero, and the sum is x. For $x \ge 1$ the sum is $x - (x-1) = 1$. The second unit switches on at $x = 1$ and cancels the slope 1 of the first, so the graph rises from 0 to 1 and then stays level. In general, start with the first slope, and at each point where the slope changes (each *breakpoint*) add a ReLU that switches on there and changes the slope by the right amount.

Lemma 7.3 (Piecewise-linear functions as sums of ReLUs)**.** *Let $a = t_0 < t_1 < \cdots < t_N = b$, and let g be continuous on $[a,b]$ and affine with slope m_k on $[t_{k-1}, t_k]$, for $k = 1, \ldots, N$. Then for every x in $[a,b]$,*

$$g(x) = g(t_0) + m_1 \,\mathrm{relu}(x - t_0) + \sum_{k=1}^{N-1} (m_{k+1} - m_k)\,\mathrm{relu}(x - t_k). \qquad (7.5)$$

Proof. Call the right side $h(x)$. Fix j between 1 and N and let $x \in [t_{j-1}, t_j]$. For $k \le j-1$, $x - t_k \ge 0$, so $\operatorname{relu}(x - t_k) = x - t_k$; for $k \ge j$, $x - t_k \le 0$, so $\operatorname{relu}(x - t_k) = 0$. Hence on $[t_{j-1}, t_j]$,

$$h(x) = g(t_0) + m_1(x - t_0) + \sum_{k=1}^{j-1} (m_{k+1} - m_k)(x - t_k),$$

an affine function with slope $m_1 + (m_2 - m_1) + \cdots + (m_j - m_{j-1}) = m_j$. So h and g are both affine with slope m_j on $[t_{j-1}, t_j]$. They agree at t_0. If they agree at t_{j-1}, they agree on all of $[t_{j-1}, t_j]$, since two affine functions with the same slope and the same value at one point are equal, and in particular they agree at t_j. By induction on j they agree on every interval. □

The right side of Equation 7.5 is a network of depth 2 with N hidden units. Unit k has input weight 1 and bias $-t_k$, its output weight is the change of slope at t_k, and the output bias is $g(t_0)$. The breakpoints set the biases and the changes of slope set the output weights.

Example 7.4. The piecewise-linear function through $(0, 0)$, $(1, 2)$, $(2, 1)$ and $(3, 1)$ has slopes 2, -1 and 0. By Equation 7.5,

$$g(x) = 2\operatorname{relu}(x) - 3\operatorname{relu}(x - 1) + \operatorname{relu}(x - 2) \qquad (0 \le x \le 3).$$

At $x = 3$ this gives $6 - 6 + 1 = 1$, as it should. The unit at $t_1 = 1$ has output weight -3 because the slope falls from 2 to -1 there.

Theorem 7.2 (Universal approximation in one variable)**.** *Let $f : [a, b] \to \mathbb{R}$ be continuous and $\varepsilon > 0$. There are a positive integer N and real numbers c, d_k and t_k ($k = 0, \dots, N-1$) such that the network*

$$g(x) = c + \sum_{k=0}^{N-1} d_k \operatorname{relu}(x - t_k)$$

satisfies $|f(x) - g(x)| < \varepsilon$ for every x in $[a, b]$. Equivalently, f is the uniform limit on $[a, b]$ of a sequence of networks of depth 2 with ReLU

activation, where a sequence of functions g_n converges uniformly *to* f *on* $[a,b]$ *if* $\max_{a\le x\le b}|f(x)-g_n(x)|\to 0$ *as* $n\to\infty$.

Proof. By Lemma 7.1 there is a $\delta>0$ with $|f(x)-f(y)|<\varepsilon$ whenever $|x-y|<\delta$. Choose N with $(b-a)/N<\delta$ and put $t_k=a+k(b-a)/N$ for $k=0,\dots,N$. Let $\hat f$ be the piecewise-linear interpolant of f at these points. By Lemma 7.2, $|f(x)-\hat f(x)|<\varepsilon$ on $[a,b]$. By Lemma 7.3, $\hat f$ is a network of the stated form, with $c=f(a)$, $d_0=m_1$ and $d_k=m_{k+1}-m_k$ for $k\ge 1$, where $m_k=(f(t_k)-f(t_{k-1}))/(t_k-t_{k-1})$.

For the second sentence, take $\varepsilon=1/n$ for $n=1,2,\dots$ to get networks g_n with $|f(x)-g_n(x)|<1/n$ for all x in $[a,b]$. Conversely, if such a sequence exists, any g_n with $1/n\le\varepsilon$ will do. □

Example 7.5. Let $f(x)=x^2$ on $[0,1]$ and $t_k=k/N$. The slope of the interpolant on $[t_{k-1},t_k]$ is $(t_k^2-t_{k-1}^2)/(t_k-t_{k-1})=t_k+t_{k-1}=(2k-1)/N$, so the slope rises by $2/N$ at each interior breakpoint, and Equation 7.5 gives

$$g_N(x)=\frac{x}{N}+\frac{2}{N}\sum_{k=1}^{N-1}\operatorname{relu}\Big(x-\frac{k}{N}\Big)\qquad(0\le x\le 1),$$

using $\operatorname{relu}(x)=x$ for $x\ge 0$. On $[t_{k-1},t_k]$ the interpolant is a chord of the parabola, and $x^2-g_N(x)=(x-t_{k-1})(x-t_k)$, because both sides are quadratics with leading coefficient 1 that vanish at t_{k-1} and t_k. The largest absolute value is at the midpoint, so

$$\max_{0\le x\le 1}|x^2-g_N(x)|=\frac{1}{4N^2}.$$

For an error below 10^{-4}, $N=51$ hidden units suffice and $N=50$ do not. The proof of Theorem 7.2 asks for many more: on $[0,1]$, $|x^2-y^2|=|x+y|\,|x-y|\le 2|x-y|$, so $\delta=\varepsilon/2$ works, and the proof takes $N>2/\varepsilon$, which is $N>20000$ for $\varepsilon=10^{-4}$.

Theorem 7.2 is an existence statement with a constructive proof, and its reach is limited in four ways.

- It gives no rate. The number of units depends on f through the δ of Lemma 7.1, and a function that changes quickly needs many more units than x^2 does (Exercise 7.7). Nothing in the theorem bounds N in terms of ε alone.
- It says nothing about training. The weights in the proof come from values of f, which training does not have. Whether gradient descent on a finite data set finds a good network is a different question, and the theorem does not address it.
- It controls the network only on $[a, b]$. Outside the interval g is affine on each side, and it may be far from f.
- It concerns one input variable. The corresponding theorem for continuous functions on a closed bounded subset of $\mathbb{R}^n$ holds for ReLU and for every continuous activation function that is not a polynomial (Leshno et al. 1993), with earlier versions for sigmoid-shaped activations by Cybenko (1989) and Hornik et al. (1989). Those proofs use functional analysis and do not construct the network the way Lemma 7.3 does.

7.5. Initialisation and the size of activations

Training starts from parameters drawn at random, a step called *initialisation*. Zero weights would not do: with all weights 0, every hidden unit in a layer computes the same thing, receives the same gradient by Equation 7.4, and stays equal to the others after every step. The spread of the random weights, measured by their standard deviation, matters too, because it sets the size of the pre-activations, and so of everything downstream.

The computation uses expectation and variance, which Section 6.1 defines for random variables on a finite probability space in Definition 6.1 and Definition 6.3, and independence, which it defines in Definition 6.2. It also uses rules proved there. Expectation is linear, by Theorem 6.1, and $\operatorname{Var}[cX] = c^2 \operatorname{Var}[X]$, by Lemma 6.2. The variance of a sum of independent random variables is the sum of their variances, by Theorem 6.3.

A small case shows what to expect. Let $\mathbf{x} = (1, 2)^\top$ and let the two weights w_1 and w_2 of a neuron be independent, each equal to s or $-s$ with probability $\frac{1}{2}$. The sample space has four equally likely outcomes, which give $z = w_1 + 2w_2$ the values $3s$, $-s$, s and $-3s$. So $\mathbb{E}[z] = 0$ and $\mathbb{E}[z^2] = \frac{1}{4}(9 + 1 + 1 + 9)s^2 = 5s^2$. The number 5 is $\|\mathbf{x}\|^2 = 1^2 + 2^2$: the variance of the pre-activation grows with the squared length of the input and with the variance s^2 of each weight.

The book's model, like most networks, draws its weights from the normal distribution, which can take any real value. A finite sample space cannot describe that, and probabilities of intervals, given by integrals, take the place of probabilities of single outcomes.

Definition 7.4 (Density and the normal distribution)**.** A *probability density* is a function $f\colon \mathbb{R} \to [0, \infty)$ with $\int_{-\infty}^{\infty} f(t)\,dt = 1$. A real random variable X *has density* f if

$$P(a \le X \le b) = \int_a^b f(t)\,dt \qquad \text{for all } a \le b.$$

Its expectation is $\mathbb{E}[X] = \int_{-\infty}^{\infty} t f(t)\,dt$, when $\int_{-\infty}^{\infty} |t| f(t)\,dt$ is finite, and more generally $\mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(t) f(t)\,dt$ when $\int_{-\infty}^{\infty} |g(t)| f(t)\,dt$ is finite. Its variance is $\mathrm{Var}[X] = \mathbb{E}\big[(X - \mathbb{E}[X])^2\big]$. For real μ and $s > 0$, the *normal distribution* $N(\mu, s^2)$ is the distribution with density

$$\varphi(t) = \frac{1}{\sqrt{2\pi s^2}}\, e^{-(t-\mu)^2/(2s^2)}.$$

Real random variables $X_1, \dots, X_k$ are *independent* if

$$P(a_1 \le X_1 \le b_1, \dots, a_k \le X_k \le b_k) = P(a_1 \le X_1 \le b_1) \cdots P(a_k \le X_k \le b_k)$$

for all intervals $[a_1, b_1], \dots, [a_k, b_k]$.

The density plays the part of the probabilities $P(\omega)$, and integrals replace sums. The density φ of $N(\mu, s^2)$ is the bell-shaped curve centred at μ; a

random variable with this distribution has mean μ and variance s^2, as the substitution $u = (t - \mu)/s$ and an integration by parts show. For example, a variable with distribution $N(0, 1)$ lies in $[-1, 1]$ with probability about 0.683 and in $[-2, 2]$ with probability about 0.954. A random variable with a density gives probability 0 to each single number c, since $P(c \le X \le c) = \int_c^c f(t)\,dt = 0$. Two random variables have the *same distribution* if they give the same probability to every interval. The rules of Chapter 6 quoted above hold for all random variables with finite variance, including those with densities and sums of them; the book uses them without proof, and Wasserman (2004, ch. 2 and ch. 3) proves them.

This settles the claim of Section 7.3 that exact zeros are rare at initialisation. Suppose the weights and bias of a unit in the first layer are independent, each with a normal distribution with mean 0 and positive variance, and $\mathbf{x}$ is a fixed input. Each term $w_j x_j$ with $x_j \neq 0$ is normal with mean 0 and variance $x_j^2 \operatorname{Var}[w_j]$, and a sum of independent normal random variables is again normal (Wasserman 2004, sec. 2.4). So $\mathbf{w} \cdot \mathbf{x} + b$, the sum of b and these terms, is normal with mean 0 and positive variance, and it equals 0 with probability 0.

Theorem 7.3 (Variance of a random linear map)**.** *Let $\mathbf{x} \in \mathbb{R}^n$ be fixed, and let W be an $m \times n$ random matrix whose entries w_{ij} are independent, with $\mathbb{E}[w_{ij}] = 0$ and $\operatorname{Var}[w_{ij}] = s^2$. Then $\mathbf{z} = W\mathbf{x}$ satisfies, for each i,*

$$\mathbb{E}[z_i] = 0, \qquad \operatorname{Var}[z_i] = \mathbb{E}[z_i^2] = s^2 \|\mathbf{x}\|^2,$$

and so $\mathbb{E}\|\mathbf{z}\|^2 = m s^2 \|\mathbf{x}\|^2$.

Proof. $z_i = \sum_j w_{ij} x_j$, where the x_j are constants. By linearity, $\mathbb{E}[z_i] = \sum_j x_j \mathbb{E}[w_{ij}] = 0$. The terms $w_{ij} x_j$ for $j = 1, \dots, n$ are independent, and $\operatorname{Var}[w_{ij} x_j] = x_j^2 s^2$, so $\operatorname{Var}[z_i] = s^2 \sum_j x_j^2 = s^2 \|\mathbf{x}\|^2$. Since $\mathbb{E}[z_i] = 0$, $\operatorname{Var}[z_i] = \mathbb{E}[z_i^2]$. Summing over i gives $\mathbb{E}\|\mathbf{z}\|^2$. □

If the input has components of typical size 1, so that $\|\mathbf{x}\|^2 \approx n$, then $\mathrm{Var}[z_i] \approx s^2 n$. Keeping the pre-activations at the size of the input means $s = 1/\sqrt{n}$: the standard deviation of the weights should shrink like one over the square root of the number of inputs to each unit. The same square root reappears in Theorem 10.1.

The ReLU halves the expected square, provided the pre-activation is as likely to be positive as negative in a precise sense. In the four-outcome example, $\mathrm{relu}(z)$ takes the values $3s$, 0, s and 0, so $\mathbb{E}[\mathrm{relu}(z)^2] = \frac{1}{4}(9+1)s^2 = \frac{5}{2}s^2$, half of $\mathbb{E}[z^2] = 5s^2$. The values of z there come in pairs $\pm 3s$ and $\pm s$ of equal probability, and the ReLU keeps one member of each pair. A random variable z is *symmetric* if $-z$ has the same distribution as z, that is, $P(a \le z \le b) = P(-b \le z \le -a)$ for all $a \le b$.

Lemma 7.4. *If z is symmetric and $\mathbb{E}[z^2]$ is finite, then $\mathbb{E}[\mathrm{relu}(z)^2] = \frac{1}{2}\mathbb{E}[z^2]$.*

Proof. For every real x, one of $\mathrm{relu}(x)$ and $\mathrm{relu}(-x)$ is 0 and the other is $|x|$, so $\mathrm{relu}(x)^2 + \mathrm{relu}(-x)^2 = x^2$. Hence $\mathbb{E}[\mathrm{relu}(z)^2] + \mathbb{E}[\mathrm{relu}(-z)^2] = \mathbb{E}[z^2]$. Since $-z$ has the same distribution as z, the two expectations on the left are equal. □

If the w_{ij} are independent and each is symmetric, as normal random variables with mean 0 are, then the variables $-w_{ij}$ are independent with the same distributions as the w_{ij}, so $-z_i = \sum_j (-w_{ij})x_j$ has the same distribution as $z_i = \sum_j w_{ij}x_j$, and each z_i is symmetric. Theorem 7.3 and Lemma 7.4 then give, for one ReLU layer without bias,

$$\mathbb{E}\|\,\mathrm{relu}(W\mathbf{x})\|^2 = \frac{ms^2}{2}\|\mathbf{x}\|^2. \tag{7.6}$$

The expected squared length of the layer's output is the squared length of its input times $ms^2/2$: the number of neurons times the variance of one weight, halved by the ReLU. In a stack of such layers with independent

weights, the input of layer l is itself random. Applying Equation 7.6 with that input held fixed, and then averaging over it (a step that needs conditional expectation, which the book does not develop), shows that layer l multiplies the expected squared norm by $n_l s_l^2/2$ (He et al. 2015, sec. 2.2). A product of many factors grows or shrinks exponentially unless the factors are close to 1. He et al. (2015) tracks the expected square of one component instead of the squared norm, which gives the factor $n_{l-1} s_l^2/2$ and the rule $s_l^2 = 2/n_{l-1}$ for ReLU layers; for layers of equal width the two factors agree. The earlier rule of Glorot and Bengio (2010, sec. 4.2) balances the forward pass against the backward pass of Equation 7.3 and takes $s_l^2 = 2/(n_{l-1} + n_l)$.

Example 7.6. The book's model draws every weight independently from $N(0, 0.08^2)$. Its MLP reads a vector $\mathbf{h} \in \mathbb{R}^{16}$ produced by RMS normalisation (Chapter 11), which divides a vector by the root mean square of its components. So $\|\mathbf{h}\|^2$ is just under 16, the width; it falls short of 16 only because a small constant is added to the mean square before the division. The first layer has 16 inputs, so by Theorem 7.3 its 64 pre-activations have mean 0 and variance $0.08^2 \times 16 = 0.1024$, a standard deviation of 0.32. The rule $s = 1/\sqrt{16}$ would give $s = 0.25$ and standard deviation 1; the rule of He et al. (2015) would give $s \approx 0.354$.

By Equation 7.6 the 64 activations have expected squared norm $64 \times 0.1024/2 \approx 3.28$, and the second layer turns them into 16 outputs of variance about $0.08^2 \times 3.28 \approx 0.021$, a standard deviation of about 0.145. The MLP adds its output to a vector whose components have root mean square about 1, so at the start of training it changes that vector by roughly 15% of its size. Simulating 300 freshly initialised models on random input sequences gives standard deviations 0.32 for the pre-activations and 0.14 for the outputs.

The small spread does no harm here because the model has one layer and normalises before each block. In a plain stack of ReLU layers of width 16 with this spread, each layer would multiply the expected squared norm by $16 \times 0.08^2/2 = 0.0512$, and five layers by about 3.5×10^{-7}. The

small weights also make the model's first predictions nearly uniform, which Chapter 8 measures.

7.6. Notes and further reading

Goodfellow et al. (2016, ch. 6) covers feedforward networks, activation functions, universal approximation and backpropagation, and [ch. 8] discusses initialisation among other practical matters of training. Rumelhart et al. (1986) popularised backpropagation as a way to train networks with hidden layers. Bolte and Pauwels (2020) give a mathematical account of what automatic differentiation computes for programs with corners, and Kakade and Lee (2018) show how to compute a correct generalised derivative at a cost comparable to that of a gradient. The variance argument of Section 7.5 is from Glorot and Bengio (2010) and, with the factor $\frac{1}{2}$ for ReLU, He et al. (2015). For universal approximation in several variables see Leshno et al. (1993), Cybenko (1989) and Hornik et al. (1989).

7.7. Exercises

Exercise 7.1.

(a) Count the parameters of an MLP with widths $3, 4, 4, 2$.

(b) A network of depth 2 with widths $16, 64, 16$ and no biases is evaluated on one input. Count the multiplications in the forward pass, and in a backward pass that also returns the gradient with respect to the input, as it must when the network sits inside a larger model.

Exercise 7.2. Let $F(\mathbf{x}) = \tanh(W\mathbf{x} + \mathbf{b})$ with $W = \begin{pmatrix} 1 & 2 \\ 0 & -1 \end{pmatrix}$ and $\mathbf{b} = (0, 1)^\top$. Find $J_F(\mathbf{0})$.

Exercise 7.3. An MLP of depth 2 with widths $1, 2, 1$ and ReLU activation has $W_1 = (1, -2)^\top$, $\mathbf{b}_1 = (1, 1)^\top$, $W_2 = (3, 1)$ and $b_2 = 0$. With loss $\ell(z) = \frac{1}{2}(z - 2)^2$ and input $x = 1$, compute the loss and all seven partial derivatives with Theorem 7.1.

Exercise 7.4. An engine assigns the value g to $\mathrm{relu}'(0)$.

(a) Show that for $f_1(x) = \mathrm{relu}(x) - \mathrm{relu}(-x)$ it returns $2g$ at $x = 0$, and for $f_2(x) = \mathrm{relu}(\mathrm{relu}(x)) - \mathrm{relu}(-x)$ it returns $g^2 + g$.

(b) Show that $f_1(x) = f_2(x) = x$ for all x, and deduce that no single value of g makes the engine return the true derivative at 0 for both programs.

Exercise 7.5. Write the function that is 0 outside $[0, 2]$, rises linearly from 0 at $x = 0$ to 1 at $x = 1$, and falls linearly to 0 at $x = 2$, as a sum of ReLUs valid for all real x.

Exercise 7.6.

(a) Show that $|x| = \mathrm{relu}(x) + \mathrm{relu}(-x)$ and $\max(x, y) = y + \mathrm{relu}(x - y)$.

(b) Find a network of depth 2 with two inputs, ReLU activation, three hidden units and one output that computes $\max(x, y)$ exactly for all x and y.

Exercise 7.7.

(a) Find the least N for which the network g_N of Example 7.5 approximates x^2 on $[0, 1]$ with error at most 10^{-6}.

(b) Let $f(x) = \sqrt{x}$ on $[0, 1]$, $t_k = k/N$, and $\hat{f}$ the interpolant. Show that $f(x) - \hat{f}(x) = 1/(4\sqrt{N})$ at $x = 1/(4N)$, and find how large N must be for the error to be at most 10^{-6} at that point.

Exercise 7.8. Let W be an $n \times n$ matrix with independent $N(0, s^2)$ entries, and $\mathbf{x} \in \mathbb{R}^n$ fixed.

(a) Find s such that $\mathbb{E}\|\operatorname{relu}(W\mathbf{x})\|^2 = \|\mathbf{x}\|^2$, and evaluate it for $n = 16$ and $n = 64$.

(b) For $s = 0.08$ and $n = 64$, find the factor by which one such layer multiplies the expected squared norm, and the least number of such layers after which the expected squared norm has fallen below 10^{-6} times its starting value.

8. Probability and loss functions

A language model predicts the next character by assigning a probability to every character that could come next. Training compares those probabilities with the character that actually came next, and the number that measures the mismatch is the loss. This chapter derives that loss from one principle, maximum likelihood: choose the parameters under which the observed data were most probable. Taking logarithms turns the likelihood into a sum, and its negative average is the cross-entropy between the data and the model.

Two pieces of mathematics carry the chapter. Gibbs' inequality says that the Kullback–Leibler divergence, a measure of how far one probability distribution is from another, is never negative. From it follows that the best model of a sequence of observations drawn from a fixed finite set is the table of relative frequencies, the result that Chapter 9 uses to build a first language model. The softmax function turns a vector of real scores into a distribution, and the gradient of the cross-entropy loss with respect to those scores is the difference between the predicted and the observed distributions. That difference is the vector δ_D with which backpropagation (Theorem 7.1) starts.

8.1. Distributions on a finite set

The book's model reads and writes text as a sequence of tokens, here the 26 letters and one boundary token that marks where a name starts and ends. The set of tokens is the vocabulary V, with $|V| = 27$ elements;

Definition 9.1 in Chapter 9 states these terms precisely. At each position the model's prediction is a list of 27 non-negative numbers summing to 1, one probability for each token that could come next. Such a list is a probability distribution on V.

Probability on a finite sample space Ω is as in Definition 6.1. There a random variable takes real values, and the same definitions serve for a function $X\colon \Omega \to V$ with values in any finite set V, which this chapter also calls a random variable: $P(X = v)$ is the probability of the event $\{\omega : X(\omega) = v\}$. For one roll of a fair die, Ω is $\{1, \dots, 6\}$ with each outcome of probability $\frac{1}{6}$; the random variable "the roll is even", with values in $\{\text{yes}, \text{no}\}$, takes each value with probability $\frac{1}{2}$. Often only these probabilities matter, not Ω itself.

A *probability distribution* on a finite set V is a function $p\colon V \to [0, 1]$ with $\sum_{v \in V} p(v) = 1$. If the elements of V are numbered $1, \dots, n$, we also write p as the vector $\mathbf{p} = (p_1, \dots, p_n)^\top$ with $p_i = p(i)$. A random variable X with values in V has distribution p if $P(X = v) = p(v)$ for each v, and we write $X \sim p$.

A distribution on V is also called a *categorical* distribution. The set of all of them is the *probability simplex* $\{\mathbf{p} \in \mathbb{R}^n : p_i \geq 0,\ \sum_i p_i = 1\}$; for $n = 3$ it is the triangle with corners $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$. The *uniform* distribution has $p(v) = 1/|V|$ for every v; it describes equally likely outcomes. For a function f on V, the expectation of $f(X)$ is $\mathbb{E}[f(X)] = \sum_{v \in V} p(v) f(v)$, by the formula over values in Theorem 6.1.

Data consist of many observations, and the model treats them as separate draws that do not influence each other. Roll two fair dice: the sample space is the set of 36 equally likely pairs, and the first roll X_1 and the second roll X_2 satisfy $P(X_1 = 1, X_2 = 2) = \frac{1}{36} = P(X_1 = 1)\, P(X_2 = 2)$, and likewise for every pair of values. As in Definition 6.2, random variables $X_1, \dots, X_N$ with values in V are independent if

$$P(X_1 = v_1, \dots, X_N = v_N) = P(X_1 = v_1) \cdots P(X_N = v_N) \qquad \text{for all } v_1, \dots, v_N \in V,$$

where the left side is the probability that all N equalities hold. They are *independent draws from* p if in addition each X_i has distribution p; then the probability of observing the sequence $v_1, \dots, v_N$ is $p(v_1) \cdots p(v_N)$. Definition 7.4 extends these notions to real random variables with densities.

8.2. Likelihood

Suppose we observe a sequence of outcomes and want to choose, among several candidate distributions, the one that explains it best. A natural score for a candidate is the probability it gives to exactly what we saw.

Example 8.1. Let $V = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and observe $\mathtt{a}, \mathtt{b}, \mathtt{a}, \mathtt{c}, \mathtt{a}$. If the five observations are independent draws from a distribution p, the probability of this sequence is $p(\mathtt{a})p(\mathtt{b})p(\mathtt{a})p(\mathtt{c})p(\mathtt{a}) = p(\mathtt{a})^3 p(\mathtt{b})p(\mathtt{c})$. The uniform distribution gives $3^{-5} = 1/243 \approx 0.00412$, and $p = (0.6, 0.2, 0.2)$ gives $0.6^3 \times 0.2^2 = 0.00864$, about twice as much. The second candidate explains the data better.

A *model* is a family of distributions p_θ indexed by parameters θ; in the example the parameters are the three probabilities themselves. Reading the probability of the observed sequence as a function of the parameters gives the likelihood.

Definition 8.1 (Likelihood)**.** Let p_θ be a distribution on V for each θ in a set Θ, and let $x_1, \dots, x_N \in V$ be observations. The *likelihood* of θ is

$$\mathcal{L}(\theta) = \prod_{i=1}^{N} p_\theta(x_i),$$

and when $\mathcal{L}(\theta) > 0$ the *log-likelihood* is $\ln \mathcal{L}(\theta) = \sum_{i=1}^{N} \ln p_\theta(x_i)$. A *maximum likelihood estimate* is a θ that maximises $\mathcal{L}$ over Θ.

The likelihood is the probability of the observed sequence when the observations are independent draws from p_θ. It is a function of the parameters, with the data held fixed. The logarithm is increasing, so $\mathcal{L}$ and $\ln \mathcal{L}$ have the same maximisers wherever $\mathcal{L} > 0$; in Example 8.1 the log-likelihoods are $-5 \ln 3 \approx -5.493$ and $\ln 0.00864 \approx -4.751$. Training minimises rather than maximises, and works with averages so that the scale does not depend on N. The loss used throughout the rest of the book is the *average negative log-likelihood*

$$L(\theta) = -\frac{1}{N} \ln \mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} (-\ln p_\theta(x_i)), \tag{8.1}$$

the average, over the observations, of minus the log-probability the model gave each one.

Logarithms are also a practical necessity. The probability of a sequence of 1000 observations, each with probability $1/27$, is $27^{-1000} \approx 10^{-1431}$, far below the smallest positive double-precision number (about 4.9×10^{-324}), so the product rounds to 0 (Appendix A). Its logarithm, $-1000 \ln 27 \approx -3295.8$, is an ordinary number.

8.3. Entropy and divergence

How unpredictable is a distribution? Measure how unexpected each outcome is by $-\ln p(v)$, its *information content* (also called its *surprisal*): 0 for an outcome of probability 1, and large for a rare one. A coin that lands heads with probability 0.9 gives heads information content $-\ln 0.9 \approx 0.105$ and tails $-\ln 0.1 \approx 2.303$. Weighting by the probabilities, its average information content is $0.9 \times 0.105 + 0.1 \times 2.303 \approx 0.325$. A fair coin gives both outcomes $\ln 2 \approx 0.693$, so its average is 0.693: the fair coin is harder to predict.

The definitions below use the natural logarithm and the convention $0 \ln 0 = 0$, which matches the limit $x \ln x \to 0$ as $x \to 0^+$. In each sum, terms with $p(v) = 0$ are 0.

Definition 8.2 (Entropy)**.** The *entropy* of a distribution p on a finite set V is

$$H(p) = -\sum_{v \in V} p(v) \ln p(v) = \mathbb{E}_{X \sim p}[-\ln p(X)].$$

Entropy is the expected information content of an outcome drawn from p. Each term $-p(v) \ln p(v)$ is non-negative, because $p(v) \leq 1$, so $H(p) \geq 0$, and $H(p) = 0$ exactly when one outcome has probability 1 (Exercise 8.1).

Example 8.2. The distribution $(0.6, 0.2, 0.2)$ has entropy $-0.6 \ln 0.6 - 0.4 \ln 0.2 \approx 0.950$. The uniform distribution on n outcomes has entropy $-n \cdot \frac{1}{n} \ln \frac{1}{n} = \ln n$, which is $\ln 3 \approx 1.099$ for three outcomes and $\ln 27 \approx 3.296$ for the vocabulary of the book's model.

A model rarely knows the true distribution. Suppose the data follow $p = (0.6, 0.2, 0.2)$ but a model predicts the uniform distribution q. The model assigns each outcome information content $\ln 3$, so on data drawn from p its average is $0.6 \ln 3 + 0.2 \ln 3 + 0.2 \ln 3 = \ln 3 \approx 1.099$, more than the 0.950 that the model $q = p$ would achieve.

Definition 8.3 (Cross-entropy)**.** The *cross-entropy* of a distribution q relative to a distribution p, both on V, is

$$H(p, q) = -\sum_{v \in V} p(v) \ln q(v) = \mathbb{E}_{X \sim p}[-\ln q(X)],$$

with $H(p, q) = +\infty$ if $q(v) = 0$ for some v with $p(v) > 0$.

If data are drawn from p and a model predicts q, then $H(p, q)$ is the average information content that the model assigns to the data. In particular, if

$\hat{p}$ is the *empirical distribution* of observations $x_1, \dots, x_N$, the fraction $\hat{p}(v) = c_v/N$ of them equal to v, where c_v is the number of observations equal to v, then

$$H(\hat{p}, q) = -\sum_{v \in V} \frac{c_v}{N} \ln q(v) = -\frac{1}{N} \sum_{i=1}^{N} \ln q(x_i), \tag{8.2}$$

because $\ln q(v)$ appears once in the second sum for each observation equal to v. So the loss Equation 8.1 is the cross-entropy of the model relative to the empirical distribution.

The excess of the cross-entropy over the entropy, $1.099 - 0.950 \approx 0.148$ in the example above, measures what the model loses by predicting q instead of p.

Definition 8.4 (Kullback–Leibler divergence)**.** The *Kullback–Leibler divergence* of q from p is

$$D_{\mathrm{KL}}(p \,\|\, q) = \sum_{v:\, p(v) > 0} p(v) \ln \frac{p(v)}{q(v)},$$

with $D_{\mathrm{KL}}(p \,\|\, q) = +\infty$ if $q(v) = 0$ for some v with $p(v) > 0$. Whenever it is finite, $D_{\mathrm{KL}}(p \,\|\, q) = H(p, q) - H(p)$.

The divergence measures the extra average information content that comes from using q when the data follow p. It is not symmetric: for $p = (0.6, 0.2, 0.2)$ and the uniform u on three outcomes, $D_{\mathrm{KL}}(p \,\|\, u) \approx 0.148$ and $D_{\mathrm{KL}}(u \,\|\, p) \approx 0.145$. Its central property rests on an inequality for the logarithm.

Lemma 8.1. *For every $x > 0$, $\ln x \le x - 1$, with equality only at $x = 1$.*

Proof. Let $h(x) = x - 1 - \ln x$ on $(0, \infty)$. Then $h(1) = 0$ and $h'(x) = 1 - 1/x$, which is negative for $x < 1$ and positive for $x > 1$. For $x \neq 1$, the mean

value theorem gives a ξ strictly between 1 and x with $h(x) = h(x) - h(1) = h'(\xi)(x-1)$. Since $\xi - 1$ has the same sign as $x - 1$, so does $h'(\xi)$, and the product is positive. So $h(x) > 0$ for $x \neq 1$. □

Theorem 8.1 (Gibbs' inequality)**.** *For distributions p and q on a finite set, $D_{\mathrm{KL}}(p \,\|\, q) \geq 0$, with equality if and only if $p = q$.*

Proof. If $q(v) = 0$ for some v with $p(v) > 0$ the divergence is $+\infty$ and $p \neq q$, so there is nothing to prove. Otherwise let $S = \{v : p(v) > 0\}$; then $q(v) > 0$ on S. By Lemma 8.1 with $x = q(v)/p(v)$,

$$-D_{\mathrm{KL}}(p \,\|\, q) = \sum_{v \in S} p(v) \ln \frac{q(v)}{p(v)} \leq \sum_{v \in S} p(v) \Big(\frac{q(v)}{p(v)} - 1 \Big) = \sum_{v \in S} q(v) - 1 \leq 0, \tag{8.3}$$

because $\sum_{v \in S} p(v) = 1$ and $\sum_{v \in S} q(v) \leq 1$. So $D_{\mathrm{KL}}(p \,\|\, q) \geq 0$.

If $p = q$, every logarithm in the definition is $\ln 1 = 0$. If $D_{\mathrm{KL}}(p \,\|\, q) = 0$, both inequalities in Equation 8.3 are equalities. The first one is a sum of term-by-term inequalities with positive weights $p(v)$, so each term is an equality, and by Lemma 8.1 $q(v)/p(v) = 1$ for every $v \in S$. The second gives $\sum_{v \in S} q(v) = 1$, so $q(v) = 0 = p(v)$ for $v \notin S$. Hence $q = p$. □

Corollary 8.1. *For every distribution p on a set V with n elements, $0 \leq H(p) \leq \ln n$, with $H(p) = \ln n$ only for the uniform distribution. For every distribution q on V, $H(p, q) \geq H(p)$, with equality only for $q = p$.*

Proof. $H(p) \geq 0$ was shown after Definition 8.2. For the uniform distribution u, $D_{\mathrm{KL}}(p \,\|\, u) = \sum_v p(v) \ln(np(v)) = \ln n - H(p)$, which is non-negative, and zero only if $p = u$, by Theorem 8.1. For the last statement, if $H(p, q) = +\infty$ there is nothing to prove, and otherwise $H(p, q) - H(p) = D_{\mathrm{KL}}(p \,\|\, q)$. □

8.4. Maximum likelihood for a categorical distribution

The simplest model of a sequence of observations from V treats them as independent draws from an unknown distribution q on V, with the $|V|$ values of q as parameters. In Example 8.1 the candidate $(0.6, 0.2, 0.2)$, which beat the uniform distribution, is the list of *relative frequencies* $\frac{3}{5}, \frac{1}{5}, \frac{1}{5}$ of the three outcomes, the fraction of the observations equal to each. Moving towards the uniform distribution, to $(0.5, 0.25, 0.25)$, lowers the likelihood to $0.5^3 \times 0.25^2 \approx 0.0078$. No other candidate does better than the relative frequencies.

Theorem 8.2 (Maximum likelihood is relative frequency)**.** *Let $x_1, \dots, x_N$ be observations in a finite set V, let c_v be the number of them equal to v, and let $\hat{p}(v) = c_v/N$. Over all distributions q on V, the likelihood $\mathcal{L}(q) = \prod_{i=1}^{N} q(x_i)$ is largest at $q = \hat{p}$ and nowhere else. Equivalently, $\hat{p}$ is the unique minimiser of the loss $H(\hat{p}, q)$.*

Proof. Group the factors of the likelihood by outcome: $\mathcal{L}(q) = \prod_{v \in V} q(v)^{c_v}$. At $q = \hat{p}$ every factor with $c_v > 0$ is positive, so $\mathcal{L}(\hat{p}) > 0$. If $q(v) = 0$ for some v with $c_v > 0$ then $\mathcal{L}(q) = 0 < \mathcal{L}(\hat{p})$. Otherwise $\mathcal{L}(q) > 0$ and, by Equation 8.2 and Definition 8.4,

$$\ln \mathcal{L}(q) = \sum_{v \in V} c_v \ln q(v) = -N\, H(\hat{p}, q) = -N\big(H(\hat{p}) + D_{\mathrm{KL}}(\hat{p} \,\|\, q)\big).$$

By Theorem 8.1 the right side is at most $-N\, H(\hat{p})$, which is its value at $q = \hat{p}$, with equality only when $q = \hat{p}$. Since ln is increasing, the same holds for $\mathcal{L}$. □

In Example 8.1 the counts are $3, 1, 1$, so the maximum likelihood estimate is $\hat{p} = (0.6, 0.2, 0.2)$, and the largest possible likelihood is $0.00864 = e^{-5H(\hat{p})}$. The proof uses no calculus. Setting derivatives to zero would have to respect the constraint $\sum_v q(v) = 1$ (with a Lagrange multiplier, a method

for constrained optimisation that the book does not use), and would not show on its own that the critical point is a maximum. Theorem 9.2 applies the theorem once for each preceding character.

8.5. The softmax function

A network produces a vector of real numbers, and a probabilistic model needs a distribution: non-negative numbers with sum 1. Take the scores $(2, 1, 0)$. Exponentiating makes every score positive, giving about 7.389, 2.718 and 1, and dividing by their sum 11.107 gives $(0.665, 0.245, 0.090)$, a distribution in which a higher score means a higher probability. The obvious alternative, dividing each score by the sum of the scores, fails: $(2, -1)$ would give $(2, -1)$, which has a negative entry, and $(1, -1)$ has sum 0.

Definition 8.5 (Softmax)**.** The *softmax* of $\mathbf{z} \in \mathbb{R}^n$ is the vector $\operatorname{softmax}(\mathbf{z}) \in \mathbb{R}^n$ with entries

$$\operatorname{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}.$$

The entries z_i of the input are called *logits*.

Every entry of $\mathbf{p} = \operatorname{softmax}(\mathbf{z})$ is positive and the entries sum to 1, so $\mathbf{p}$ is a distribution on $\{1, \dots, n\}$ that gives every outcome some probability. The exponential is increasing, so softmax preserves order: $p_i > p_j$ exactly when $z_i > z_j$. Every distribution with positive entries arises, from $z_i = \ln p_i$, because then $\sum_j e^{z_j} = 1$. For $n = 2$, dividing numerator and denominator by e^{z_1} gives $\operatorname{softmax}(\mathbf{z})_1 = 1/(1 + e^{-(z_1 - z_2)})$, the logistic function of the difference of the logits. Solving for that difference gives $z_1 - z_2 = \ln(p_1/(1 - p_1))$, the *logit* of p_1 in statistics, which is where the name comes from.

Adding the same number c to every logit multiplies the numerator and the denominator by e^c, so

$$\operatorname{softmax}(\mathbf{z} + c\mathbf{1}) = \operatorname{softmax}(\mathbf{z}), \tag{8.4}$$

where $\mathbf{1} = (1, \dots, 1)^\top$. Only differences of logits matter. Exercise 8.3 shows that these are the only changes of $\mathbf{z}$ that leave the softmax alone. The identity has a practical use: computing e^{1000} overflows in double precision, but subtracting the largest logit first makes every exponent at most 0. Theorem A.3 and Appendix A analyse that computation.

Example 8.3. The computation at the start of this section says $\operatorname{softmax}(2, 1, 0) \approx (0.665, 0.245, 0.090)$. The logits $(1002, 1001, 1000)$ give the same distribution by Equation 8.4, although evaluating e^{1002} directly gives infinity in double precision.

The logarithm of the denominator appears so often that it has a name.

Definition 8.6 (Log-sum-exp)**.** The *log-sum-exp* function is $\operatorname{lse}(\mathbf{z}) = \ln \sum_{j=1}^{n} e^{z_j}$ for $\mathbf{z} \in \mathbb{R}^n$.

For $\mathbf{z} = (2, 1, 0)$, $\operatorname{lse}(\mathbf{z}) = \ln 11.107 \approx 2.408$, a little above the largest logit 2. From the definitions,

$$\ln \operatorname{softmax}(\mathbf{z})_i = z_i - \operatorname{lse}(\mathbf{z}). \tag{8.5}$$

With $m = \max_j z_j$, the sum $\sum_j e^{z_j}$ lies between e^m (one of its terms) and ne^m (its n terms are each at most e^m), so

$$\max_j z_j \le \operatorname{lse}(\mathbf{z}) \le \max_j z_j + \ln n. \tag{8.6}$$

Log-sum-exp is a smooth version of the maximum. The bound also gives the stable way to compute it, $\operatorname{lse}(\mathbf{z}) = m + \ln \sum_j e^{z_j - m}$, in which the sum lies between 1 and n.

Theorem 8.3. *Log-sum-exp is differentiable on $\mathbb{R}^n$, and $\nabla \operatorname{lse}(\mathbf{z}) = \operatorname{softmax}(\mathbf{z})$.*

Proof. By the chain rule for one variable, $\partial \operatorname{lse} / \partial z_k = e^{z_k} / \sum_j e^{z_j}$, which is continuous on $\mathbb{R}^n$. By Theorem 1.6, lse is differentiable, and by Theorem 1.5 its gradient is the vector of these partial derivatives. □

How does the distribution respond when one logit changes? At $\mathbf{z} = (2, 1, 0)$, increase z_1 by 0.01. The probabilities change by about $(0.00222, -0.00163, -0.00060)$: p_1 rises, and the other two fall, by amounts proportional to their own size, since $0.00163/0.245 \approx 0.00060/0.090 \approx 0.0066$. The increases and decreases cancel, because the probabilities still sum to 1. The next theorem gives these rates exactly: the change is about $0.01\, p_1(\mathbf{e}_1 - \mathbf{p})$.

Theorem 8.4 (Jacobian of softmax)**.** *Softmax is differentiable on $\mathbb{R}^n$. For $\mathbf{z} \in \mathbb{R}^n$ and $\mathbf{p} = \operatorname{softmax}(\mathbf{z})$,*

$$\frac{\partial p_i}{\partial z_k} = p_i(\delta_{ik} - p_k), \qquad \textit{that is,} \qquad J_{\operatorname{softmax}}(\mathbf{z}) = \operatorname{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top,$$

where δ_{ik} is 1 if $i = k$ and 0 otherwise. The matrix $J = J_{\operatorname{softmax}}(\mathbf{z})$ has these properties:

(a) J is symmetric;

(b) $J\mathbf{1} = \mathbf{0}$: each row sums to 0, so J is singular;

(c) $\mathbf{v}^\top J \mathbf{v} = \sum_i p_i v_i^2 - \left(\sum_i p_i v_i\right)^2 \geq 0$ for every $\mathbf{v}$, with equality exactly when $\mathbf{v}$ is a multiple of $\mathbf{1}$. So J is positive semidefinite in the sense of Definition 2.5, its eigenvalues are real, 0 is an eigenvalue whose eigenvectors are the non-zero multiples of $\mathbf{1}$, and every other eigenvalue is positive.

Column k of J lists how every probability responds to an increment of the logit z_k: p_k rises at rate $p_k(1-p_k)$ and each other p_i falls at rate $p_i p_k$. Row i lists how p_i responds to each logit. Entry (i,k) is the rate of change of p_i per unit increment of z_k.

Proof. Write $S = \sum_j e^{z_j}$, so $p_i = e^{z_i}/S$ and $\partial S/\partial z_k = e^{z_k}$. By the quotient rule,

$$\frac{\partial p_i}{\partial z_k} = \frac{\delta_{ik} e^{z_i} S - e^{z_i} e^{z_k}}{S^2} = \delta_{ik} p_i - p_i p_k.$$

These partial derivatives are continuous, so softmax is differentiable by Theorem 1.6, and Theorem 1.5 identifies J. Entry (i,k) of $\operatorname{diag}(\mathbf{p})$ is $\delta_{ik} p_i$ and entry (i,k) of $\mathbf{p}\mathbf{p}^\top$ is $p_i p_k$.

(a) Both $\operatorname{diag}(\mathbf{p})$ and $\mathbf{p}\mathbf{p}^\top$ are symmetric.

(b) $J\mathbf{1} = \mathbf{p} - \mathbf{p}(\mathbf{p}\cdot\mathbf{1}) = \mathbf{p} - \mathbf{p} = \mathbf{0}$, since the entries of $\mathbf{p}$ sum to 1.

(c) $\mathbf{v}^\top \operatorname{diag}(\mathbf{p})\mathbf{v} = \sum_i p_i v_i^2$ and $\mathbf{v}^\top \mathbf{p}\mathbf{p}^\top \mathbf{v} = (\mathbf{p}\cdot\mathbf{v})^2$. Apply Theorem 1.1 to the vectors $\mathbf{a}$ and $\mathbf{b}$ with $a_i = \sqrt{p_i}$ and $b_i = \sqrt{p_i}\, v_i$:

$$\Big(\sum_i p_i v_i\Big)^2 = (\mathbf{a}\cdot\mathbf{b})^2 \le \|\mathbf{a}\|^2 \|\mathbf{b}\|^2 = 1 \cdot \sum_i p_i v_i^2.$$

Equality holds exactly when one of $\mathbf{a}$, $\mathbf{b}$ is a multiple of the other. Since $\mathbf{a} \neq \mathbf{0}$, that means $\mathbf{b} = c\mathbf{a}$ for some c, that is, $\sqrt{p_i} v_i = c\sqrt{p_i}$ for all i, and since every $p_i > 0$, $v_i = c$ for all i.

For the eigenvalues, let $J\mathbf{v} = \lambda\mathbf{v}$ with $\mathbf{v} \neq \mathbf{0}$, allowing complex λ and complex entries in $\mathbf{v}$, and let $\bar{\mathbf{v}}$ be the vector of complex conjugates. The number $\bar{\mathbf{v}}^\top J\mathbf{v}$ equals its own conjugate $\mathbf{v}^\top J\bar{\mathbf{v}}$, because J is real and symmetric, so it is real. It also equals $\lambda \sum_i |v_i|^2$, so λ is real, and then $\mathbf{v}$ can be taken real (the real or the imaginary part of a complex eigenvector is a real one). For real $\mathbf{v}$, $\mathbf{v}^\top J\mathbf{v} = \lambda\|\mathbf{v}\|^2$, so $\lambda \ge 0$, with $\lambda = 0$ only when $\mathbf{v}$ is a multiple of $\mathbf{1}$. □

The quantity in (c) has a probabilistic reading: if I is a random index with distribution $\mathbf{p}$, then $\mathbf{v}^\top J\mathbf{v} = \mathbb{E}[v_I^2] - (\mathbb{E}[v_I])^2 = \operatorname{Var}[v_I]$. By Theorem 8.3, J is also the Hessian of log-sum-exp, the matrix of its second partial derivatives (Definition 2.3), so (c) says that the Hessian of lse is positive semidefinite everywhere. Property (b) restates Equation 8.4: moving $\mathbf{z}$ along $\mathbf{1}$ does not change $\mathbf{p}$, so the derivative in that direction is $\mathbf{0}$.

For $n = 2$, $J = p_1 p_2 \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$, with eigenvalues 0 and $2p_1p_2$; with $p_1 = \sigma(z_1 - z_2)$ for the logistic function σ, entry $(1,1)$ is the familiar $\sigma' = \sigma(1-\sigma)$.

8.6. Cross-entropy loss for a softmax model

A *classifier* is a model that assigns an input to one of n classes, numbered $1, \dots, n$; a language model predicting the next token is one, with the 27 tokens as classes. It computes logits $\mathbf{z}$, turns them into probabilities $\mathbf{p} = \operatorname{softmax}(\mathbf{z})$, and is scored on the observed class y, the target, by the information content $-\ln p_y$. By Equation 8.1, the loss on a data set is the average of these numbers, so the gradient of the loss is the average of their gradients. The target y corresponds to the distribution $\mathbf{e}_y$ that gives probability 1 to class y and 0 to every other class; a vector with one entry 1 and the others 0 is called *one-hot*. In this notation $-\ln p_y = H(\mathbf{e}_y, \mathbf{p})$, the cross-entropy of the prediction relative to the one-hot target.

With the logits $(2, 1, 0)$ and observed outcome $y = 2$, the loss is $-\ln 0.245 \approx 1.408$. Increasing z_2 by 0.01 lowers it by about $0.0075 \approx 0.01(1 - p_2)$, and increasing z_1 by 0.01 raises it by about $0.0067 \approx 0.01\, p_1$. So the partial derivatives appear to be p_i for the wrong outcomes and $p_y - 1$ for the observed one.

Theorem 8.5 (Gradient of the cross-entropy loss)**.** *For $y \in \{1, \dots, n\}$ let $L(\mathbf{z}) = -\ln \operatorname{softmax}(\mathbf{z})_y$. Then $L(\mathbf{z}) = \operatorname{lse}(\mathbf{z}) - z_y$, L is differentiable on*

$\mathbb{R}^n$, *and*

$$\nabla L(\mathbf{z}) = \mathbf{p} - \mathbf{e}_y, \qquad \mathbf{p} = \operatorname{softmax}(\mathbf{z}).$$

More generally, for any distribution $\mathbf{y}$ *on* $\{1, \dots, n\}$*, the loss* $L(\mathbf{z}) = H(\mathbf{y}, \operatorname{softmax}(\mathbf{z})) = -\sum_i y_i \ln p_i$ *has gradient* $\mathbf{p} - \mathbf{y}$*; the first case is the one-hot target* $\mathbf{y} = \mathbf{e}_y$*.*

Proof. By Equation 8.5, $-\sum_i y_i \ln p_i = \sum_i y_i(\operatorname{lse}(\mathbf{z}) - z_i) = \operatorname{lse}(\mathbf{z}) - \mathbf{y} \cdot \mathbf{z}$, since $\sum_i y_i = 1$. For $\mathbf{y} = \mathbf{e}_y$ this is $\operatorname{lse}(\mathbf{z}) - z_y$. By Theorem 8.3 and Example 1.6, L is differentiable with gradient $\operatorname{softmax}(\mathbf{z}) - \mathbf{y}$. □

The same result follows from Theorem 8.4 by the chain rule: $\nabla L = -\frac{1}{p_y}\nabla p_y = -\frac{1}{p_y} p_y(\mathbf{e}_y - \mathbf{p}) = \mathbf{p} - \mathbf{e}_y$, using row y of J. The proof above is shorter because log-sum-exp absorbs the quotient.

The gradient is simple to read. A gradient descent step with learning rate $\eta > 0$ (Chapter 5) changes the logits by $-\eta(\mathbf{p} - \mathbf{e}_y)$: it raises the logit of the observed outcome by $\eta(1 - p_y)$ and lowers every other logit z_i by ηp_i, so the outcomes that took the most probability away from y lose the most. Every component lies strictly between -1 and 1, and the components sum to 0, as (b) of Theorem 8.4 requires. The gradient is never $\mathbf{0}$, since $p_y < 1$, and $L(\mathbf{z}) > 0$ for every $\mathbf{z}$; the loss tends to 0 only as $z_y - z_i \to \infty$ for every $i \neq y$. In the notation of Theorem 7.1, an MLP whose output $\mathbf{z}_D$ is read as logits has $\delta_D = \mathbf{p} - \mathbf{e}_y$.

Example 8.4. Take the logits $\mathbf{z} = (2, 1, 0)$ of Example 8.3 and observed outcome $y = 2$. Then $L = -\ln 0.245 \approx 1.408$ and $\nabla L \approx (0.665, -0.755, 0.090)$. A gradient step with $\eta = 1$ moves the logits to about $(1.335, 1.755, -0.090)$, where the probability of outcome 2 is about 0.55 and the loss about 0.60.

Example 8.5 (The loss of a uniform model)**.** If all logits are equal, $\mathbf{p}$ is uniform and $L = -\ln(1/n) = \ln n$ whatever the observed outcome. For the book's vocabulary, $\ln 27 \approx 3.30$. A model whose average loss is above $\ln |V|$ does worse than one that predicts nothing at all.

The book's model starts close to the uniform model. Its last layer applies a 27×16 matrix with entries drawn from $N(0, 0.08^2)$ to a vector of squared norm about 16, so by Theorem 7.3 each logit has mean 0 and variance about $0.08^2 \times 16 \approx 0.10$, a standard deviation of about 0.32. The second-order Taylor expansion (Theorem 2.6) approximates a function near a point $\mathbf{a}$ by $f(\mathbf{a}+\mathbf{h}) \approx f(\mathbf{a}) + \nabla f(\mathbf{a}) \cdot \mathbf{h} + \frac{1}{2}\mathbf{h}^\top \nabla^2 f(\mathbf{a})\, \mathbf{h}$. At a constant vector $c\mathbf{1}$ the softmax is uniform, so by Theorem 8.3 and Theorem 8.4 log-sum-exp has gradient $\frac{1}{n}\mathbf{1}$ and Hessian $\frac{1}{n}I - \frac{1}{n^2}\mathbf{1}\mathbf{1}^\top$ there. Taking $c = \bar{z}$, the average defined below, the expansion gives

$$\operatorname{lse}(\mathbf{z}) \approx \ln n + \bar{z} + \frac{1}{2} \cdot \frac{1}{n} \sum_i (z_i - \bar{z})^2, \qquad \bar{z} = \frac{1}{n} \sum_i z_i.$$

The loss is $\operatorname{lse}(\mathbf{z}) - z_y$, and every logit has mean 0, so $\mathbb{E}[\bar{z} - z_y] = 0$ and the expected loss is about $\ln n$ plus half the expected value of $\frac{1}{n}\sum_i (z_i - \bar{z})^2$. For a fixed input vector the rows of the matrix are independent, so the logits are independent, each with mean 0 and variance $\sigma^2 \approx 0.10$. Then $\frac{1}{n}\sum_i (z_i - \bar{z})^2 = \frac{1}{n}\sum_i z_i^2 - \bar{z}^2$, $\mathbb{E}[z_i^2] = \sigma^2$ and $\mathbb{E}[\bar{z}^2] = \operatorname{Var}[\bar{z}] = \sigma^2/n$, so its expected value is $\frac{n-1}{n}\sigma^2$. The expected loss is about $\ln 27 + \frac{1}{2} \cdot \frac{26}{27} \cdot 0.10 \approx 3.35$. Averaged over 2000 freshly initialised models, each run on a random sequence of tokens, the simulated initial loss is also 3.35. The Taylor expansion gives no bound on its own error, and Theorem 11.5 in Chapter 11 replaces the estimate by inequalities.

8.7. Where squared error comes from

Not every loss in machine learning is a cross-entropy, but most are negative log-likelihoods of some model. The squared error is the standard example. Suppose a real-valued observation y is modelled as $f(\mathbf{x}; \theta)$ plus a random error, the *noise*, with the normal distribution $N(0, s^2)$ of Definition 7.4, whose density is $\varphi(t) = (2\pi s^2)^{-1/2} e^{-t^2/(2s^2)}$. Then y has density $\varphi(t - f(\mathbf{x}; \theta))$.

The likelihood of Definition 8.1 multiplies the probabilities of the observations, and for a variable with a density the probability of each exact value is 0, so that product would be 0 for every θ. The density is the right replacement. For a small $\varepsilon > 0$, the probability that y lands within ε of an observed value y_i is $\int_{y_i-\varepsilon}^{y_i+\varepsilon} \varphi(t-f)\,dt \approx 2\varepsilon\,\varphi(y_i-f)$, so maximising the product of the densities maximises, approximately, the probability of landing that close to every observation; the factor $(2\varepsilon)^N$ does not depend on θ. For continuous observations the likelihood therefore uses the density in place of the probability (Wasserman 2004, sec. 9.3), and for independent observations $(\mathbf{x}_i, y_i)$, $i = 1, \dots, N$,

$$-\ln \mathcal{L}(\theta) = \frac{1}{2s^2} \sum_{i=1}^{N} \big(y_i - f(\mathbf{x}_i;\theta)\big)^2 + \frac{N}{2} \ln(2\pi s^2).$$

For fixed s, maximising the likelihood is the same as minimising the sum of squared errors, because the two differ by a positive factor and a constant. The choice of loss is a choice of noise model: normal noise gives squared error, and a categorical model with softmax outputs gives cross-entropy.

8.8. Notes and further reading

MacKay (2003, ch. 2) introduces entropy, relative entropy and Gibbs' inequality, which it proves in sec. 2.6, and Cover and Thomas (2006, ch. 2) gives a fuller treatment of the same quantities. Goodfellow et al. (2016, ch. 3) collects the probability and information theory used in deep learning, [ch. 5] discusses maximum likelihood, and [ch. 6] discusses softmax output units and the cross-entropy loss. Wasserman (2004, ch. 9) covers maximum likelihood estimation in general, and Deisenroth et al. (2020, ch. 9) derives least squares from a normal noise model as in Section 8.7.

8.9. Exercises

Exercise 8.1.

(a) Show that $H(p) = 0$ if and only if $p(v) = 1$ for some v.

(b) For $p = (\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$ and the uniform u on three outcomes, compute $H(p)$, $D_{\mathrm{KL}}(p \,\|\, u)$ and $D_{\mathrm{KL}}(u \,\|\, p)$ exactly.

Exercise 8.2. A coin lands heads with unknown probability θ. In N tosses it lands heads k times, with $0 < k < N$. Maximise $\theta^k (1-\theta)^{N-k}$ over $0 \le \theta \le 1$ by calculus, and check the answer against Theorem 8.2. What happens when $k = 0$?

Exercise 8.3.

(a) Show that $\operatorname{softmax}(\mathbf{z}) = \operatorname{softmax}(\mathbf{z}')$ if and only if $\mathbf{z} - \mathbf{z}' = c\mathbf{1}$ for some real c.

(b) Find the logits $\mathbf{z}$ with $z_1 = 0$ and $\operatorname{softmax}(\mathbf{z}) = (\frac{1}{2}, \frac{1}{3}, \frac{1}{6})$.

Exercise 8.4. Let $\mathbf{z} = (0, \ln 2, \ln 3)$. Find $\mathbf{p} = \operatorname{softmax}(\mathbf{z})$ and $J_{\operatorname{softmax}}(\mathbf{z})$, check that its rows sum to 0, and compute $\mathbf{v}^{\top} J \mathbf{v}$ for $\mathbf{v} = (1, 0, 0)$ and for $\mathbf{v} = (1, 1, 1)$.

Exercise 8.5. Let $\sigma(t) = 1/(1 + e^{-t})$, and for an observed class $y \in \{0, 1\}$ (often called the *label*) let $L(t) = -y \ln \sigma(t) - (1-y) \ln(1 - \sigma(t))$, the *binary cross-entropy*.

(a) Show that $\sigma'(t) = \sigma(t)(1 - \sigma(t))$ and that $1 - \sigma(t) = \sigma(-t)$.

(b) Show that $L'(t) = \sigma(t) - y$, and explain how this is the case $n = 2$ of Theorem 8.5.

Exercise 8.6. Let $\mathbf{y}$ be a distribution on $\{1, \dots, n\}$ with every $y_i > 0$, and $L(\mathbf{z}) = H(\mathbf{y}, \operatorname{softmax}(\mathbf{z}))$.

(a) Using Corollary 8.1, show that L attains its minimum value $H(\mathbf{y})$, exactly at the $\mathbf{z}$ with $\operatorname{softmax}(\mathbf{z}) = \mathbf{y}$, and find those $\mathbf{z}$.

(b) Show that for the one-hot target $\mathbf{e}_y$ the infimum of L is 0 but is not attained.

(c) *Label smoothing* replaces $\mathbf{e}_y$ by $\mathbf{y} = (1-\alpha)\mathbf{e}_y + \frac{\alpha}{n}\mathbf{1}$ with $0 < \alpha < 1$. For $n = 27$ and $\alpha = 0.1$, find the difference $z_y - z_i$ $(i \neq y)$ at the minimiser.

Exercise 8.7. Observations $y_1, \dots, y_N$ are modelled as independent draws from $N(\mu, s^2)$, with $s > 0$ and μ unknown.

(a) Show that the negative log-likelihood, as a function of μ for fixed s, is minimised at the mean $\bar{y} = \frac{1}{N}\sum_i y_i$.

(b) Suppose the y_i are not all equal, and let s vary too. Show that at $\mu = \bar{y}$ the negative log-likelihood is minimised over $s > 0$ at $s^2 = \frac{1}{N}\sum_i (y_i - \bar{y})^2$.

Exercise 8.8. Prove that for every $\mathbf{z} \in \mathbb{R}^n$ and every y,

$$-\ln \operatorname{softmax}(\mathbf{z})_y \geq \ln n + \bar{z} - z_y, \qquad \bar{z} = \frac{1}{n}\sum_i z_i,$$

with equality when all z_i are equal. (Use $\ln x \leq x - 1$, or compare the arithmetic and geometric means of the e^{z_i}.) Deduce that if the logits of a randomly initialised model satisfy $\mathbb{E}[z_i] = \mathbb{E}[z_y]$ for all i, its expected loss is at least $\ln n$.

Part V.

Language models

9. Language models

A language model assigns probabilities to sequences of symbols. The model this book builds reads a name one character at a time and, after each character, gives a probability for every character that could come next. Multiplying those probabilities along a name gives the probability of the whole name, and training adjusts the model's parameters to make the names in a list of real names probable.

This chapter sets up the probability behind that description and studies the simplest language model, in which the next character depends only on the current one. For that model the maximum-likelihood parameters have a closed form: they are relative frequencies of pairs, and we can compute them by counting. The same model written as a small neural network introduces embeddings, the vectors that represent tokens in every later model, and shows how a neural model shares information between tokens. The chapter ends with the problem that counting cannot solve, predicting from many earlier characters at once, which Chapter 10 takes up.

9.1. Tokens and vocabularies

A model computes with numbers and vectors, so text has to be cut into pieces from a fixed, finite list before the model can see it, and the model needs a way to tell where a piece of text starts and stops.

Take the name `emma`. Cut into characters it is `e`, `m`, `m`, `a`. A model that reads it one character at a time must also learn which characters names start

with and when a name ends, so we add a marker at both ends, written ⋄:

$$(\diamond, \texttt{e}, \texttt{m}, \texttt{m}, \texttt{a}, \diamond).$$

The model reads this sequence from the left and, at each position, predicts the next symbol. From the leading ⋄ it predicts the first letter `e`; from ⋄`e` it predicts `m`; and from the whole name it predicts the closing ⋄, which says that the name ends there. That is five predictions for a name of four letters. The first ⋄ is never predicted; it only starts the sequence.

Definition 9.1 (Vocabulary and tokens)**.** A *vocabulary* is a finite set V whose elements are called *tokens*; $|V|$ is the number of tokens. A *tokeniser* turns a piece of text into a finite sequence of tokens. A *character-level* tokeniser makes each character a token. The *boundary token* ⋄ is a token that marks the start and the end of a piece of text.

For names written in the 26 lower-case letters, the vocabulary is the 26 letters together with ⋄, so $|V| = 27$. A name of n letters becomes $n + 2$ tokens and gives $n + 1$ predictions.

Programs refer to tokens by number. The microgpt program that this book's model follows (Karpathy 2026a) numbers the letters $\texttt{a}, \ldots, \texttt{z}$ as $0, \ldots, 25$ and gives ⋄ the number 26, so `emma` becomes $(26, 4, 12, 12, 0, 26)$. The numbers are labels. Nothing in a model may depend on `b` having a number one larger than `a`, and Section 9.5 replaces each number by a vector that the model learns.

Character-level tokenisation keeps the vocabulary small. Large models for general text use tokens that are pieces of words, found by merging frequent pairs of symbols; GPT-2, for example, has a vocabulary of 50,257 tokens (Radford et al. 2019). Nothing in this chapter or the next depends on how the tokens were chosen.

The examples in this chapter use five names as a training *corpus*, the collection of text a model learns from:

`anna`, `ava`, `emma`, `eva`, `ada`.

To keep tables small, the examples use the vocabulary of the symbols that occur in these names, $V = \{\diamond, \mathtt{a}, \mathtt{d}, \mathtt{e}, \mathtt{m}, \mathtt{n}, \mathtt{v}\}$, with $|V| = 7$. The names have 17 letters between them, so they give $17 + 5 = 22$ predictions.

9.2. The probability of a sequence

A language model gives a probability to a whole name by giving a probability to each prediction and multiplying. This section shows why multiplying is right. The tool is conditional probability, the probability of one event once we know that another has happened.

Choose one of the five corpus names at random, each with probability 1/5. The probability that the name starts with `a` is 3/5 (`anna`, `ava`, `ada`). Once we know that it starts with `a`, only those three names remain, and one of them, `ava`, has `v` second, so the probability of `v` second is now 1/3. Multiplying gives $\frac{3}{5} \cdot \frac{1}{3} = \frac{1}{5}$, which is the probability that the name begins `av`, counted directly: one name in five. The probability of the first two letters is the probability of the first, times the probability of the second given the first.

We work on a finite probability space (Definition 6.1), with random variables that take values in the vocabulary, as in Section 8.1. In the example, Ω is the set of the five names, each with probability 1/5, and "starts with `a`" is the event $\{\mathtt{anna}, \mathtt{ava}, \mathtt{ada}\}$. The first letter X_1 and the second letter X_2 of the chosen name are random variables with values in the vocabulary. As in Definition 6.2, $P(X_1 = x_1, \dots, X_t = x_t)$ is the probability of the event $\{X_1 = x_1\} \cap \cdots \cap \{X_t = x_t\}$ that all t statements hold.

Definition 9.2 (Conditional probability)**.** Let A and B be events in a finite probability space with $P(A) > 0$. The *conditional probability* of B given A is

$$P(B \mid A) = \frac{P(A \cap B)}{P(A)}. \tag{9.1}$$

In words, $P(B \mid A)$ is the probability of B when we keep only the outcomes in A and rescale their probabilities so that they sum to 1. In the example, A is "starts with `a`", B is "has `v` second", $P(A \cap B) = 1/5$ and $P(A) = 3/5$, so $P(B \mid A) = 1/3$. Rearranged, Equation 9.1 says $P(A \cap B) = P(A)\,P(B \mid A)$, and applying this once per position breaks the probability of a sequence into one factor per position.

Theorem 9.1 (Chain rule of probability)**.** *Let $X_1, \dots, X_T$ be random variables on a finite probability space, with values in a vocabulary V, and let $x_1, \dots, x_T \in V$ satisfy $P(X_1 = x_1, \dots, X_{T-1} = x_{T-1}) > 0$. Then*

$$P(X_1 = x_1, \dots, X_T = x_T) = \prod_{t=1}^{T} P(X_t = x_t \mid X_1 = x_1, \dots, X_{t-1} = x_{t-1}), \tag{9.2}$$

where the factor for $t = 1$ is $P(X_1 = x_1)$.

The probability of the whole sequence is the product, over the positions, of the probability of the token at that position given all the tokens before it.

Proof. Let $A_0 = \Omega$ and, for $t \geq 1$, $A_t = \{X_1 = x_1\} \cap \cdots \cap \{X_t = x_t\}$. Then $A_t = A_{t-1} \cap \{X_t = x_t\}$, so $A_t \subseteq A_{t-1}$, and the probabilities $P(A_t)$ decrease as t grows. In particular $P(A_{t-1}) \geq P(A_{T-1}) > 0$ for $t = 1, \dots, T$, so every conditional probability in Equation 9.2 is defined. The factor for t is $P(X_t = x_t \mid A_{t-1})$, and for $t = 1$ this is $P(X_1 = x_1 \mid \Omega) = P(X_1 = x_1)$ because $P(\Omega) = 1$.

We prove by induction on t that $P(A_t) = \prod_{s=1}^{t} P(X_s = x_s \mid A_{s-1})$ for $t = 1, \dots, T$. For $t = 1$ both sides are $P(X_1 = x_1)$. If the statement holds for $t - 1$, then by Equation 9.1,

$$P(A_t) = P(A_{t-1} \cap \{X_t = x_t\}) = P(A_{t-1})\,P(X_t = x_t \mid A_{t-1}) = \prod_{s=1}^{t} P(X_s = x_s \mid A_{s-1}).$$

The case $t = T$ is Equation 9.2. □

If instead $P(X_1 = x_1, \dots, X_{T-1} = x_{T-1}) = 0$, the left side of Equation 9.2 is 0 as well, because $A_T \subseteq A_{T-1}$.

Example 9.1. In the example above, with X_1 and X_2 the first and second letters of a randomly chosen corpus name,

$$P(X_1 = \texttt{a}, X_2 = \texttt{v}) = P(X_1 = \texttt{a})\, P(X_2 = \texttt{v} \mid X_1 = \texttt{a}) = \frac{3}{5} \cdot \frac{1}{3} = \frac{1}{5}.$$

The theorem applies in the other order too: Exercise 9.2 conditions on the second letter first.

By Theorem 9.1, to give a probability to every sequence it is enough to give, for every possible beginning of a sequence, a probability distribution for the next token. That is how every model in this book is built.

Definition 9.3 (Language model)**.** Let V be a vocabulary containing a boundary token $\diamond$. A *language model* over V gives, for each finite sequence $(x_1, \dots, x_t)$ of tokens with $x_1 = \diamond$, a probability distribution $p(\,\cdot \mid x_1, \dots, x_t)$ on V for the next token. The sequence $(x_1, \dots, x_t)$ is the *context*. A *document* (a name, in our examples) is a finite sequence $w = (w_1, \dots, w_n)$ of tokens other than $\diamond$. It is wrapped as $x = (x_1, \dots, x_{n+2}) = (\diamond, w_1, \dots, w_n, \diamond)$ and receives the probability

$$P(w) = \prod_{t=1}^{n+1} p(x_{t+1} \mid x_1, \dots, x_t). \tag{9.3}$$

Position t of the wrapped sequence holds the token x_t, and the prediction made at position t is the distribution of x_{t+1}, so positions and predictions are both numbered $t = 1, \dots, n+1$. Positions are numbered the same way in Chapter 10 and Chapter 11.

The product has one factor for each of the $n+1$ predictions: the probability that the model gives to the token that actually comes next, given everything before it. For `emma` the five factors are

$$p(\texttt{e} \mid \diamond), \quad p(\texttt{m} \mid \diamond\,\texttt{e}), \quad p(\texttt{m} \mid \diamond\,\texttt{e}\,\texttt{m}), \quad p(\texttt{a} \mid \diamond\,\texttt{e}\,\texttt{m}\,\texttt{m}), \quad p(\diamond \mid \diamond\,\texttt{e}\,\texttt{m}\,\texttt{m}\,\texttt{a}),$$

one for each prediction listed in Section 9.1. The model's distributions play the part of the conditional probabilities in the chain rule (Equation 9.2). A model *generates* a document by the same route: start from $\diamond$, draw a token from $p(\,\cdot \mid \diamond)$, append it, draw the next token from the distribution for the longer context, and stop when it draws $\diamond$. Exercise 9.6 shows that, under a mild condition, the probabilities of all documents sum to 1.

9.3. Bigram models

The simplest useful language model forgets everything but the latest token. To see what such a model should predict, look at what follows `a` in the corpus. The wrapped names contain eight pairs that start with `a`: five times `a` ends a name, and `d`, `n` and `v` each follow it once. A natural guess for the probability that `a` is followed by the boundary token is 5/8, and 1/8 for each of the three letters. This section shows that the guess is the maximum-likelihood estimate. Shannon (1948) used models of this kind, built from frequencies of letter pairs in English, to generate imitations of English text.

Definition 9.4 (Bigram model)**.** A *bigram model* is a language model whose next-token distribution depends only on the last token of the context: $p(b \mid x_1, \dots, x_t) = p(b \mid x_t)$. It is given by $|V|^2$ numbers $p(b \mid a)$, the *transition probabilities*, with

$$p(b \mid a) \geq 0, \qquad \sum_{b \in V} p(b \mid a) = 1 \quad \text{for every } a \in V.$$

The number $p(b \mid a)$ is the probability that token b comes next when the current token is a, and the conditions say that, for each current token, these probabilities form a distribution. Arranged as a $|V| \times |V|$ *transition matrix*, row a lists the probabilities of the possible next tokens after a, column b lists the probabilities of reaching b from each token, and each row sums to 1. A sequence in which each token is drawn from a distribution

that depends only on the token before is called a *Markov chain*. Because each row sums to 1, the transition matrix sends the all-ones vector to itself: 1 is an eigenvalue with eigenvector $(1, \dots, 1)^\top$.

To fit a bigram model to data we choose the transition probabilities that make the data most probable. Write n_{ab} for the number of times token b follows token a in the corpus, counting the pairs in every wrapped name, and $n_a = \sum_b n_{ab}$ for the number of pairs that start with a. By Equation 9.3, the probability of one name is a product with one factor $p(b \mid a)$ for each of its pairs. The names are treated as independent, so the probability of the corpus is the product over all names. Chapter 8 calls the probability that a model gives to the observed data, viewed as a function of the model's parameters with the data held fixed, the likelihood (Definition 8.1). There the observations were independent draws from one distribution; here each factor is a transition probability, but the idea is the same. Gathering equal factors, the likelihood of the bigram model is

$$\mathcal{L}(p) = \prod_{a,b \in V} p(b \mid a)^{n_{ab}},$$

in which each transition probability appears once for every time its pair occurs. Maximising the likelihood is the same as maximising its logarithm, the log-likelihood

$$\ln \mathcal{L}(p) = \sum_{a,b \in V} n_{ab} \ln p(b \mid a), \tag{9.4}$$

where p stands for the whole table of transition probabilities, terms with $n_{ab} = 0$ count as 0, and $\ln \mathcal{L}(p) = -\infty$ if $p(b \mid a) = 0$ for a pair with $n_{ab} > 0$.

Counting the pairs takes a few lines of code. Here the boundary token is written as a full stop, and each line of output lists what followed one token.

```
names = ["anna", "ava", "emma", "eva", "ada"]
counts = {}
```

```
for name in names:
    tokens = ["."] + list(name) + ["."]
    for a, b in zip(tokens, tokens[1:]):
        counts[a, b] = counts.get((a, b), 0) + 1
for a in sorted({a for a, _ in counts}):
    print(a, {b: n for (x, b), n in sorted(counts.items()) if x ==
    ↪  a})
```

```
. {'a': 3, 'e': 2}
a {'.': 5, 'd': 1, 'n': 1, 'v': 1}
d {'a': 1}
e {'m': 1, 'v': 1}
m {'a': 1, 'm': 1}
n {'a': 1, 'n': 1}
v {'a': 2}
```

The same counts form the matrix in Table 9.1. Row a lists what followed a, the entry in column b is n_{ab}, and the last column is the row total n_a.

Table 9.1.: Pair counts n_{ab} for the five names: row a is the earlier token, column b the later one.

$a \backslash b$	◇	a	d	e	m	n	v	n_a
◇	0	3	0	2	0	0	0	5
a	5	0	1	0	0	1	1	8
d	0	1	0	0	0	0	0	1
e	0	0	0	0	1	0	1	2
m	0	1	0	0	1	0	0	2
n	0	1	0	0	0	1	0	2
v	0	2	0	0	0	0	0	2

The maximiser of Equation 9.4 comes from Theorem 8.2: for observations treated as independent draws from one distribution on a finite set, the likelihood is largest at the relative frequencies of the observations and

nowhere else. A bigram model holds one such distribution for each current token, and the theorem applies to one row of the transition matrix at a time.

Theorem 9.2 (Maximum likelihood for a bigram model)**.** *Let n_{ab} be the pair counts of a corpus over a vocabulary V, and $n_a = \sum_b n_{ab}$. A bigram model p over V maximises the log-likelihood Equation 9.4 if and only if*

$$p(b \mid a) = \frac{n_{ab}}{n_a} \quad \text{for every } a \text{ with } n_a > 0 \text{ and every } b.$$

Rows with $n_a = 0$ can be any probability distributions.

The best transition probabilities are the relative frequencies of each row: the fraction of the pairs starting with a that continue with b.

Proof. The likelihood is a product over rows, $\mathcal{L}(p) = \prod_a \mathcal{L}_a(p)$ with $\mathcal{L}_a(p) = \prod_b p(b \mid a)^{n_{ab}}$, so $\ln \mathcal{L}(p) = \sum_a \ln \mathcal{L}_a(p)$, and $\mathcal{L}_a$ depends only on row a of p. The constraints of Definition 9.4 also act on each row separately. So p maximises $\ln \mathcal{L}$ exactly when each row maximises its own $\ln \mathcal{L}_a$. If $n_a = 0$ then $\mathcal{L}_a = 1$ whatever the row is. If $n_a > 0$, take as observations the n_a tokens that follow a in the corpus, of which n_{ab} equal b. Then $\mathcal{L}_a(p)$ is their likelihood under the distribution $p(\cdot \mid a)$, so by Theorem 8.2 it is largest exactly when $p(b \mid a) = n_{ab}/n_a$ for every b, and since ln is increasing, so is $\ln \mathcal{L}_a$. □

Fitting the model needs no optimisation, only counting. The rows sum to 1 because $\sum_b n_{ab} = n_a$.

Example 9.2. Dividing each row of Table 9.1 by its total gives the maximum-likelihood model. Its non-zero entries are

$$p(\texttt{a} \mid \diamond) = \tfrac{3}{5}, \quad p(\texttt{e} \mid \diamond) = \tfrac{2}{5}, \quad p(\diamond \mid \texttt{a}) = \tfrac{5}{8},$$
$$p(\texttt{d} \mid \texttt{a}) = p(\texttt{n} \mid \texttt{a}) = p(\texttt{v} \mid \texttt{a}) = \tfrac{1}{8}, \quad p(\texttt{a} \mid \texttt{d}) = p(\texttt{a} \mid \texttt{v}) = 1,$$
$$p(\texttt{m} \mid \texttt{e}) = p(\texttt{v} \mid \texttt{e}) = p(\texttt{a} \mid \texttt{m}) = p(\texttt{m} \mid \texttt{m}) = p(\texttt{a} \mid \texttt{n}) = p(\texttt{n} \mid \texttt{n}) = \tfrac{1}{2}.$$

Row `a` is the guess made at the start of this section. The name `eva` has probability $p(\texttt{e} \mid \diamond)\, p(\texttt{v} \mid \texttt{e})\, p(\texttt{a} \mid \texttt{v})\, p(\diamond \mid \texttt{a}) = \frac{2}{5} \cdot \frac{1}{2} \cdot 1 \cdot \frac{5}{8} = \frac{1}{8}$. The model also gives positive probability to names it has not seen: `ana` has probability $\frac{3}{5} \cdot \frac{1}{8} \cdot \frac{1}{2} \cdot \frac{5}{8} = \frac{3}{128}$. Its pairs `an` and `na` both occur in `anna`, and the model recombines them.

9.3.1. Zero counts and smoothing

The maximum-likelihood model gives probability 0 to every pair it has not seen. The name `dana` is plausible, but no name in the corpus starts with `d`, so $p(\texttt{d} \mid \diamond) = 0$ and the model gives `dana` probability 0. A model that rules out every unseen pair fits the training names well and new names badly.

The simplest repair is *add-one* (or *Laplace*) *smoothing*: pretend that every pair occurred once more than it did,

$$p(b \mid a) = \frac{n_{ab} + 1}{n_a + |V|}. \tag{9.5}$$

The numerator adds one to each count; the denominator adds $|V|$, one for each possible next token, so each row still sums to 1. Every probability is now positive, and Exercise 9.4 shows that Equation 9.5 is the maximum-likelihood model for the corpus with one extra copy of each of the $|V|^2$ pairs. With $|V| = 7$, the smoothed model gives $p(\texttt{d} \mid \diamond) = 1/12$ and

$$P(\texttt{dana}) = \frac{1}{12} \cdot \frac{2}{8} \cdot \frac{2}{15} \cdot \frac{2}{9} \cdot \frac{6}{15} = \frac{1}{4050}.$$

The price is a worse fit to the training data, measured in the next section. Add-one smoothing moves a lot of probability to unseen pairs when the counts are small, and practical systems use better methods; Chen and Goodman (1999) compare many of them.

9.4. Loss and perplexity

To compare models we need one number that says how well a model predicts a text. The probability of the whole text will not do, because it shrinks as the text gets longer. Under the model of Example 9.2, `eva` has probability $1/8$ and `anna` has $3/256$, partly because `anna` needs five predictions and `eva` four. The four predictions for `eva` have probabilities $\frac{2}{5}, \frac{1}{2}, 1, \frac{5}{8}$, and their *geometric mean*, the fourth root of their product, is $(1/8)^{1/4} \approx 0.595$. For `anna` the geometric mean of the five probabilities is $(3/256)^{1/5} \approx 0.411$. Per prediction, the model finds `eva` easier. The reciprocal of the geometric mean, $8^{1/4} \approx 1.68$ for `eva`, reads as a number of choices: the model is as uncertain, on average, as if it were choosing among 1.68 equally likely tokens at each step. Taking logarithms turns the geometric mean into an ordinary average, which is easier to compute and to differentiate.

Definition 9.5 (Loss and perplexity)**.** Suppose a model makes N predictions on a text and gives probability $p_i > 0$ to the token that actually comes next at prediction i. Its loss on the text is the average negative log-likelihood

$$L = -\frac{1}{N} \sum_{i=1}^{N} \ln p_i, \tag{9.6}$$

and its *perplexity* is

$$\mathrm{PPL} = e^{L} = \Big(\prod_{i=1}^{N} p_i \Big)^{-1/N}. \tag{9.7}$$

The loss is the average, over the predictions, of how surprised the model is by the correct token ($-\ln p_i$ is 0 when $p_i = 1$ and grows as p_i falls), and the perplexity is the reciprocal of the geometric mean of the p_i. The loss L is the average negative log-likelihood of Equation 8.1, the loss that every model in the book uses, and training minimises it. For a bigram model on a corpus with pair counts n_{ab}, the loss is $-\ln \mathcal{L}(p)/N$, with $N = \sum_{a,b} n_{ab}$

and $\ln \mathcal{L}$ from Equation 9.4, so minimising the loss and maximising the likelihood are the same problem.

The loss is also an average of cross-entropies. The cross-entropy of a distribution q relative to a distribution r on V is $H(r, q) = -\sum_{v \in V} r(v) \ln q(v)$ (Definition 8.3), the average of $-\ln q$ when outcomes are drawn from r. For a token a with $n_a > 0$, let $\hat{r}_a(b) = n_{ab}/n_a$ be the observed frequency of b after a. Grouping the predictions by the current token,

$$L = -\frac{1}{N} \sum_{a,b \in V} n_{ab} \ln p(b \mid a) = \sum_{a:\, n_a > 0} \frac{n_a}{N} H(\hat{r}_a,\ p(\cdot \mid a)).$$

In words, the loss averages, over the current tokens and weighted by how often each occurs, the cross-entropy of the model's prediction relative to the frequencies observed after that token.

The next result makes the reading of perplexity as a number of choices exact.

Theorem 9.3 (The uniform model)**.** *Let V be a vocabulary. A model that gives every token of V probability $1/|V|$ at every prediction has loss $\ln |V|$ and perplexity $|V|$ on every text.*

Proof. Every p_i is $1/|V|$, so $L = -\frac{1}{N} \cdot N \ln(1/|V|) = \ln |V|$ and $e^L = |V|$. □

For the 27-token vocabulary of names, the uniform model has loss $\ln 27 \approx 3.30$, the value that Example 8.5 found for a single prediction. This is where training starts. The book's model starts from small random weights, so its first predictions are close to uniform: Example 8.5 estimates its expected loss at the start as about 3.35, and the loss on the first training step of microgpt is about 3.37 (Karpathy 2026a). A model that has learned nothing has a loss near 3.30, and every improvement shows as a loss below that.

Example 9.3. On the 22 predictions of the training corpus, the maximum-likelihood model of Example 9.2 has log-likelihood

$$\ln \mathcal{L}(p) = 3\ln\tfrac{3}{5} + 2\ln\tfrac{2}{5} + 5\ln\tfrac{5}{8} + 3\ln\tfrac{1}{8} + 6\ln\tfrac{1}{2} \approx -16.11,$$

one term for each non-zero entry of Table 9.1, grouped by equal probabilities. So its loss is $16.11/22 \approx 0.732$ and its perplexity is $e^{0.732} \approx 2.08$. The uniform model over the 7 tokens has loss $\ln 7 \approx 1.95$ and perplexity 7. The add-one smoothed model has loss about 1.33 and perplexity about 3.79 on the training names: worse than the maximum-likelihood model there, as Theorem 9.2 guarantees. On the new name `dana`, with 5 predictions, the maximum-likelihood model has infinite loss, and the smoothed model has loss $\ln(4050)/5 \approx 1.66$ and perplexity $4050^{1/5} \approx 5.27$.

9.5. Embeddings and a neural bigram model

A table of transition probabilities treats every token as unrelated to every other: what the corpus says about `a` tells the table nothing about `e`. Neural language models instead give each token a vector, compute the probabilities from the vectors, and so let tokens with similar vectors receive similar predictions. The first step is to turn a token into a vector.

Number the tokens $1, 2, 3$ and attach to each a row of the matrix

$$E = \begin{pmatrix} 1 & 0 \\ 2 & -1 \\ 0 & 3 \end{pmatrix},$$

so that token 2 has the vector $(2, -1)$. Writing token 2 as the standard basis vector $\mathbf{e}_2 = (0, 1, 0)^\top$ of $\mathbb{R}^3$ turns the lookup into a matrix product: $E^\top \mathbf{e}_2$ is the combination of the columns of $E^\top$ with weights $0, 1, 0$, which is column 2 of $E^\top$, the vector $(2, -1)^\top$.

Definition 9.6 (One-hot vector and embedding)**.** Number the tokens of a vocabulary V as $1, \dots, |V|$. The one-hot vector of token a, as in Section 8.6, is the standard basis vector $\mathbf{e}_a$ of $\mathbb{R}^{|V|}$. An *embedding matrix* of width d is a $|V| \times d$ matrix E, and the *embedding* of token a is

$$\mathbf{u}_a = E^\top \mathbf{e}_a \in \mathbb{R}^d,$$

the ath row of E written as a column.

Row a of E is the vector of token a, and column j holds the jth component of every token's vector. Multiplying by a one-hot vector picks out one row, so programs store E and look up row a directly instead of forming a product.

The book's model has an embedding matrix with $|V| = 27$ rows and width $d = 16$. An embedding alone predicts nothing. The smallest neural language model maps the embedding of the current token to one score for each possible next token, and turns the scores into probabilities with the softmax function of Definition 8.5, $\operatorname{softmax}(\mathbf{z})_i = e^{z_i} / \sum_j e^{z_j}$, which turns any vector into positive numbers that sum to 1. Continue the example with a second 3×2 matrix W, with rows $(1, 0)$, $(0, 1)$ and $(1, 1)$. After token 2, the scores are $W\mathbf{u}_2 = (2, -1, 1)$, one for each token that could come next, and softmax turns them into the probabilities

$$\frac{(e^2, e^{-1}, e^1)}{e^2 + e^{-1} + e^1} \approx (0.705, 0.035, 0.259).$$

Definition 9.7 (Neural bigram model)**.** Let V be a vocabulary and E, W be $|V| \times d$ matrices. The *neural bigram model* with parameters E, W is the bigram model with transition probabilities

$$p(\cdot \mid a) = \operatorname{softmax}(\mathbf{z}_a), \qquad \mathbf{z}_a = W\mathbf{u}_a = WE^\top \mathbf{e}_a.$$

The vector $\mathbf{z}_a \in \mathbb{R}^{|V|}$ holds the logits (the scores that go into the softmax, as in Definition 8.5) for the token after a. The $|V| \times |V|$ matrix $Z = WE^\top$ is the *logit matrix*.

Entry (b, a) of Z is the logit of b as the next token after a: column a of Z is $\mathbf{z}_a$, the scores used after a, and row b holds the scores that token b receives after each possible current token. Row b of W is the vector that the embedding of the current token is compared with, by a dot product, to score b.

The book's model ends the same way: a 27×16 output matrix turns a 16-dimensional vector into 27 logits. In between, the full model replaces $\mathbf{u}_a$ by a vector that depends on the whole context, which is what Chapter 10 and Chapter 11 build.

What can such a model represent? Its probabilities come out of a softmax, so they are never zero, and the logit matrix is a product through a d-dimensional space, which limits its *rank*, the dimension of the space spanned by its columns.

Theorem 9.4 (What a neural bigram model can represent)**.** *Let V be a vocabulary and $d \geq 1$.*

(a) Every neural bigram model with $|V| \times d$ parameter matrices is a bigram model whose transition probabilities are all positive.

(b) Every logit matrix $Z = WE^\top$ with E and W of size $|V| \times d$ has rank at most d.

(c) If $d = |V|$, every bigram model over V whose transition probabilities are all positive is a neural bigram model: take $E = I$ and let W have entries $w_{ba} = \ln p(b \mid a)$.

Proof.

(a) Softmax outputs are positive and sum to 1, so each $p(\cdot \mid a)$ is a probability distribution with positive entries.

(b) Each column of $Z = WE^\top$ is W times a vector, so it is a combination of the d columns of W. The column space of Z lies in a space spanned by d vectors, so its dimension is at most d.

(c) With $E = I$, $\mathbf{u}_a = \mathbf{e}_a$ and $\mathbf{z}_a$ is column a of W, whose entries are $\ln p(b \mid a)$. Then

$$\operatorname{softmax}(\mathbf{z}_a)_b = \frac{e^{\ln p(b|a)}}{\sum_c e^{\ln p(c|a)}} = \frac{p(b \mid a)}{\sum_c p(c \mid a)} = p(b \mid a).$$

□

So with $d = |V|$ the neural bigram model can represent every bigram model except those with zero entries, and those it can approach by letting logits tend to $-\infty$. Training it by gradient descent on the loss of Definition 9.5 leads to the same place as counting.

Corollary 9.1. *Let V be a vocabulary, $d = |V|$, and let $L(E, W)$ be the loss of the neural bigram model with parameters E, W on a corpus with pair counts n_{ab} and row totals n_a.*

(a) If every n_{ab} is positive, the minimum of L is attained, and (E, W) attains it exactly when the model's transition probabilities are the relative frequencies n_{ab}/n_a.

(b) If $n_{ab} = 0$ for some pair with $n_a > 0$, then L has no minimum: its infimum is the loss of the relative frequencies, and no (E, W) attains it.

Proof. Write $f(b \mid a) = n_{ab}/n_a$ for the relative frequencies and L^* for their loss. By Theorem 9.2, every bigram model has loss at least L^*, and a model has loss L^* only if it agrees with f on every row with $n_a > 0$.

(a) Every n_a is positive, so every row is determined. The frequencies are positive, so by part (c) of Theorem 9.4 some (E, W) gives them, with loss L^*, and any (E, W) with loss L^* gives them.

(b) By part (a) of Theorem 9.4, no neural bigram model agrees with f, because f has a zero entry in a row with $n_a > 0$. So $L > L^*$ everywhere. For $0 < \delta < 1$, the model $p_\delta = (1-\delta)f + \delta/|V|$ (with f extended by uniform rows where $n_a = 0$) has positive entries, so it is a neural bigram model. Each term of its loss with $n_{ab} > 0$ has $f(b \mid a) > 0$, and $\ln p_\delta(b \mid a) \to \ln f(b \mid a)$ as $\delta \to 0$ because ln is continuous at positive numbers. So the loss of p_δ tends to L^*.

□

In case (b), the models p_δ approach the infimum as the probabilities of unseen pairs tend to 0. A softmax entry satisfies $e^{z_b}/\sum_c e^{z_c} \geq e^{z_b - \max_c z_c}/|V|$, so it can tend to 0 only if its logit falls without bound below the largest logit in its column, and then some entries of E or W grow without bound. Gradient descent behaves this way: on the five names, with $d = |V| = 7$, it keeps lowering the loss towards the infimum while the weights keep growing.

9.5.1. How a neural model shares information

The two models differ in what one training example changes. In a count table, an occurrence of the pair (a, b) changes row a and nothing else, so the predictions after a rare token rest on that token's few occurrences alone. In the neural model the matrix W is used in every row, so a gradient step taken for one pair changes predictions after other tokens too. The next lemma says by how much.

Lemma 9.1. *Consider a neural bigram model with parameters E, W, and let $\ell = -\ln p(b \mid a)$ be the loss of one occurrence of the pair (a, b). Write $\mathbf{p} = p(\cdot \mid a)$. Then*

$$\frac{\partial \ell}{\partial W} = (\mathbf{p} - \mathbf{e}_b)\, \mathbf{u}_a^\top,$$

where $\partial \ell / \partial W$ is the matrix of partial derivatives $\partial \ell / \partial w_{ij}$. After a gradient step $W \leftarrow W - \eta\, \partial \ell / \partial W$ with learning rate $\eta > 0$ and E held fixed, the logits after any token c change by

$$-\eta\, (\mathbf{u}_a \cdot \mathbf{u}_c)\, (\mathbf{p} - \mathbf{e}_b).$$

The gradient is an outer product: row i is the embedding $\mathbf{u}_a$ scaled by $p_i - [i = b]$, where $[i = b]$ is 1 if $i = b$ and 0 otherwise. The step lowers the logit of the observed token b relative to the others after a, and moves the logits after c in the same pattern, scaled by the dot product of the two embeddings.

Proof. By Theorem 8.5, $\partial \ell / \partial z_i = p_i - [i = b]$, the ith entry of $\mathbf{p} - \mathbf{e}_b$, where $\mathbf{z} = W \mathbf{u}_a$. Since $z_i = \sum_j w_{ij} u_{a,j}$, the weight w_{ij} affects only z_i, with $\partial z_i / \partial w_{ij} = u_{a,j}$. By the chain rule, $\partial \ell / \partial w_{ij} = (p_i - [i = b])\, u_{a,j}$, which is entry (i, j) of $(\mathbf{p} - \mathbf{e}_b)\mathbf{u}_a^\top$. The new logits after c are

$$\Big(W - \eta (\mathbf{p} - \mathbf{e}_b)\mathbf{u}_a^\top\Big)\mathbf{u}_c = W\mathbf{u}_c - \eta\, (\mathbf{u}_a^\top \mathbf{u}_c)(\mathbf{p} - \mathbf{e}_b).$$

□

So a step taken for an occurrence of a also moves the predictions after every token c whose embedding is not perpendicular to $\mathbf{u}_a$. When $d < |V|$ this cannot be avoided. Vectors $\mathbf{u}_1, \ldots, \mathbf{u}_m$ are *linearly independent* if the only combination $\sum_i \lambda_i \mathbf{u}_i$ equal to $\mathbf{0}$ has every $\lambda_i = 0$, and $\mathbb{R}^d$ contains at most d linearly independent vectors. Non-zero vectors that are pairwise perpendicular are linearly independent: if $\sum_i \lambda_i \mathbf{u}_i = \mathbf{0}$, taking the dot product with $\mathbf{u}_j$ gives $\lambda_j \|\mathbf{u}_j\|^2 = 0$. So if the $|V|$ embeddings are non-zero, some two of them have a non-zero dot product.

This is the precise sense in which a neural model with small embeddings shares information between tokens. If training gives similar embeddings to tokens that are followed by similar tokens, then evidence about one helps the predictions after the other. Nothing proved here says that training does

this, or that sharing lowers the loss on names outside the training data; that depends on the data and on the training run. Bengio et al. (2003) proposed learned embeddings for word-level language models with this argument, and reported on two text corpora that their models improved on the best n-gram models of the time. The neural model is not smaller than the table for bigrams: with $|V| = 27$ and $d = 16$ it has $2 \cdot 27 \cdot 16 = 864$ parameters, against $27 \cdot 26 = 702$ free numbers in a table (each row of 27 probabilities has one fewer free number, because it must sum to 1). The saving in parameters appears with longer contexts.

9.6. Longer contexts

A bigram model cannot know that `emm` is likely to continue with `a`, because it sees only the last `m`, and after a single `m` the corpus has `a` and `m` equally often. The obvious extension conditions on more tokens.

Definition 9.8 (n-gram model)**.** Let $n \geq 2$. An *n-gram model* is a language model whose next-token distribution depends only on the last $n-1$ tokens of the context, with the context padded by $\diamond$ at the start when it is shorter.

A bigram model is the case $n = 2$. An n-gram model is a table with one row for each possible sequence of $n-1$ tokens and one entry in each row for each next token, so it has $|V|^{n-1}$ rows and $|V|^n$ entries. Its maximum-likelihood estimate is again given by relative frequencies, by the proof of Theorem 9.2 with rows indexed by contexts. The table grows exponentially with the context length: with $|V| = 27$, a trigram ($n = 3$) model has $27^2 = 729$ rows, a 4-gram model $27^3 = 19{,}683$ rows, and a model that conditions on 16 tokens, the context length of the book's model, has $27^{16} \approx 7.98 \times 10^{22}$ rows.

The data cannot fill such a table. Each prediction in the training data adds a count to one row, so a corpus with N predictions gives counts to at most N rows. The names data of microgpt has 32,033 names (Karpathy 2026a),

and the model makes at most 16 predictions per name, so there are at most $32{,}033 \times 16 = 512{,}528$ predictions. Even for the 4-gram model, a corpus of 1000 predictions leaves more than 94% of the rows empty (Exercise 9.9), and for a 16-token context the fraction of rows that receive any count at all is below 10^{-17}. A row without counts is a context about which the table knows nothing, and smoothing can only spread probability evenly over its entries.

Neural models avoid this growth by computing the distribution from the context instead of looking it up. Bengio et al. (2003) concatenated the embeddings of the last $n - 1$ tokens and passed them through a small network, so the number of parameters grows linearly with n. That model still has a fixed window, and each position in the window has its own weights. The book's model forms, at each position, a weighted average of vectors computed from all the earlier tokens, with weights that the tokens themselves determine, and it uses the same parameters for every context length up to 16. That operation is attention, the subject of Chapter 10.

9.7. Notes and further reading

Section 3 of Shannon (1948) builds a series of approximations to English from letter and word frequencies, including letter-pair (bigram) models, and treats them as Markov processes. Jurafsky and Martin (2026, ch. 3) cover n-gram models, perplexity and smoothing at greater length, and Chen and Goodman (1999) compare smoothing methods on large corpora. The neural language model with learned embeddings is due to Bengio et al. (2003). Radford et al. (2019) describe GPT-2, a large model that factorises the probability of text with the chain rule as in Definition 9.3 and is evaluated by perplexity. Karpathy (2026a) describes the small model that this book rebuilds, including its tokeniser and training data.

9.8. Exercises

Exercise 9.1.

(a) Write the name `ada` as a sequence of microgpt token numbers.

(b) Show that a corpus of m names with M letters in total gives $M + m$ predictions, and check this against the five names of Section 9.1.

Exercise 9.2. Choose one of the five corpus names uniformly at random, and let X_1 and X_2 be its first and second letters. Compute $P(X_2 = \texttt{v})$ and $P(X_1 = \texttt{a} \mid X_2 = \texttt{v})$, and check that $P(X_2 = \texttt{v})\, P(X_1 = \texttt{a} \mid X_2 = \texttt{v})$ equals $P(X_1 = \texttt{a}, X_2 = \texttt{v})$ from Example 9.1.

Exercise 9.3. Using the maximum-likelihood model of Example 9.2, compute the probabilities of `anna` and `emma`. Which name in the corpus has the largest probability under the model?

Exercise 9.4.

(a) Show that each row of the add-one model Equation 9.5 sums to 1.

(b) Show that Equation 9.5 is the maximum-likelihood bigram model for the counts $n_{ab} + 1$.

(c) Compute $p(\diamond \mid \texttt{a})$ in the add-one model of Section 9.3 and compare it with the maximum-likelihood value 5/8.

Exercise 9.5.

(a) Show that the perplexity of Definition 9.5 is at least 1, with equality exactly when every $p_i = 1$.

(b) Model A has loss 0.1 lower than model B on the same text. Find the ratio of their perplexities.

Exercise 9.6. Consider a bigram model with $p(\diamond \mid a) \geq \varepsilon$ for every $a \in V$, where $0 < \varepsilon \leq 1$, and generate names as described after Definition 9.3. For $k \geq 0$ let r_k be the probability that the first k tokens generated are all letters (so $r_0 = 1$), and let q_k be the total probability of all names with exactly k letters.

(a) Show that $r_{k+1} = r_k - q_k$ and $r_{k+1} \leq (1 - \varepsilon) r_k$.

(b) Deduce that the probabilities of all names sum to 1.

Exercise 9.7. Let $|V| = 2$ and $d = 1$. Show that every bigram model with positive transition probabilities is a neural bigram model. (Hint: with two tokens, the softmax depends only on the difference of the two logits.)

Exercise 9.8. In the setting of Lemma 9.1, show that $\partial \ell / \partial E = \mathbf{e}_a (W^\top (\mathbf{p} - \mathbf{e}_b))^\top$, so that a gradient step on E changes only the embedding of a.

Exercise 9.9. A corpus gives N predictions. Show that at most N rows of an n-gram table have a non-zero count. For $|V| = 27$, $n = 4$ and $N = 1000$, find a lower bound for the fraction of rows with no counts.

10. Attention

A bigram model predicts from the latest token alone, and a table indexed by longer contexts is too large to fill (Section 9.6). The book's model takes a different route. At each position it forms a weighted average of vectors computed from the earlier tokens, and the weights are themselves computed from the tokens: each position decides how much to draw on each earlier one. That operation is *attention*. Its parameters are a few matrices whose sizes do not depend on how many tokens came before, so the same parameters serve every context length.

This chapter defines attention and the variants that the book's model uses: scaled dot products, the causal mask, several heads and position embeddings. Each comes with a mathematical reason. Dividing the scores by the square root of the length of the vectors keeps the softmax away from the region where its derivatives vanish. The causal mask stops each position from drawing on later ones, so each prediction depends only on the past, and work done for earlier positions can be stored and reused. Position embeddings are needed because attention without them ignores the order of the tokens. The chapter ends with the derivative of attention, which shows how gradients pass through it, and with its cost.

10.1. Attention as a lookup

A *lookup table* (a dictionary, in programming) stores pairs of a *key* and a *value*, and answers a *query* by returning the value whose key equals the query. A model that wants to use earlier tokens could store, for each earlier

position, a key describing it and a value carrying its information, and give the current position a query describing what it needs, to be matched against the keys. An exact lookup cannot be trained by gradient descent: moving the query slightly either changes nothing or jumps to a different value, so its derivative with respect to the query is zero wherever it exists. Attention replaces the exact match by a graded one.

Take a query $\mathbf{q} = (1,1,1,1)$ and three keys $\mathbf{k}_1 = (1,1,0,0)$, $\mathbf{k}_2 = (1,-1,0,0)$, $\mathbf{k}_3 = (0,0,1,-1)$, stored with values $\mathbf{v}_1 = (1,0)$, $\mathbf{v}_2 = (0,1)$, $\mathbf{v}_3 = (0,0)$. Score each key by its dot product with the query: $2,0,0$. Divide the scores by 2 (the square root of the width 4; Section 10.2 explains why), giving $1,0,0$. The softmax of Definition 8.5, $\mathrm{softmax}(\mathbf{s})_j = e^{s_j}/\sum_r e^{s_r}$, turns the scores into weights that are positive and sum to 1,

$$\frac{(e,1,1)}{e+2} \approx (0.576, 0.212, 0.212),$$

and the answer is the weighted average of the values, $(e/(e{+}2),\ 1/(e{+}2)) \approx (0.576, 0.212)$. The first key matches the query best, and its value dominates without taking over completely. Every step is differentiable, so a small change in the query, a key or a value changes the answer a little.

Definition 10.1 (Attention)**.** Let $\mathbf{q} \in \mathbb{R}^{d_h}$ be a *query*, $\mathbf{k}_1, \dots, \mathbf{k}_T \in \mathbb{R}^{d_h}$ *keys* and $\mathbf{v}_1, \dots, \mathbf{v}_T \in \mathbb{R}^{d_v}$ *values*. The *scores* and *attention weights* are

$$s_j = \frac{\mathbf{q}\cdot\mathbf{k}_j}{\sqrt{d_h}}, \qquad \mathbf{a} = \mathrm{softmax}(\mathbf{s}), \quad \text{so} \quad a_j = \frac{e^{s_j}}{\sum_{r=1}^{T} e^{s_r}},$$

and the *output* of attention is

$$\mathrm{Attn}(\mathbf{q}; \mathbf{k}_{1:T}, \mathbf{v}_{1:T}) = \sum_{j=1}^{T} a_j \mathbf{v}_j. \tag{10.1}$$

In matrix form, let K be the $T \times d_h$ matrix with rows $\mathbf{k}_j^\top$ and V the $T \times d_v$ matrix with rows $\mathbf{v}_j^\top$. For queries $\mathbf{q}_1, \dots, \mathbf{q}_m$, the rows of an $m \times d_h$ matrix

Q,

$$\mathrm{Attn}(Q, K, V) = AV, \qquad A = \mathrm{softmax}\Big(\frac{QK^\top}{\sqrt{d_h}}\Big), \tag{10.2}$$

where softmax acts on each row separately.

In Equation 10.2, row t of Q is query t, and row j of K and row j of V are the key and the value stored together as item j. Entry (t, j) of the $m \times T$ matrix $QK^\top/\sqrt{d_h}$ is the score of key j for query t. After the softmax, row t of the $m \times T$ matrix A holds the weights that query t gives to the T keys, so each row sums to 1, and row t of AV is the output for query t. (In this chapter V is the matrix of values. The vocabulary, which Definition 9.1 also calls V, does not appear by name in this chapter.)

The weights are positive and sum to 1, so the output is a *convex combination* of the values, a weighted sum with non-negative weights that add up to 1. Each component of the output therefore lies between the smallest and largest values of that component. In a model the query, keys and values come from three different linear maps, so "matching" can mean whatever training makes it mean.

Attention was introduced for translation (Bahdanau et al. 2014), where the query comes from the sentence being written and the keys and values from the sentence being translated. The scaled dot-product form of Definition 10.1 is from Vaswani et al. (2017).

10.2. Why the scores are scaled

The dot product of two vectors of width d_h is a sum of d_h terms, so it tends to grow with the width. For vectors with random entries the growth can be measured exactly. Let each component of $\mathbf{q}$ and $\mathbf{k}$ be 1 or -1, each with probability $\frac{1}{2}$, independently. With $d_h = 2$ the dot product $q_1k_1 + q_2k_2$ is a sum of two terms that are each ± 1 with probability $\frac{1}{2}$, so it equals 2, 0 or -2 with probabilities $\frac{1}{4}$, $\frac{1}{2}$, $\frac{1}{4}$. Its mean is 0 and its variance is

$\frac{1}{4}\cdot 4+\frac{1}{2}\cdot 0+\frac{1}{4}\cdot 4=2$. The variance equals the width, and the next theorem shows that this is no coincidence. So the standard deviation of an unscaled score, its typical size, grows like $\sqrt{d_h}$, and dividing by $\sqrt{d_h}$ undoes the growth.

The proof uses the probability of Section 6.1: random variables on a finite probability space with their expectations and variances (Definition 6.1 and Definition 6.3), and independence (Definition 6.2). It also uses four facts proved there. Expectation is linear by Theorem 6.1, and functions of independent random vectors are independent by Lemma 6.1. The expectation of a product of independent random variables is the product of the expectations by Theorem 6.2, and $\operatorname{Var}[X]=\mathbb{E}[X^2]-\mathbb{E}[X]^2$ and $\operatorname{Var}[cX]=c^2\operatorname{Var}[X]$ for every real c by Lemma 6.2.

Theorem 10.1 (Size of a random dot product)**.** *Let* $\mathbf{q}$ *and* $\mathbf{k}$ *be independent random vectors in* $\mathbb{R}^{d_h}$ *on a finite probability space. Suppose that every component* q_i *and* k_i *has mean* 0 *and variance* 1*, and that for* $i\neq j$ *the components* q_i *and* q_j *are independent, and so are* k_i *and* k_j*. Then*

$$\mathbb{E}[\mathbf{q}\cdot\mathbf{k}]=0, \qquad \operatorname{Var}[\mathbf{q}\cdot\mathbf{k}]=d_h,$$

and the score $s=\mathbf{q}\cdot\mathbf{k}/\sqrt{d_h}$ *has mean* 0 *and variance* 1*.*

Proof. Each component has mean 0 and variance 1, so by Lemma 6.2 $\mathbb{E}[q_i^2]=1$, and likewise $\mathbb{E}[k_i^2]=1$. For $i\neq j$, $\mathbb{E}[q_iq_j]=\mathbb{E}[q_i]\,\mathbb{E}[q_j]=0$ by Theorem 6.2, and likewise $\mathbb{E}[k_ik_j]=0$.

Any function of $\mathbf{q}$ is independent of any function of $\mathbf{k}$ by Lemma 6.1. So, by linearity of expectation (Theorem 6.1) and Theorem 6.2,

$$\mathbb{E}[\mathbf{q}\cdot\mathbf{k}]=\sum_{i=1}^{d_h}\mathbb{E}[q_ik_i]=\sum_{i=1}^{d_h}\mathbb{E}[q_i]\,\mathbb{E}[k_i]=0.$$

Expanding the square of the dot product and grouping the factors from each vector,

$$\mathbb{E}[(\mathbf{q}\cdot\mathbf{k})^2] = \sum_{i=1}^{d_h}\sum_{j=1}^{d_h}\mathbb{E}\big[(q_iq_j)(k_ik_j)\big] = \sum_{i=1}^{d_h}\sum_{j=1}^{d_h}\mathbb{E}[q_iq_j]\,\mathbb{E}[k_ik_j].$$

The terms with $i \neq j$ are 0, and the d_h terms with $i = j$ are $1 \cdot 1$. So $\mathrm{Var}[\mathbf{q}\cdot\mathbf{k}] = \mathbb{E}[(\mathbf{q}\cdot\mathbf{k})^2] - 0^2 = d_h$. Finally, by Theorem 6.1 and Lemma 6.2 with $c = 1/\sqrt{d_h}$, the score has mean 0 and variance $d_h/d_h = 1$. □

The hypotheses hold in particular when all $2d_h$ components are drawn independently of one another, as when the entries of a model are initialised at random. The book proves the facts about independence and expectation only on finite probability spaces. As Section 7.5 notes, they hold for random variables with densities too, in the sense of Definition 7.4, including the normal distributions used to initialise the book's model, and with them the theorem. For any distributions whose variances exist, Billingsley (1995, Theorem 20.2 and eq. (21.18)) proves them using measure theory. The same calculation appears in a footnote of Vaswani et al. (2017).

Why does the size of the scores matter? Compare the softmax of the scores $(1, 0, -1)$ with the softmax of $(8, 0, -8)$. The first gives weights $(0.665, 0.245, 0.090)$; the second gives $(0.99966, 0.00034, 0.0000001)$, essentially all the weight on one key. Gradients reach the scores only through the derivatives of the softmax, and by Theorem 8.4 these are

$$\frac{\partial a_i}{\partial s_j} = a_i([i = j] - a_j), \tag{10.3}$$

where $[i = j]$ is 1 if $i = j$ and 0 otherwise: entry (i, j) of the Jacobian is the rate at which weight i changes per unit increase of score j. For the scores $(1, 0, -1)$ the largest entry in absolute value is about 0.22; for $(8, 0, -8)$ it is about 0.0003. When one weight is close to 1 we say that the softmax *saturates*, and then every derivative is small.

Lemma 10.1. *Let* $\mathbf{s} \in \mathbb{R}^T$ *and* $\mathbf{a} = \mathrm{softmax}(\mathbf{s})$*, and let* $0 \le \varepsilon \le 1$*. If some component satisfies* $a_m \ge 1 - \varepsilon$*, then every entry of the Jacobian of softmax at* $\mathbf{s}$ *has absolute value at most* ε*.*

Proof. The weights are positive and sum to 1, so every a_i with $i \neq m$ is at most $1 - a_m \le \varepsilon$. By Equation 10.3, the diagonal entry for i is $a_i(1 - a_i)$, which is at most $1 - a_m \le \varepsilon$ when $i = m$ and at most $a_i \le \varepsilon$ otherwise. An off-diagonal entry is $-a_i a_j$ with $i \neq j$, so at least one of i, j differs from m, and $|a_i a_j| \le \varepsilon$. □

The effect shows in Table 10.1, for random queries and keys with independent standard normal components, $T = 8$ keys, and 20,000 random trials for each row, with and without the scale.

Table 10.1.: Saturation of the softmax for random scores. The spread is the standard deviation of the scores; the Jacobian entry is the largest absolute entry of the softmax Jacobian in a trial; the last column is the fraction of trials in which one weight exceeds 0.99.

d_h	scores	spread of scores	mean largest weight	median largest Jacobian entry	largest weight above 0.99
4	unscaled	2.01	0.53	0.23	0.7%
4	scaled	1.00	0.34	0.21	0%
16	unscaled	3.98	0.75	0.17	9.4%
16	scaled	1.00	0.36	0.22	0%
64	unscaled	7.99	0.88	0.03	37.6%
64	scaled	1.00	0.36	0.22	0%

With the scale, the weights and the derivatives look the same at every width. Without it, at $d_h = 64$ (the head width of Vaswani et al. (2017))

more than a third of the trials have one weight above 0.99, and the typical largest derivative is less than a sixth of its scaled value, so gradient steps barely change the queries and keys. The book's model has $d_h = 4$, where the unscaled spread is only 2 and the effect is mild; the scale matters for wider heads, and the model keeps it to follow the standard design.

Why divide by $\sqrt{d_h}$ and not by some other power of d_h, such as d_h itself? Under the hypotheses of Theorem 10.1, the score $\mathbf{q} \cdot \mathbf{k}/d_h^{\alpha}$ has variance $d_h/d_h^{2\alpha} = d_h^{1-2\alpha}$, which is the same for every width only when $\alpha = \frac{1}{2}$. A smaller power lets the scores grow with the width, and the softmax saturates as in the table. A larger power makes them shrink. Dividing by d_h gives scores of variance $1/d_h$, so for wide heads every score is close to 0 and every weight close to $1/T$, whatever the query and the keys. At $d_h = 64$ with $T = 8$ keys these scores have standard deviation $1/8$, and in 20,000 random trials like those of the table the largest weight averages about 0.15 and the smallest about 0.10, on either side of the uniform weight $1/8 = 0.125$. Attention would then return nearly the plain average of the values, whatever the query. Among the powers of d_h, only $\sqrt{d_h}$ keeps the scores at variance 1 for every width, neither saturating the softmax nor flattening it.

10.3. Self-attention and the causal mask

In a language model the queries, keys and values all come from the same sequence. The sequence has T positions, $t = 1, \dots, T$, and position t holds the token x_t; the computation at position t ends in the prediction of the token at position $t+1$. Each position holds a vector $\mathbf{x}_t \in \mathbb{R}^d$. In the book's model this vector is built from the embedding of the token, the learned vector $\mathbf{u}_{x_t}$ that is row x_t of the embedding matrix E of Definition 9.6, as Section 10.5 describes. Three $d_h \times d$ matrices W_Q, W_K, W_V turn it into a query, a key and a value:

$$\mathbf{q}_t = W_Q \mathbf{x}_t, \qquad \mathbf{k}_t = W_K \mathbf{x}_t, \qquad \mathbf{v}_t = W_V \mathbf{x}_t.$$

Row i of W_Q gives component i of a query as a linear combination of the d components of the vector at that position, and likewise for W_K and W_V; the same three matrices serve every position. If X is the $T \times d$ matrix whose row t is $\mathbf{x}_t^\top$ (one row per position, one column per component), then the queries are the rows of $Q = XW_Q^\top$, and similarly $K = XW_K^\top$ and $V = XW_V^\top$.

For the smallest example take $d = d_h = 1$ and all three matrices equal to 1, so $q_t = k_t = v_t = x_t$, and take two positions with $x_1 = 1$ and $x_2 = 2$. Position 1 has query 1 and keys 1 and 2, so its scores are 1 and 2 (the scale $\sqrt{d_h}$ is 1). Its weights are $(e, e^2)/(e + e^2) \approx (0.27, 0.73)$, and its output is

$$y_1 = \frac{e \cdot 1 + e^2 \cdot 2}{e + e^2} \approx 1.731.$$

Definition 10.2 (Self-attention)**.** Let $\mathbf{x}_1, \dots, \mathbf{x}_T \in \mathbb{R}^d$ and let W_Q, W_K, W_V be $d_h \times d$ matrices. The *self-attention* of $\mathbf{x}_1, \dots, \mathbf{x}_T$ is the list of outputs $\mathbf{y}_t = \mathrm{Attn}(\mathbf{q}_t; \mathbf{k}_{1:T}, \mathbf{v}_{1:T})$ for $t = 1, \dots, T$, with $\mathbf{q}_t = W_Q\mathbf{x}_t$, $\mathbf{k}_t = W_K\mathbf{x}_t$ and $\mathbf{v}_t = W_V\mathbf{x}_t$. They are the rows of $\mathrm{Attn}(XW_Q^\top, XW_K^\top, XW_V^\top)$.

In self-attention every position sees every other, including later ones, and that is a problem for a language model. In the example, position 1 puts weight 0.73 on position 2. A language model predicts the token at position 2 from the vector at position 1. If that vector could see position 2, the model could read the answer instead of predicting it, and the product of its predictions would not be the factorisation of Theorem 9.1, in which the probability of each token is conditioned only on the tokens before it. The *causal mask* removes the later positions.

Definition 10.3 (Causal mask)**.** Let $\mathbf{q}_t$, $\mathbf{k}_t$, $\mathbf{v}_t$ for $t = 1, \dots, T$ be as in Definition 10.2, with scores $s_{tj} = \mathbf{q}_t \cdot \mathbf{k}_j / \sqrt{d_h}$. *Causal* (or *masked*)

self-attention gives position t weights only on positions $j \le t$:

$$a_{tj} = \begin{cases} \dfrac{e^{s_{tj}}}{\sum_{r=1}^{t} e^{s_{tr}}} & j \le t, \\ 0 & j > t, \end{cases} \qquad \mathbf{y}_t = \sum_{j=1}^{T} a_{tj} \mathbf{v}_j.$$

The *causal mask* is the $T \times T$ matrix M with entries $m_{tj} = 0$ for $j \le t$ and $m_{tj} = -\infty$ for $j > t$. With the convention $e^{-\infty} = 0$, the weight matrix $A = (a_{tj})$ is the softmax, row by row, of $S + M$, where $S = QK^\top / \sqrt{d_h}$ is the matrix of scores s_{tj}.

The weight a_{tj} is how much position t draws on position j: the softmax of the scores of positions 1 to t, and 0 for later positions. The weight matrix A is lower triangular (row t has non-zero entries only in columns 1 to t), and each row still sums to 1. Adding M hides, or *masks*, the scores of the later positions, which is where the name comes from, and programs compute A this way. In the example above, position 1 now attends only to itself, and $y_1 = 1$.

The mask has a consequence that the rest of the book relies on: the output at position t can be computed from positions 1 to t alone, so the work done for earlier positions never needs redoing when a new token arrives.

Theorem 10.2 (Causal attention needs only the past)**.** *Let $\mathbf{y}_1, \dots, \mathbf{y}_T$ be the outputs of causal self-attention of $\mathbf{x}_1, \dots, \mathbf{x}_T$ with matrices W_Q, W_K, W_V. For each t:*

(a) $\mathbf{y}_t = \mathrm{Attn}(\mathbf{q}_t; \mathbf{k}_{1:t}, \mathbf{v}_{1:t})$*, the unmasked attention of $\mathbf{q}_t$ over the keys and values of positions $1, \dots, t$ only;*

(b) $\mathbf{y}_t$ *depends only on $\mathbf{x}_1, \dots, \mathbf{x}_t$: changing the vectors at positions after t, or appending more positions, leaves $\mathbf{y}_t$ unchanged.*

Proof.

(a) In Definition 10.3 the terms with $j > t$ have weight 0, so $\mathbf{y}_t = \sum_{j=1}^{t} a_{tj}\mathbf{v}_j$, and for $j \le t$ the weight a_{tj} is the softmax of the t scores $s_{t1}, \dots, s_{tt}$. That is Definition 10.1 with the t keys $\mathbf{k}_1, \dots, \mathbf{k}_t$ and values $\mathbf{v}_1, \dots, \mathbf{v}_t$.

(b) By (a), $\mathbf{y}_t$ is computed from $\mathbf{q}_t$ and from $\mathbf{k}_j$, $\mathbf{v}_j$ for $j \le t$, and each of these is a fixed matrix times one of $\mathbf{x}_1, \dots, \mathbf{x}_t$.

□

Part (a) is what a *key–value cache* relies on. To generate text, a model computes position after position. At position t it computes $\mathbf{k}_t$ and $\mathbf{v}_t$, appends them to a list of the keys and values of the earlier positions (the cache), and attends from $\mathbf{q}_t$ over the list. By part (b), the keys and values already in the list never need recomputing, because adding a position does not change anything at earlier positions. The microgpt program (Karpathy 2026a) works exactly this way, during training as well as generation: it processes one position at a time and keeps the keys and values in lists. By Theorem 10.2 this computes the same outputs as applying the mask to all positions at once. The code below does both on random inputs and prints the largest difference.

```
demo = np.random.default_rng(0)
T, d = 5, 4
X = demo.normal(size=(T, d))
WQ, WK, WV = (demo.normal(size=(d, d)) for _ in range(3))
Q, K, V = X @ WQ.T, X @ WK.T, X @ WV.T

# All positions at once, with scores above the diagonal set to
↪  -inf.
S = Q @ K.T / np.sqrt(d)
S[np.triu_indices(T, k=1)] = -np.inf
A = np.exp(S - S.max(axis=1, keepdims=True))
Y_masked = (A / A.sum(axis=1, keepdims=True)) @ V

# One position at a time, keeping the keys and values seen so far.
```

```
keys, values, Y_cached = [], [], []
for t in range(T):
    keys.append(K[t])
    values.append(V[t])
    s = np.array(keys) @ Q[t] / np.sqrt(d)
    a = np.exp(s - s.max())
    Y_cached.append((a / a.sum()) @ np.array(values))

print(np.abs(Y_masked - np.array(Y_cached)).max())
```

```
2.220446049250313e-16
```

The difference is at the level of rounding error in floating-point arithmetic.

The same property makes a causal model a language model in the sense of Definition 9.3, a distribution for the next token given each context. Every other operation in the book's model acts on each position separately, so the vector at position t, and with it the prediction of the token at position $t+1$, depends only on the tokens at positions $1, \dots, t$.

10.4. Multi-head attention

A single set of attention weights gives each position one way to combine the others, and one set of weights cannot favour two different positions for two different reasons. Take $d = 4$, the query $\mathbf{q} = (1, 0, 0, 1)$ and two keys $\mathbf{k}_1 = (1, 0, 0, 0)$ and $\mathbf{k}_2 = (0, 0, 0, 1)$. Both dot products are 1, so one attention of width 4 gives the weights $(0.5, 0.5)$. Now split every vector into two halves of width 2 and attend separately in each half. In the first halves the query $(1, 0)$ matches $\mathbf{k}_1$'s $(1, 0)$ and not $\mathbf{k}_2$'s $(0, 0)$; the scores are $1/\sqrt{2}$ and 0, and the weights are about $(0.67, 0.33)$. In the second halves the roles swap and the weights are about $(0.33, 0.67)$. Each half is called a *head*, here of width $d_h = 2$. The two heads draw mainly on different positions, and their outputs are kept side by side.

Definition 10.4 (Multi-head attention)**.** Let $d = hd_h$ with integers $h, d_h \geq 1$, let $\mathbf{x}_1, \ldots, \mathbf{x}_T \in \mathbb{R}^d$, and let W_Q, W_K, W_V, W_O be $d \times d$ matrices. For each position t compute $\mathbf{q}_t = W_Q\mathbf{x}_t$, $\mathbf{k}_t = W_K\mathbf{x}_t$ and $\mathbf{v}_t = W_V\mathbf{x}_t$, and split each into h blocks of d_h consecutive components, $\mathbf{q}_t = (\mathbf{q}_t^{(1)}, \ldots, \mathbf{q}_t^{(h)})$ and likewise for keys and values. *Head* i applies causal self-attention (Definition 10.3) of width d_h to the ith blocks:

$$\mathbf{o}_t^{(i)} = \sum_{j \leq t} a_{tj}^{(i)} \mathbf{v}_j^{(i)}, \qquad a_{tj}^{(i)} = \frac{e^{\mathbf{q}_t^{(i)} \cdot \mathbf{k}_j^{(i)} / \sqrt{d_h}}}{\sum_{r \leq t} e^{\mathbf{q}_t^{(i)} \cdot \mathbf{k}_r^{(i)} / \sqrt{d_h}}}.$$

The heads' outputs are stacked into $\mathbf{o}_t = (\mathbf{o}_t^{(1)}, \ldots, \mathbf{o}_t^{(h)}) \in \mathbb{R}^d$, and the output of *multi-head attention* at position t is $\mathbf{y}_t = W_O\mathbf{o}_t$. Without the mask (sums over all j and r from 1 to T) this is *unmasked* multi-head attention.

Block i of $\mathbf{q}_t = W_Q\mathbf{x}_t$ is $W_Q^{(i)}\mathbf{x}_t$, where $W_Q^{(i)}$ consists of rows $(i-1)d_h + 1$ to id_h of W_Q. So each head is the single-head attention of Section 10.3 with its own $d_h \times d$ matrices, and the heads differ only in which rows of W_Q, W_K and W_V they use. Each head has its own weights $a_{tj}^{(i)}$. In the product $W_O\mathbf{o}_t$, columns $(i-1)d_h + 1$ to id_h of W_O multiply the output of head i, so row k of the $d \times d$ matrix W_O gives component k of $\mathbf{y}_t$ as a linear combination of the outputs of all the heads: the *output matrix* W_O mixes the heads' results (Exercise 10.8). The scale is $1/\sqrt{d_h}$ because d_h, the width of a head, is the length of the dot products in each head (Theorem 10.1).

The book's model has $d = 16$ and $h = 4$ heads of width $d_h = 4$, following microgpt (Karpathy 2026a). Its four matrices have $4 \cdot 16^2 = 1024$ parameters, the same as one head of width 16 would use: splitting changes how the components are combined, not how many weights there are.

10.5. Order and position

Nothing in Definition 10.1 refers to where a key sits in the list. Return to the example of Section 10.3, with $q_t = k_t = v_t = x_t$, and use unmasked self-attention. For the inputs $(1, 2)$ the outputs are

$$y_1 = \frac{e \cdot 1 + e^2 \cdot 2}{e + e^2} \approx 1.731, \qquad y_2 = \frac{e^2 \cdot 1 + e^4 \cdot 2}{e^2 + e^4} \approx 1.881.$$

For the swapped inputs $(2, 1)$ the outputs are $(1.881, 1.731)$: the same two numbers, swapped. Attention saw the same query and the same set of key–value pairs at each token, only listed in a different order, and the order of a sum does not matter.

To state this in general we need a name for reordering. A *permutation* of $\{1, \dots, T\}$ is a function π from $\{1, \dots, T\}$ to itself that takes every value exactly once. It reorders a sequence: the sequence $\mathbf{x}_{\pi(1)}, \dots, \mathbf{x}_{\pi(T)}$ has at position t the vector that was at position $\pi(t)$. Its *permutation matrix* is the $T \times T$ matrix Π with $\Pi_{t,\pi(t)} = 1$ and all other entries 0; row t of Π is $\mathbf{e}_{\pi(t)}^\top$, so row t of ΠX is row $\pi(t)$ of X. For $T = 2$, the swap $\pi(1) = 2$, $\pi(2) = 1$ has permutation matrix $\Pi = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, and the example says that $F(\Pi X) = \Pi F(X)$ when F is unmasked self-attention and $X = (1, 2)^\top$. A map F from $T \times d$ matrices to $T \times d$ matrices is *permutation equivariant* if $F(\Pi X) = \Pi F(X)$ for every $T \times T$ permutation matrix Π: reordering the rows of the input reorders the rows of the output in the same way.

Theorem 10.3 (Permutation equivariance)**.** *Let $d = hd_h$, let W_Q, W_K, W_V, W_O be $d \times d$ matrices, and let F be unmasked multi-head attention (Definition 10.4) with these matrices, written as the map from the $T \times d$ matrix X with rows $\mathbf{x}_t^\top$ to the $T \times d$ matrix $F(X)$ with rows $\mathbf{y}_t^\top$. Then F is permutation equivariant: for every $T \times T$ permutation matrix Π and every $T \times d$ matrix X,*

$$F(\Pi X) = \Pi F(X).$$

Proof. Let $X' = \Pi X$, so $\mathbf{x}'_t = \mathbf{x}_{\pi(t)}$. The queries, keys and values are computed position by position, so $\mathbf{q}'_t = \mathbf{q}_{\pi(t)}$, $\mathbf{k}'_j = \mathbf{k}_{\pi(j)}$ and $\mathbf{v}'_j = \mathbf{v}_{\pi(j)}$, and the same holds for their blocks. For head i,

$$\mathbf{o}'^{(i)}_t = \frac{\sum_{j=1}^{T} e^{\mathbf{q}^{(i)}_{\pi(t)} \cdot \mathbf{k}^{(i)}_{\pi(j)} / \sqrt{d_h}} \, \mathbf{v}^{(i)}_{\pi(j)}}{\sum_{r=1}^{T} e^{\mathbf{q}^{(i)}_{\pi(t)} \cdot \mathbf{k}^{(i)}_{\pi(r)} / \sqrt{d_h}}}.$$

As j runs over $1, \dots, T$, so does $\pi(j)$, each value once, so the sum in the numerator is the same sum as with $\pi(j)$ replaced by j, and the same holds for the denominator. So $\mathbf{o}'^{(i)}_t = \mathbf{o}^{(i)}_{\pi(t)}$ for every head, hence $\mathbf{o}'_t = \mathbf{o}_{\pi(t)}$ and $\mathbf{y}'_t = W_O \mathbf{o}_{\pi(t)} = \mathbf{y}_{\pi(t)}$. Row t of $F(\Pi X)$ is row $\pi(t)$ of $F(X)$, which is row t of $\Pi F(X)$. □

Maps that act on each position separately, such as the MLP of Definition 7.3 applied to every $\mathbf{y}_t$, also commute with Π. So a model built from unmasked attention and such maps, with no information about position, gives the same outputs, reordered, for every reordering of its input. It cannot tell `ab` from `ba`. The causal mask breaks this symmetry only partly.

Corollary 10.1. *Let $\mathbf{y}_1, \dots, \mathbf{y}_T$ be the outputs of causal multi-head attention (Definition 10.4) of $\mathbf{x}_1, \dots, \mathbf{x}_T$, and let $2 \le t \le T$. Then $\mathbf{y}_t$ is unchanged when $\mathbf{x}_1, \dots, \mathbf{x}_{t-1}$ are permuted among themselves.*

Proof. By Theorem 10.2(a), applied to each head, $\mathbf{o}^{(i)}_t$ is the unmasked attention of $\mathbf{q}^{(i)}_t$ over the keys and values at positions $1, \dots, t$. Permuting positions $1, \dots, t-1$ leaves $\mathbf{x}_t$, and so $\mathbf{q}^{(i)}_t$, unchanged, and reorders the key–value pairs. The proof of Theorem 10.3 shows that reordering the pairs does not change the sums in Equation 10.1. □

The book's model has a single layer of attention, and every other operation acts on one position at a time. Without position information, its prediction after position t would depend only on the token at t and on which tokens

came before, not on their order: it would give the same distribution for the letter after `emm` as after `mem`. Position embeddings remove this limitation by giving each position a vector of its own, added to the vector of whatever token sits there.

In the example with $d = 1$, add the number $p_1 = 0$ at position 1 and $p_2 = 1$ at position 2. The inputs $(1, 2)$ become $(1, 3)$, and the swapped inputs $(2, 1)$ become $(2, 2)$, which is not a reordering of $(1, 3)$. Unmasked self-attention now gives the outputs $(2.762, 2.995)$ for the first order and $(2, 2)$ for the second, so the two orders are told apart.

Definition 10.5 (Position embeddings)**.** Let E be a token embedding matrix of width d, as in Definition 9.6, so that token a has embedding $\mathbf{u}_a$, row a of E written as a column. Let $T_{\max}$, the *context length*, be the largest number of positions the model accepts. A *position embedding matrix* is a $T_{\max} \times d$ matrix E_{pos} with rows $\mathbf{p}_1^\top, \ldots, \mathbf{p}_{T_{\max}}^\top$. The input vector at position t of a sequence of tokens $x_1, x_2, \ldots$ is the sum of the token embedding and the position embedding,

$$\mathbf{x}_t = \mathbf{u}_{x_t} + \mathbf{p}_t.$$

Row t of E_{pos} is the vector added at position t, whatever token sits there. Permuting the tokens now moves the vectors $\mathbf{u}_{x_t}$ but leaves the $\mathbf{p}_t$ in place, so the new inputs are not a permutation of the old ones, and the conclusion of Theorem 10.3 does not apply to them. The book's model, like microgpt, learns E_{pos} along with the other parameters, with $T_{\max} = 16$ and $d = 16$, so 256 parameters; microgpt adds the two embeddings and then normalises the sum (Chapter 11). The original transformer used fixed vectors built from sines and cosines instead, and its authors report that learned position embeddings gave nearly identical results (Vaswani et al. 2017, sec. 3.5).

10.6. How gradients pass through attention

Training needs the derivative of the loss with respect to every parameter, and Chapter 4 computes it by passing adjoints back through each operation. For attention, the derivative with respect to the query shows the pattern. Start with the smallest case: $d_h = 1$, keys $k_1 = 1$ and $k_2 = -1$, values $v_1 = 1$ and $v_2 = 0$. At $q = 0$ both scores are 0, the weights are $\frac{1}{2}$ each, and the output is $y = \frac{1}{2}$. At $q = 0.01$ the scores are ± 0.01, the first weight rises to about 0.505, and so does y: the output changes at a rate of about 0.5 per unit increment of q. The increment raised the score of the key that is above average ($k_1 = 1$, against the average key 0), and that key's value is above the average output. The theorem below says that the derivative is always built from these two deviations.

Theorem 10.4 (Jacobian of attention with respect to the query)**.** *Fix keys $\mathbf{k}_1, \dots, \mathbf{k}_T \in \mathbb{R}^{d_h}$ and values $\mathbf{v}_1, \dots, \mathbf{v}_T \in \mathbb{R}^{d_v}$, the rows of matrices K and V, and let $\mathbf{y}(\mathbf{q}) = \mathrm{Attn}(\mathbf{q}; \mathbf{k}_{1:T}, \mathbf{v}_{1:T})$ with weights $\mathbf{a}$. Then $\mathbf{y}$ is differentiable at every $\mathbf{q} \in \mathbb{R}^{d_h}$, and with $\mathbf{c} = \sum_j a_j \mathbf{k}_j$, the weighted average of the keys,*

$$J_{\mathbf{y}}(\mathbf{q}) = \frac{1}{\sqrt{d_h}} V^\top (\mathrm{diag}(\mathbf{a}) - \mathbf{a}\mathbf{a}^\top) K = \frac{1}{\sqrt{d_h}} \sum_{j=1}^{T} a_j \, (\mathbf{v}_j - \mathbf{y})(\mathbf{k}_j - \mathbf{c})^\top. \tag{10.4}$$

The Jacobian is a $d_v \times d_h$ matrix: entry (i, l) is the rate at which component i of the output changes per unit increment of component l of the query. The second form has the shape of a covariance, an average of products of deviations: each position contributes the deviation of its value from the output, times the deviation of its key from the average key, weighted by its attention weight.

Proof. The map $\mathbf{y}$ is a composition of three maps: $\mathbf{q} \mapsto \mathbf{s} = K\mathbf{q}/\sqrt{d_h}$, which is linear with Jacobian $K/\sqrt{d_h}$ (Example 1.6); then $\mathbf{s} \mapsto \mathbf{a} =$

softmax($\mathbf{s}$), with Jacobian $\operatorname{diag}(\mathbf{a}) - \mathbf{a}\mathbf{a}^\top$ by Theorem 8.4; then $\mathbf{a} \mapsto V^\top \mathbf{a}$, linear with Jacobian $V^\top$. By the chain rule, Theorem 2.1, the composition is differentiable and its Jacobian is the product, the first form in Equation 10.4.

For the second form, expand. Since $V^\top \operatorname{diag}(\mathbf{a}) K = \sum_j a_j \mathbf{v}_j \mathbf{k}_j^\top$, $V^\top \mathbf{a} = \mathbf{y}$ and $K^\top \mathbf{a} = \mathbf{c}$, the first form is $\frac{1}{\sqrt{d_h}}(\sum_j a_j \mathbf{v}_j \mathbf{k}_j^\top - \mathbf{y}\mathbf{c}^\top)$. Using $\sum_j a_j = 1$,

$$\sum_j a_j (\mathbf{v}_j - \mathbf{y})(\mathbf{k}_j - \mathbf{c})^\top = \sum_j a_j \mathbf{v}_j \mathbf{k}_j^\top - \mathbf{y}\mathbf{c}^\top - \mathbf{y}\mathbf{c}^\top + \mathbf{y}\mathbf{c}^\top,$$

which is the same matrix. □

In the opening example, $c = 0$, $y = \frac{1}{2}$, and the formula gives $\frac{1}{2} \cdot \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot (-\frac{1}{2})(-1) = \frac{1}{2}$, the rate found from the increment. If all keys are equal, $\mathbf{k}_j = \mathbf{c}$ for every j and the Jacobian is zero: the weights are uniform for every query. If the weights are concentrated on one position, the Jacobian is small.

Corollary 10.2. *Let $\mathbf{y}(\mathbf{q})$ be attention with keys $\mathbf{k}_1, \dots, \mathbf{k}_T \in \mathbb{R}^{d_h}$ and values $\mathbf{v}_1, \dots, \mathbf{v}_T$, and suppose that $\|\mathbf{k}_j\| \le B_k$ and $\|\mathbf{v}_j\| \le B_v$ for all j. If at the query $\mathbf{q}$ some weight satisfies $a_m \ge 1 - \varepsilon$, with $0 \le \varepsilon \le 1$, then every entry of $J_{\mathbf{y}}(\mathbf{q})$ has absolute value at most $8\varepsilon B_k B_v / \sqrt{d_h}$.*

Proof. An entry of a matrix $\mathbf{u}\mathbf{w}^\top$ is $u_i w_l$, of absolute value at most $\|\mathbf{u}\| \, \|\mathbf{w}\|$. The vectors $\mathbf{y}$ and $\mathbf{c}$ are convex combinations, so $\|\mathbf{y}\| \le B_v$ and $\|\mathbf{c}\| \le B_k$ by the triangle inequality, and so $\|\mathbf{v}_j - \mathbf{y}\| \le 2B_v$ and $\|\mathbf{k}_j - \mathbf{c}\| \le 2B_k$. The terms with $j \ne m$ in Equation 10.4 have weights summing to $1 - a_m \le \varepsilon$, so together they contribute at most $4\varepsilon B_k B_v$ to each entry. For $j = m$, $\mathbf{v}_m - \mathbf{y} = \sum_{j \ne m} a_j (\mathbf{v}_m - \mathbf{v}_j)$, so $\|\mathbf{v}_m - \mathbf{y}\| \le 2\varepsilon B_v$, and the term contributes at most $a_m \cdot 2\varepsilon B_v \cdot 2B_k \le 4\varepsilon B_k B_v$. Dividing by $\sqrt{d_h}$ gives the bound. □

In reverse mode (Chapter 4) the adjoint of the output, $\bar{\mathbf{y}} = \partial L/\partial \mathbf{y}$, becomes the adjoint of the query by multiplying by the transpose of the Jacobian:

$$\bar{\mathbf{q}} = J_{\mathbf{y}}^{\top}\bar{\mathbf{y}} = \frac{1}{\sqrt{d_h}} \sum_{j=1}^{T} a_j \left((\mathbf{v}_j - \mathbf{y}) \cdot \bar{\mathbf{y}}\right)(\mathbf{k}_j - \mathbf{c}).$$

Each position contributes its key's deviation from the average key, weighted by its attention weight and by how much its value's deviation from the output agrees with $\bar{\mathbf{y}}$. A gradient step moves the query by $-\eta\bar{\mathbf{q}}$, so towards the keys whose values would move the output in the direction that lowers the loss, and away from the others. Corollary 10.2 and Table 10.1 show the same thing from two sides: when the weights saturate, this signal fades. Exercise 10.6 gives the derivatives with respect to the keys and values.

10.7. Cost

How much arithmetic does one layer of attention need, and how does it grow with the number of positions? Count multiplications for multi-head attention on T positions with width d. The four matrices cost d^2 multiplications per position each, $4Td^2$ in all. Each head computes T^2 scores, dot products of length d_h, and then T weighted sums of T value vectors of length d_h: $2T^2d_h$ multiplications per head, and $2T^2d$ over the h heads. The total is

$$4Td^2 + 2T^2d.$$

The first term grows linearly with T and the second quadratically, so attention takes $O(T^2d)$ time (at most a constant multiple of T^2d operations), on top of the $O(Td^2)$ for the matrices, and storing the weights takes T^2 numbers per head. The causal mask halves the attention term, since only the $T(T+1)/2$ pairs with $j \le t$ are needed. The quadratic term overtakes the linear one when $T > 2d$. For the book's model, $T \le 16$ and $d = 16$, so the matrices cost $4 \cdot 16 \cdot 16^2 = 16{,}384$ multiplications and the masked attention $16 \cdot 17 \cdot 16 = 4352$: at this size the cost is in the matrices. For

long documents the T^2 term dominates, and much work on large models goes into reducing it.

10.8. Notes and further reading

Attention was introduced for translation by Bahdanau et al. (2014), with scores from a small network rather than a dot product. Vaswani et al. (2017) built the transformer from scaled dot-product attention, with several heads, a causal mask for generation and position encodings, and their footnote 4 gives the variance argument of Theorem 10.1. Jurafsky and Martin (2026, ch. 7) explain attention, causal masking and multi-head attention in the setting of language modelling. Karpathy (2026a) describes the small model this book rebuilds, including its key–value cache and learned position embeddings. Grinstead and Snell (1997, sec. 6.1 and §6.2) treat expectation, variance and independence for discrete random variables, and Billingsley (1995, sec. 20 and §21) gives the general theory.

10.9. Exercises

Exercise 10.1. Let $d_h = 1$, $q = 2$, keys $k_1 = 1$, $k_2 = 0$, $k_3 = -1$ and values $v_1 = 1$, $v_2 = 2$, $v_3 = 3$. Find the attention weights and the output, exactly and to three decimal places.

Exercise 10.2.

(a) Show that adding the same vector $\mathbf{b}$ to every key leaves the attention weights unchanged.

(b) Show that adding the same vector $\mathbf{w}$ to every value adds $\mathbf{w}$ to the output.

Exercise 10.3. Fix a query, keys and values, and replace the scores s_j by λs_j with $\lambda > 0$. Let M be the set of j with s_j equal to the largest score. Show that as $\lambda \to \infty$ the weights tend to $1/|M|$ for $j \in M$ and to 0 otherwise, so the output tends to the average of the values $\mathbf{v}_j$ with $j \in M$. What do the weights tend to as $\lambda \to 0$?

Exercise 10.4.

(a) Generalise Theorem 10.1: keep its hypotheses except that $\mathrm{Var}[q_i] = \sigma^2$ and $\mathrm{Var}[k_i] = \tau^2$ for every i, and show that $\mathrm{Var}[\mathbf{q} \cdot \mathbf{k}] = d_h \sigma^2 \tau^2$.

(b) In the book's model, at initialisation each entry of W_Q and W_K is drawn independently from the normal distribution with mean 0 and standard deviation 0.08, and after RMS normalisation (Chapter 11) the input vectors to attention have squared norm close to 16. Treat a query $\mathbf{q} = W_Q \mathbf{x}$ and a key $\mathbf{k} = W_K \mathbf{x}'$ as coming from fixed vectors with $\|\mathbf{x}\|^2 = \|\mathbf{x}'\|^2 = 16$. Find the variance of each component of $\mathbf{q}$ and of $\mathbf{k}$, and then the variance of a score in a head of width 4. What does this say about the attention weights at the start of training?

Exercise 10.5. A model generates T tokens one at a time with one head of causal attention. Count the dot products $\mathbf{q}_t \cdot \mathbf{k}_j$ it computes (a) with a key–value cache, and (b) if at each step it recomputes attention for every position so far from scratch. Evaluate both for $T = 16$.

Exercise 10.6. In the setting of Theorem 10.4, show that

$$\frac{\partial \mathbf{y}}{\partial \mathbf{v}_j} = a_j I \qquad \text{and} \qquad \frac{\partial \mathbf{y}}{\partial \mathbf{k}_j} = \frac{a_j}{\sqrt{d_h}} \left(\mathbf{v}_j - \mathbf{y}\right) \mathbf{q}^\top,$$

where each is the Jacobian of $\mathbf{y}$ as a function of $\mathbf{v}_j$ or $\mathbf{k}_j$ alone, with everything else fixed.

Exercise 10.7. Take $T = 2$, $d = d_h = 1$ and $W_Q = W_K = W_V = 1$, so $q_t = k_t = v_t = x_t$. Compute the causal self-attention outputs for the input $(x_1, x_2) = (1, 2)$ and for the swapped input $(2, 1)$, and show that the outputs for the swapped input are not the swapped outputs.

Exercise 10.8. Write $W_O = (W_O^{(1)} \ \cdots \ W_O^{(h)})$, where $W_O^{(i)}$ consists of columns $(i-1)d_h + 1$ to id_h. Show that $W_O \mathbf{o}_t = \sum_{i=1}^{h} W_O^{(i)} \mathbf{o}_t^{(i)}$, so multi-head attention is a sum of h single-head attentions, each with its own output matrix.

11. A small transformer

This chapter puts the parts of the book together into one model: a language model with 4192 parameters that learns to generate names one letter at a time. It is a *transformer*, a network in which attention (Chapter 10) lets each position of a sequence draw on earlier positions, and an MLP then processes each position on its own. It has the shape of Andrej Karpathy's microgpt (Karpathy 2026a), a single file of Python that defines the model and trains it with Adam, computing each gradient by reverse mode on a graph of linked nodes (Section 4.8). We have checked every architectural detail stated below against that source (Karpathy 2026b).

Two ingredients are new. Call each stage of the network that has its own parameters, such as the attention stage or the MLP, a *block*. *RMS normalisation* rescales a vector to a fixed size before a block uses it, and *residual connections* add each block's output to its input instead of replacing it. Both change how gradients flow, and both have short exact descriptions in terms of Jacobians. The rest of the chapter writes down the complete model, counts its parameters, explains the value of the loss before training and the effect of the sampling temperature, and proves that storing the keys and values of earlier positions during generation computes exactly what causal attention computes.

11.1. RMS normalisation

The vectors inside a network change size as they pass through it. A linear map with large weights stretches them, and a sum of many terms grows

with the number of terms. If the next layer is to receive inputs on the same scale whatever happened before, something has to divide each vector by a measure of its size. The question is which measure, and what to do with vectors whose size is close to zero.

A natural measure for a vector of n components is the *root mean square* of its components: the square root of the mean of their squares. For $\mathbf{x} = (1, -1, 2, 0)$ the squares are $1, 1, 4, 0$, their mean is 1.5, and the root mean square is $\sqrt{1.5} \approx 1.2247$. Dividing by it gives

$$\frac{\mathbf{x}}{\sqrt{1.5}} \approx (0.8165, -0.8165, 1.6330, 0),$$

a vector whose components have root mean square exactly 1. The vector $10\mathbf{x} = (10, -10, 20, 0)$ has root mean square $10\sqrt{1.5}$, and dividing gives the same result: the division removes the size of the vector and keeps its direction.

The obvious definition, $\mathbf{x}$ divided by its root mean square, fails at $\mathbf{0}$, where it divides by zero, and it cannot be repaired there by a choice of value. Along the line $\mathbf{x} = (t, 0, 0, 0)$ the root mean square is $|t|/2$, so the quotient is $(2, 0, 0, 0)$ for every $t > 0$ and $(-2, 0, 0, 0)$ for every $t < 0$: it jumps as t passes through 0, and so it is not continuous, let alone differentiable, at $\mathbf{0}$. A gradient method needs a differentiable function, and the remedy is to add a small positive constant under the square root.

Definition 11.1 (RMS normalisation)**.** Let $n \geq 1$ and $\varepsilon > 0$. For $\mathbf{x} \in \mathbb{R}^n$ let $\mu(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} x_i^2 = \|\mathbf{x}\|^2/n$, the mean of the squares of the components. The *RMS normalisation* of $\mathbf{x}$ is

$$\operatorname{rms}(\mathbf{x}) = \frac{\mathbf{x}}{\sqrt{\mu(\mathbf{x}) + \varepsilon}}.$$

In words: divide every component of $\mathbf{x}$ by the same positive number, the root mean square of the components with ε added under the square root. The denominator is at least $\sqrt{\varepsilon}$, so rms is defined on all of $\mathbb{R}^n$, and

$\mathrm{rms}(\mathbf{0}) = \mathbf{0}$. The model of this chapter uses $n = 16$ and $\varepsilon = 10^{-5}$, and multiplies the result by nothing: there is no learned gain, a vector of weights that some models multiply the result by componentwise. Since $\mu(\mathrm{rms}(\mathbf{x})) = \mu(\mathbf{x})/(\mu(\mathbf{x})+\varepsilon)$, the root mean square of the output is slightly below 1, and far below 1 when $\mu(\mathbf{x})$ is comparable with ε.

Example 11.1. For $\mathbf{x} = (1, -1, 2, 0)$ and $\varepsilon = 10^{-5}$, the denominator is $\sqrt{1.50001}$ instead of $\sqrt{1.5}$, so each component of the result above changes by a factor $\sqrt{1.5/1.50001} \approx 1 - 3.3 \times 10^{-6}$. For $\mathbf{x} = (t, 0, 0, 0)$ the result is $(t/\sqrt{t^2/4 + \varepsilon}, 0, 0, 0)$, which passes continuously through $\mathbf{0}$ at $t = 0$ and is close to $(\pm 2, 0, 0, 0)$ once $|t|$ is large compared with $\sqrt{\varepsilon}$.

11.1.1. The Jacobian

Reverse mode (Theorem 4.1) passes the gradient of the loss back through every function in the model, and for that it needs each function's Jacobian. For RMS normalisation the Jacobian treats increments in two directions differently, and a small case shows how. Take $n = 2$, a deliberately large $\varepsilon = 1$ so that the effect is visible, and $\mathbf{x} = (1, 0)$. Then $\mu(\mathbf{x}) = 1/2$ and $\mathrm{rms}(\mathbf{x}) = (1, 0)/\sqrt{3/2} \approx (0.8165, 0)$.

- An increment of 0.01 in x_2, perpendicular to $\mathbf{x}$, changes the output by about $(0.0000, 0.008165)$: the increment passes through multiplied by $0.8165 = (3/2)^{-1/2}$, because to first order it does not change μ.
- An increment of 0.01 in x_1, along $\mathbf{x}$, changes the output by about $(0.005416, 0)$: a factor 0.54, smaller, because enlarging $\mathbf{x}$ also enlarges the denominator.

With $\varepsilon = 0$ the second factor would be exactly 0, since then scaling $\mathbf{x}$ does not change the output at all. The theorem gives both factors in general.

Theorem 11.1 (Jacobian of RMS normalisation)**.** *Let $n \geq 1$ and $\varepsilon > 0$, let* rms *be RMS normalisation on $\mathbb{R}^n$ with this ε, and write $s(\mathbf{x}) = (\mu(\mathbf{x}) +$*

$\varepsilon)^{-1/2}$. *Then* rms *is differentiable on* $\mathbb{R}^n$, *with*

$$J_{\mathrm{rms}}(\mathbf{x}) = s\,I - \frac{s^3}{n}\,\mathbf{x}\mathbf{x}^\top = s\Big(I - \frac{\mathbf{x}\mathbf{x}^\top}{\|\mathbf{x}\|^2 + n\varepsilon}\Big), \qquad s = s(\mathbf{x}). \tag{11.1}$$

The matrix is symmetric. For $\mathbf{x} \neq \mathbf{0}$, *every non-zero vector perpendicular to* $\mathbf{x}$ *is an eigenvector with eigenvalue* s, *and* $\mathbf{x}$ *is an eigenvector with eigenvalue* εs^3.

Entry (i, j) of this matrix is the rate at which output component i changes per unit increment of input component j. The term sI says that each input component, on its own, passes to the same output component scaled by s; the term $-(s^3/n)\mathbf{x}\mathbf{x}^\top$, whose entry (i, j) is $-s^3 x_i x_j/n$, corrects every output component for the change in the common denominator. In the example above, $s = (3/2)^{-1/2} \approx 0.8165$ and $\varepsilon s^3 = (2/3)^{3/2} \approx 0.5443$.

Proof. Component i of $\mathrm{rms}(\mathbf{x})$ is $x_i\, s(\mathbf{x})$. The function μ is a polynomial and $\mu + \varepsilon \geq \varepsilon > 0$, so s has continuous partial derivatives

$$\frac{\partial s}{\partial x_j} = -\frac{1}{2}(\mu + \varepsilon)^{-3/2}\,\frac{2x_j}{n} = -\frac{s^3 x_j}{n}.$$

By the product rule, $\partial(x_i s)/\partial x_j = \delta_{ij} s - s^3 x_i x_j/n$, where δ_{ij} is 1 if $i = j$ and 0 otherwise. These partial derivatives are continuous, so by Theorem 1.6 and Theorem 1.5 rms is differentiable with the Jacobian in Equation 11.1. The second form follows from $s^2 = n/(\|\mathbf{x}\|^2 + n\varepsilon)$.

If $\mathbf{y} \cdot \mathbf{x} = 0$ then $J\mathbf{y} = s\mathbf{y} - (s^3/n)\mathbf{x}(\mathbf{x} \cdot \mathbf{y}) = s\mathbf{y}$. For $\mathbf{x}$ itself,

$$J\mathbf{x} = s\mathbf{x} - \frac{s^3}{n}\|\mathbf{x}\|^2\mathbf{x} = s^3\Big(\frac{1}{s^2} - \mu\Big)\mathbf{x} = \varepsilon s^3\,\mathbf{x},$$

because $1/s^2 = \mu + \varepsilon$. □

In reverse mode every intermediate vector $\mathbf{v}$ carries an adjoint $\bar{\mathbf{v}}$ (Definition 4.1), the gradient of the loss L with respect to $\mathbf{v}$, and reverse mode (Theorem 4.1) obtains the adjoint of a function's input by multiplying the adjoint of its output by the transposed Jacobian. Write $\mathbf{y} = \mathrm{rms}(\mathbf{x})$. Since J is symmetric, $\bar{\mathbf{x}} = J^\top \bar{\mathbf{y}} = J\bar{\mathbf{y}}$, that is,

$$\bar{\mathbf{x}} = s\,\bar{\mathbf{y}} - \frac{s^3}{n}\,(\mathbf{x}\cdot\bar{\mathbf{y}})\,\mathbf{x}.$$

The gradient keeps its component perpendicular to $\mathbf{x}$, scaled by s, and almost loses its component along $\mathbf{x}$, which J scales by εs^3 instead. A change of $\mathbf{x}$ along itself only changes its size, and normalisation removes size.

11.1.2. Scaling and the effect of ε

The point of normalising is that the output should not depend on the size of the input. With $\varepsilon = 0$ that would hold exactly; the question is how much ε spoils it. Take again $\mathbf{x} = (1, -1, 2, 0)$, with $\mu(\mathbf{x}) = 1.5$, and $\varepsilon = 10^{-5}$. For $10\mathbf{x}$ the denominator is $\sqrt{150 + 10^{-5}} = 10\sqrt{1.5 + 10^{-7}}$, so the output is $\mathbf{x}/\sqrt{1.5 + 10^{-7}}$: the same as normalising $\mathbf{x}$ with ε replaced by 10^{-7}. For $0.001\mathbf{x}$ the output is $\mathbf{x}/\sqrt{1.5 + 10}$, the same as normalising $\mathbf{x}$ with $\varepsilon = 10$, and it is only $\sqrt{1.5/11.5} \approx 0.36$ times the output for $\mathbf{x}$ itself. Write rms_ε when the value of ε matters.

Theorem 11.2 (Scaling)**.** *Let $\varepsilon > 0$, $c > 0$ and $\mathbf{x} \in \mathbb{R}^n$.*

(a) $\mathrm{rms}_\varepsilon(c\mathbf{x}) = \mathrm{rms}_{\varepsilon/c^2}(\mathbf{x})$.

(b) If $\mathbf{x} \neq \mathbf{0}$, then $\mathrm{rms}_\varepsilon(\mathbf{x}) = \lambda\,\mathbf{x}/\sqrt{\mu(\mathbf{x})}$ *with*

$$\lambda = \sqrt{\frac{\mu(\mathbf{x})}{\mu(\mathbf{x}) + \varepsilon}}, \qquad 1 - \frac{\varepsilon}{2\mu(\mathbf{x})} \le \lambda < 1.$$

Proof.

(a) $\mu(c\mathbf{x}) = c^2\mu(\mathbf{x})$, so

$$\mathrm{rms}_\varepsilon(c\mathbf{x}) = \frac{c\mathbf{x}}{\sqrt{c^2\mu(\mathbf{x}) + \varepsilon}} = \frac{c\mathbf{x}}{c\sqrt{\mu(\mathbf{x}) + \varepsilon/c^2}} = \mathrm{rms}_{\varepsilon/c^2}(\mathbf{x}).$$

(b) The formula for λ is a rearrangement of the definition, and $\lambda < 1$ because $\varepsilon > 0$. Put $r = \varepsilon/\mu(\mathbf{x}) > 0$, so that $\lambda^2 = 1/(1+r)$. If $r > 2$ the lower bound $1 - r/2$ is negative and there is nothing to prove. If $0 < r \le 2$, then $1 - r/2 \ge 0$ and

$$\left(1 - \frac{r}{2}\right)^2 (1 + r) = 1 - \frac{r^2(3-r)}{4} \le 1,$$

so $(1 - r/2)^2 \le \lambda^2$, and taking square roots gives $1 - r/2 \le \lambda$.

□

So normalisation is invariant under positive scaling exactly when $\varepsilon = 0$. With $\varepsilon > 0$, scaling $\mathbf{x}$ by c has the same effect as replacing ε by ε/c^2: enlarging the input weakens ε, and shrinking it strengthens ε. By (b), ε shrinks the output by a relative amount of at most $\varepsilon/(2\mu(\mathbf{x}))$. For inputs whose mean square is large compared with ε this is negligible. For inputs of mean square comparable with ε it is not, and as $\mathbf{x} \to \mathbf{0}$ the output tends to $\mathbf{0}$ instead of jumping between vectors of root mean square 1. That continuity at $\mathbf{0}$ is what ε buys.

In the model of Section 11.3, the first vector normalised is the sum of two vectors of 16 parameters each (a row of the token embedding and a row of the position embedding). Before training, every parameter is drawn independently with mean 0 and variance 0.08^2, so each component of the sum has variance $2 \times 0.08^2 = 0.0128$. Then $\mu \approx 0.0128$ and $\varepsilon/(2\mu) \approx 3.9 \times 10^{-4}$: at the start of training, ε shrinks this vector by less than 0.04%.

11.1.3. Layer normalisation

RMS normalisation is a simplification of the older *layer normalisation* (Ba et al. 2016), which GPT-2, the language model whose design this chapter's model follows, uses (Radford et al. 2019). The difference shows when the same constant is added to every component. Ignoring ε, RMS normalisation sends $(1, 2, 3)$ to about $(0.463, 0.926, 1.389)$ and $(11, 12, 13)$ to about $(0.915, 0.998, 1.081)$: the shift by 10 changes the result. Layer normalisation first subtracts the mean of the components, which turns both vectors into $(-1, 0, 1)$, and then divides by the root mean square of what remains, $\sqrt{2/3}$, giving $(-1.225, 0, 1.225)$ for both.

Definition 11.2 (Layer normalisation)**.** Let $n \geq 1$ and $\varepsilon > 0$. For $\mathbf{x} \in \mathbb{R}^n$ let $m(\mathbf{x}) = \frac{1}{n}\sum_i x_i$ be the mean of the components and $v(\mathbf{x}) = \frac{1}{n}\sum_i (x_i - m(\mathbf{x}))^2$ their variance. The *layer normalisation* of $\mathbf{x}$, with *gain* $\mathbf{g}$ and *bias* $\mathbf{b}$ in $\mathbb{R}^n$, is the vector with components

$$\mathrm{LN}(\mathbf{x})_i = g_i \, \frac{x_i - m(\mathbf{x})}{\sqrt{v(\mathbf{x}) + \varepsilon}} + b_i.$$

In words: subtract the mean, divide by the standard deviation (with ε added to the variance), then scale component i by g_i and shift it by b_i. In a network $\mathbf{g}$ and $\mathbf{b}$ are parameters, learned with the rest. With $\mathbf{g} = \mathbf{1}$ and $\mathbf{b} = \mathbf{0}$ it is RMS normalisation applied to the centred vector $\mathbf{x} - m(\mathbf{x})\mathbf{1}$, because the variance of $\mathbf{x}$ is the mean square of the centred vector (Exercise 11.3). So layer normalisation is unchanged when the same constant is added to every component of $\mathbf{x}$, and RMS normalisation is not; on vectors with mean zero the two agree. Zhang and Sennrich (2019) propose RMS normalisation on the hypothesis that the rescaling, not the centring, is what helps training, and they report comparable accuracy at lower cost. microgpt drops the gain as well, so its normalisations have no parameters.

11.2. Residual connections

A deep network is a composition of many functions, one per block, and by the chain rule (Theorem 2.1) the derivative of the loss with respect to the input of the first block is a product of one Jacobian per block. A product of many factors can be very small. In one dimension, ten blocks $x \mapsto 0.1x$ in a row have derivative $0.1^{10} = 10^{-10}$: an increment at the bottom of the stack has almost no effect at the top, and gradient descent barely changes the parameters of the early blocks. Now let each block add its output to its input instead, $x \mapsto x + 0.1x$. Ten of these have derivative $1.1^{10} \approx 2.59$, of the same order as 1, because each factor is 1 plus something small instead of something small.

Definition 11.3 (Residual connection)**.** Let $U \subseteq \mathbb{R}^n$ and $F\colon U \to \mathbb{R}^n$. The *residual block* built from F is the map $R\colon U \to \mathbb{R}^n$, $R(\mathbf{x}) = \mathbf{x} + F(\mathbf{x})$. The added term $\mathbf{x}$ is the *residual connection*, and F is the *block*.

The block F computes a correction to its input instead of a replacement for it. He et al. (2016) introduced residual connections for image recognition, where they made networks with over a hundred layers trainable. Their effect on derivatives takes one line.

Theorem 11.3 (Jacobian of a residual block)**.** *Let $U \subseteq \mathbb{R}^n$ be open, let $F\colon U \to \mathbb{R}^n$ be differentiable at $\mathbf{a} \in U$, and let $R(\mathbf{x}) = \mathbf{x} + F(\mathbf{x})$. Then R is differentiable at $\mathbf{a}$ and*

$$J_R(\mathbf{a}) = I + J_F(\mathbf{a}).$$

If also $\ell\colon \mathbb{R}^n \to \mathbb{R}$ is differentiable at $R(\mathbf{a})$ and $L(\mathbf{x}) = \ell(R(\mathbf{x}))$, then with $\bar{\mathbf{y}} = \nabla\ell(R(\mathbf{a}))$,

$$\nabla L(\mathbf{a}) = \bar{\mathbf{y}} + J_F(\mathbf{a})^\top \bar{\mathbf{y}}. \tag{11.2}$$

Proof. For $\mathbf{h}$ small enough that $\mathbf{a}+\mathbf{h} \in U$,

$$R(\mathbf{a}+\mathbf{h}) - R(\mathbf{a}) - (I + J_F(\mathbf{a}))\mathbf{h} = F(\mathbf{a}+\mathbf{h}) - F(\mathbf{a}) - J_F(\mathbf{a})\mathbf{h}.$$

The right side is the remainder of F, which divided by $\|\mathbf{h}\|$ tends to $\mathbf{0}$, so $I + J_F(\mathbf{a})$ satisfies Equation 1.2 for R. By Theorem 2.1, $J_L(\mathbf{a}) = J_\ell(R(\mathbf{a}))\,(I + J_F(\mathbf{a}))$, and transposing this row gives Equation 11.2. □

In the language of reverse mode, the adjoint $\bar{\mathbf{y}}$ arriving at the output of the block reaches the input unchanged along the identity path, and the block adds its own contribution $J_F^\top \bar{\mathbf{y}}$. However small J_F is, even zero, the gradient passes through. It can vanish only if the block's contribution cancels it exactly, $J_F^\top \bar{\mathbf{y}} = -\bar{\mathbf{y}}$, and a block with $\|J_F\|_F < 1$ (in the norm defined next) cannot do that (Exercise 11.8). The difference matters when a network stacks many blocks. To make it quantitative we need a measure of the size of a matrix, and the Frobenius norm of Lemma 1.1 serves: for an $m \times n$ matrix $A = (a_{ij})$, $\|A\|_F = \left(\sum_{i,j} a_{ij}^2\right)^{1/2}$, the Euclidean norm of the mn entries written as one long vector, and $\|A\mathbf{x}\| \le \|A\|_F \, \|\mathbf{x}\|$ for every $\mathbf{x} \in \mathbb{R}^n$. For example, $A = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$ has $\|A\|_F = \sqrt{6}$, and $A^2 = \begin{pmatrix} 1 & 4 \\ 0 & 1 \end{pmatrix}$ has $\|A^2\|_F = \sqrt{18} \approx 4.24$, below $\|A\|_F^2 = 6$. The norm of a product never exceeds the product of the norms.

Lemma 11.1. *For matrices A $(m \times n)$ and B $(n \times p)$, $\|AB\|_F \le \|A\|_F \, \|B\|_F$, and $\|A^\top\|_F = \|A\|_F$.*

Proof. Column j of AB is $A\mathbf{b}_j$, where $\mathbf{b}_j$ is column j of B. By Lemma 1.1, $\|A\mathbf{b}_j\|^2 \le \|A\|_F^2 \|\mathbf{b}_j\|^2$, and summing over j gives $\|AB\|_F^2 \le \|A\|_F^2 \|B\|_F^2$. The second claim holds because A and $A^\top$ have the same entries. □

Corollary 11.1 (Stacked residual blocks)**.** *Let $R_k(\mathbf{x}) = \mathbf{x} + F_k(\mathbf{x})$ for $k = 1, \dots, K$, with each F_k differentiable on $\mathbb{R}^n$. Fix $\mathbf{x}_0$, let $\mathbf{x}_k = R_k(\mathbf{x}_{k-1})$, let $J_k = J_{F_k}(\mathbf{x}_{k-1})$, and let $G = R_K \circ \dots \circ R_1$.*

(a) *$J_G(\mathbf{x}_0) = (I + J_K)\cdots(I + J_1) = \sum_S J_S$, where the sum is over all 2^K subsets S of $\{1, \dots, K\}$ and J_S is the product of the J_k with $k \in S$ in decreasing order of k, with $J_\emptyset = I$.*

(b) *If $\|J_k\|_F \le \delta_k$ for each k, then*

$$\|J_G(\mathbf{x}_0) - I\|_F \le \prod_{k=1}^{K}(1 + \delta_k) - 1 \le e^{\delta_1 + \cdots + \delta_K} - 1.$$

(c) *Under the hypothesis of (b), for a loss $L(\mathbf{x}) = \ell(G(\mathbf{x}))$ with $\ell\colon \mathbb{R}^n \to \mathbb{R}$ differentiable at $\mathbf{x}_K$ and $\bar{\mathbf{x}}_K = \nabla\ell(\mathbf{x}_K)$, the gradient $\bar{\mathbf{x}}_0 = \nabla L(\mathbf{x}_0)$ satisfies $\|\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}_K\| \le (e^{\delta_1 + \cdots + \delta_K} - 1)\|\bar{\mathbf{x}}_K\|$.*

Proof.

(a) The first equality is Theorem 2.1 applied $K - 1$ times with Theorem 11.3. Expanding the product by distributivity chooses, in each factor, either I or J_k, and the choices correspond to subsets S (the set of k where the expansion takes J_k).

(b) By (a), $J_G - I$ is the sum of J_S over non-empty S. The Frobenius norm is the Euclidean norm of the matrix written as one long vector, so it satisfies the triangle inequality, and with Lemma 11.1,

$$\|J_G - I\|_F \le \sum_{S \neq \emptyset} \prod_{k \in S} \delta_k = \prod_{k=1}^{K}(1 + \delta_k) - 1,$$

where the equality is the same expansion of a product, now of numbers. Finally $1 + \delta \le e^\delta$ for every real δ.

(c) $\bar{\mathbf{x}}_0 = J_G^\top \bar{\mathbf{x}}_K$ by Theorem 2.1, so $\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}_K = (J_G - I)^\top \bar{\mathbf{x}}_K$, and Lemma 1.1 with Lemma 11.1 and (b) gives the bound.

□

Part (a) is the chain rule as a sum over paths (Theorem 2.2): each subset S is a route through the stack that enters the blocks in S and skips the others, and the route that skips every block carries the identity. Without the identity paths the Jacobian would be the single product $J_K \cdots J_1$, whose norm is at most $\delta_1 \cdots \delta_K$; when every δ_k is at most some $\delta < 1$ this is at most δ^K, which tends to 0 as K grows, and gradients at the bottom of a deep stack become too small to train on. With residual connections, part (c) says that small blocks leave the gradient nearly unchanged instead.

Example 11.2. Take $n = 2$, $K = 2$ and two blocks with the same Jacobian $N = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$. Without residual connections the Jacobian of the composition is $N^2 = 0$: no gradient at all reaches the input, although each block alone has a non-zero Jacobian. With them it is $(I+N)^2 = I + 2N = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$, which is invertible.

11.3. The complete model

The previous chapters supply every part of the model; this section says exactly how they fit together, as one function from a sequence of symbols to predictions. As in Chapter 9, the symbols are tokens, the elements of a finite vocabulary V (Definition 9.1). Here $|V| = 27$: the letters a to z, numbered 0 to 25 in alphabetical order, and the boundary token $\diamond$, numbered 26, which marks the start and the end of a name. The name `emma` becomes the sequence $\diamond, \mathtt{e}, \mathtt{m}, \mathtt{m}, \mathtt{a}, \diamond$, that is, $26, 4, 12, 12, 0, 26$. At each position the model produces a vector of 27 logits, one real number per token, and the softmax of that vector, $\mathrm{softmax}(\mathbf{z})_i = e^{z_i} / \sum_j e^{z_j}$ (see Definition 8.5), is the model's probability distribution for the next token. A larger logit means a more probable token, and only differences between logits matter (Theorem A.3).

The other parts come from chapters 7 to 10.

- A token embedding (Definition 9.6) represents each token by a learned vector, here in $\mathbb{R}^{16}$; the width $d = 16$ is the length of the vector that represents each position throughout the model.
- Attention (Definition 10.1) computes, at each position t, three vectors from the vector there: a query $\mathbf{q}_t$, a key $\mathbf{k}_t$ and a value $\mathbf{v}_t$. Position t gives position j a weight that grows with the score $\mathbf{q}_t \cdot \mathbf{k}_j$ (the weights are a softmax of the scores), and its output is the weighted average of the values. The causal mask (Definition 10.3) gives weight zero to positions $j > t$, so no position sees later ones.
- Multi-head attention (Definition 10.4) splits the query, key and value vectors into h consecutive pieces of length $d_h = d/h$, the heads, runs attention separately on each piece, and places the h outputs one after another. Here $h = 4$ and $d_h = 4$.
- An MLP (Definition 7.3) applies a matrix, then $\mathrm{relu}(x) = \max(x, 0)$ (Definition 7.2) to each component, then a second matrix. Here the intermediate hidden vector has $64 = 4d$ components.

The model reads at most 16 tokens, its context length (Definition 10.5), and has a single *layer* in the sense used for transformers: one attention block followed by one MLP block. (A layer of an MLP in Definition 7.1 is a smaller unit, one affine map and one activation function.) Its parameters are nine matrices, with no bias vectors and no normalisation gains:

- E (27×16), the token embedding (Definition 9.6): row x is the vector that represents token x, and E_x denotes that row written as a column, the embedding $\mathbf{u}_x$ of that definition. The rows of E, like those of W_U, are numbered 0 to 26 to match the tokens.
- P (16×16), the position embedding matrix, written E_{pos} in Definition 10.5: row t, written P_t, is the vector added at position t (written $\mathbf{p}_t$ in that definition; here $\mathbf{p}_t$ is the model's probability vector at position t).
- W_Q, W_K, W_V (each 16×16): row i of W_Q gives component i of a query as a linear combination of the 16 components of the vector at that position, and likewise for keys and values.

- W_O (16×16): row i gives component i of the attention output as a linear combination of the 16 components of the four head outputs, so it mixes the heads.
- W_1 (64×16) and W_2 (16×64): the two matrices of the MLP. Row i of W_1 gives hidden component i before the ReLU; row i of W_2 gives output component i from the 64 hidden components.
- W_U (27×16), the output matrix: row i, dotted with the final vector at a position, gives the logit of token i.

A vector $\mathbf{q} \in \mathbb{R}^{16}$ splits into four heads, $\mathbf{q}^{(i)} = (q_{4i-3}, \dots, q_{4i})$ for $i = 1, \dots, 4$.

Definition 11.4 (The model)**.** Let the parameters be matrices E, P, W_Q, W_K, W_V, W_O, W_1, W_2, W_U of the shapes listed above. For tokens $x_1, \dots, x_T$ in $\{0, \dots, 26\}$ with $T \le 16$, and each position $t = 1, \dots, T$, the model computes:

$$
\begin{aligned}
\mathbf{r}_t^{(0)} &= \operatorname{rms}(E_{x_t} + P_t) && && \text{embedding} \\
\mathbf{u}_t &= \operatorname{rms}(\mathbf{r}_t^{(0)}), \quad \mathbf{q}_t = W_Q \mathbf{u}_t, \quad \mathbf{k}_t = W_K \mathbf{u}_t, \quad \mathbf{v}_t = W_V \mathbf{u}_t && && \text{queries, keys, values} \\
\alpha_t^{(i)} &= \operatorname{softmax}\Big(\frac{\mathbf{q}_t^{(i)} \cdot \mathbf{k}_1^{(i)}}{2}, \dots, \frac{\mathbf{q}_t^{(i)} \cdot \mathbf{k}_t^{(i)}}{2} \Big) && && \text{head } i = 1, \dots, 4 \\
\mathbf{o}_t^{(i)} &= \sum_{j=1}^{t} \alpha_{tj}^{(i)} \, \mathbf{v}_j^{(i)}, \qquad \mathbf{o}_t = (\mathbf{o}_t^{(1)}, \dots, \mathbf{o}_t^{(4)}) && && \text{attention output} \\
\mathbf{r}_t^{(1)} &= \mathbf{r}_t^{(0)} + W_O \, \mathbf{o}_t && && \text{attention block} \\
\mathbf{r}_t^{(2)} &= \mathbf{r}_t^{(1)} + W_2 \operatorname{relu}(W_1 \operatorname{rms}(\mathbf{r}_t^{(1)})) && && \text{MLP block} \\
\mathbf{z}_t &= W_U \, \mathbf{r}_t^{(2)} && && \text{logits}
\end{aligned}
$$

with rms as in Definition 11.1 ($n = 16$, $\varepsilon = 10^{-5}$), $\alpha_{tj}^{(i)}$ component j of $\alpha_t^{(i)}$, and $\mathbf{o}_t \in \mathbb{R}^{16}$ the four head outputs placed one after another. The model's distribution for the next token is $\mathbf{p}_t = \operatorname{softmax}(\mathbf{z}_t)$.

Read from top to bottom, the equations say: represent token x_t at position t by the sum of two embedding rows and normalise it; let each head compare the query of position t with the keys of positions 1 to t and average their values with the resulting weights; add the mixed head outputs to the vector; add the output of the MLP; and multiply by W_U to get one logit per token. Only the attention line lets position t see other positions.

The divisor 2 in the attention scores is $\sqrt{d_h}$. When the components of a query and a key are independent with mean 0 and variance 1, their dot product of d_h terms has variance d_h, and dividing by $\sqrt{d_h}$ brings the variance of the score back to 1 (Theorem 10.1). The attention line is causal multi-head attention (Definition 10.4, Definition 10.3): head i uses rows $4i-3$ to $4i$ of W_Q, W_K and W_V as its own query, key and value matrices, the weights $\alpha_{tj}^{(i)}$ for $j > t$ are zero, and W_O mixes the concatenated head outputs. The source counts positions from 0; here they run from 1, so P_t is row t of P in this book's numbering.

Several details differ from GPT-2 (Radford et al. 2019), and each is visible in the equations. The vectors $\mathbf{r}_t^{(0)}$, $\mathbf{r}_t^{(1)}$, $\mathbf{r}_t^{(2)}$, the successive states of the vector that represents position t, form the *residual stream* at position t. Each block reads a normalised copy of the stream and adds its output back. That is the arrangement of GPT-2, which moved normalisation from after the residual sum (Vaswani et al. 2017) to the input of each block. GPT-2 also normalises the stream once more before the output matrix; this model does not, so $\mathbf{z}_t$ is a linear function of $\mathbf{r}_t^{(2)}$. Instead, this model normalises the embedding sum before it enters the stream, so the stream starts with root mean square close to 1. It replaces layer normalisation by RMS normalisation without gain, and GELU (a smooth function close to ReLU) by ReLU. It has no biases. The output matrix W_U is a separate parameter, not a copy of E.

The two blocks are residual blocks in the sense of Section 11.2. For the attention block the map F acts on the whole sequence $(\mathbf{r}_1^{(0)}, \dots, \mathbf{r}_T^{(0)}) \in \mathbb{R}^{16T}$, because position t attends to earlier positions, and Theorem 11.3 applies in $\mathbb{R}^{16T}$. The normalisation sits inside F, so the identity path from $\mathbf{r}_t^{(0)}$ to

$\mathbf{r}_t^{(2)}$ passes through no normalisation and no matrix. The adjoint of the logits reaches $\mathbf{r}_t^{(0)}$ as $W_U^\top \bar{\mathbf{z}}_t$ plus the corrections from the two blocks.

The vector $\mathbf{u}_t$ normalises a vector that is already normalised. In the forward pass this changes little: by Theorem 11.2 the mean square of $\mathbf{r}_t^{(0)}$ is $\mu/(\mu+\varepsilon)$, just below 1, and normalising it again multiplies it by a number close to 1 (Exercise 11.4). In the backward pass it does more: by Theorem 11.1 its Jacobian nearly removes the component of the adjoint along $\mathbf{r}_t^{(0)}$, while the identity path keeps it.

The equations translate directly into array code. The function below computes every position at once, as matrices whose row t holds the vectors for position t, and it implements the causal mask by setting the scores for $j > t$ to $-\infty$ before the softmax, so that their weights are $e^{-\infty} = 0$. As in Equation 10.2, `Q`, `K` and `V` hold the queries, keys and values, so in the code `V` is the matrix of values, not the vocabulary.

```
import numpy as np

def rms_rows(X, eps=1e-5):
    """RMS-normalise each row of X."""
    return X / np.sqrt(np.mean(X**2, axis=-1, keepdims=True) +
     ↪  eps)

def logits(theta, tokens):
    """Logits z_1, ..., z_T (as rows) for a list of token
     ↪  numbers."""
    T = len(tokens)
    R0 = rms_rows(theta["E"][tokens] + theta["P"][:T])
    U = rms_rows(R0)
    Q, K, V = U @ theta["WQ"].T, U @ theta["WK"].T, U @
↪   theta["WV"].T
    later = np.triu(np.ones((T, T), dtype=bool), k=1)   # entries
↪   with j > t
    O = np.zeros((T, 16))
    for j in range(4):
        c = slice(4 * j, 4 * j + 4)              # the columns of
↪   head j
```

```
        S = Q[:, c] @ K[:, c].T / 2.0  # scores q_t . k_j /
↪  sqrt(4)
        S[later] = -np.inf
        A = np.exp(S - S.max(axis=1, keepdims=True))
        A = A / A.sum(axis=1, keepdims=True)            # row t
↪  is alpha_t
        O[:, c] = A @ V[:, c]
    R1 = R0 + O @ theta["WO"].T
    R2 = R1 + np.maximum(rms_rows(R1) @ theta["W1"].T, 0.0) @
↪  theta["W2"].T
    return R2 @ theta["WU"].T
```

11.3.1. Counting the parameters

Theorem 11.4 (Parameter count)**.** *The model has exactly 4192 parameters, distributed as follows.*

Matrix	*Role*	*Shape*	*Entries*
E	*token embedding*	27×16	*432*
P	*position embedding*	16×16	*256*
W_Q, W_K, W_V	*queries, keys, values*	16×16 *each*	*768*
W_O	*attention output*	16×16	*256*
W_1	*MLP, first layer*	64×16	*1024*
W_2	*MLP, second layer*	16×64	*1024*
W_U	*output*	27×16	*432*
total			*4192*

Proof. The parameters are the entries of the nine matrices of Definition 11.4: the normalisations have no gain and there are no biases, so nothing else changes in training. An $m \times n$ matrix has mn entries, which gives the last column, and $432 + 256 + 768 + 256 + 1024 + 1024 + 432 = 4192$. □

The embeddings hold 688 parameters, the attention block 1024, the MLP block 2048 and the output matrix 432, so almost half the parameters sit in the MLP. The parameter vector θ of the model lies in $\mathbb{R}^{4192}$.

11.4. Training

Training adjusts the parameters so that the model's predictions match real names, and it needs a single number that measures the mismatch. For `emma` the model makes five predictions, one per position: $\diamond$ followed by `e`, then `e` by `m`, `m` by `m`, `m` by `a`, and `a` by $\diamond$. If it gave each correct token probability $1/2$, the average of $-\ln$ of these probabilities would be $\ln 2 \approx 0.693$; if it gave each probability $1/27$, as a model that knows nothing might, the average would be $\ln 27 \approx 3.296$. That average is the loss of Definition 9.5.

In general, a name $c_1 c_2 \cdots c_m$ becomes the token sequence $x_1 = \diamond$, $x_{i+1} = c_i$ for $i = 1, \ldots, m$, and $x_{m+2} = \diamond$, numbered as in Definition 9.3. The model predicts each token from those before it. With $T = \min(16, m + 1)$ predictions, the loss on the name is the average cross-entropy (Definition 8.3)

$$L(\theta) = \frac{1}{T} \sum_{t=1}^{T} -\ln p_{t,\, x_{t+1}}, \tag{11.3}$$

where $p_{t,\,x}$ is component x of $\mathbf{p}_t$ and $\theta \in \mathbb{R}^{4192}$ is the parameter vector of Section 11.3.1. The names in the training list have at most 15 letters (the source notes this), so $T = m + 1$ and the last prediction is always the closing boundary token: the model learns where names end as well as how they start. The minimum in T only matters for longer documents,

whose last targets the model never sees. By the chain rule of probability (Theorem 9.1), TL is minus the logarithm of the probability that the model assigns to the whole name, closing boundary included, and e^L is the perplexity of the model on it (Definition 9.5).

By Theorem 8.5, the adjoint of the logits at position t is $\bar{\mathbf{z}}_t = (\mathbf{p}_t - \mathbf{e}_{x_{t+1}})/T$, where $\mathbf{e}_x$ is the standard basis vector of token x. Reverse mode carries these 27-component vectors back through the equations of Definition 11.4 to all 4192 parameters at once (Theorem 4.1).

Training runs Adam (Definition 6.7). Adam keeps, for each parameter, two exponential moving averages (Definition 6.6): m of its recent gradients, with decay rate β_1, and v of their squares, with decay rate β_2. It corrects both for their bias towards the starting value 0 (Theorem 6.9), giving $\hat{m}$ and $\hat{v}$, and subtracts the learning rate times $\hat{m}/(\sqrt{\hat{v}}+\epsilon)$ from the parameter, so each parameter's step is scaled by the recent size of its own gradients. Here $\beta_1 = 0.85$, $\beta_2 = 0.99$ and $\epsilon = 10^{-8}$ (a different constant from the ε of RMS normalisation), and training runs for 1000 steps. Each step $k = 1, \dots, 1000$, numbered as in Definition 6.7, takes one name from a shuffled list and computes L and its gradient, and Adam then updates θ with learning rate

$$\eta_k = 0.01\Big(1 - \frac{k-1}{1000}\Big),$$

which falls linearly from 0.01 at the first step to 10^{-5} at the last, as in Section 6.6. (The source numbers the steps from 0.) There is no batching: each gradient comes from a single name. The list holds 32,033 names (Karpathy 2026a), so 1000 steps use about 3% of it, each name at most once. Adam keeps two numbers per parameter, so the optimiser state is $2 \times 4192 = 8384$ numbers.

The source computes each softmax after subtracting the largest logit, and it treats that maximum as a constant in the computational graph. This does not change the derivative: subtracting a constant from every component does not change the softmax at all (Theorem A.3), so the two computations define the same function of the logits.

11.4.1. The loss before training

Before training, the loss tells us nothing about names, but its value is a useful test of an implementation, and Example 8.5 estimates it. A model that gives each of the 27 tokens probability $1/27$ has loss $\ln 27 \approx 3.296$. The book's model starts close to that model, because every weight starts as an independent random sample with mean 0 and standard deviation $\sigma_w = 0.08$, so the logits are small. Small random logits raise the expected loss above $\ln 27$, and the second-order Taylor expansion of log-sum-exp in that example puts the excess at about $\frac{1}{2} \cdot \frac{26}{27} \cdot 0.08^2 \cdot 16 \approx 0.049$, for an expected loss of about 3.35. The expansion comes with no bound on its error. The theorem below replaces it by inequalities that hold for the model of Definition 11.4.

Making this precise for the model needs random variables that can take any real value. Definition 7.4 defines the normal distribution $N(\mu, s^2)$ through its density φ: a random variable w with this distribution has mean μ and variance s^2, and $\mathbb{E}[g(w)] = \int_{-\infty}^{\infty} g(t)\varphi(t)\,dt$ whenever $\int_{-\infty}^{\infty} |g(t)|\varphi(t)\,dt$ is finite. microgpt draws every weight from $N(0, 0.08^2)$, independently of the others. The proof also uses the rule that the expectation of a product of independent random variables is the product of their expectations, when these exist. Theorem 6.2 proves it for two random variables on a finite probability space, induction on the number of factors extends it to more (with Lemma 6.1), and as Section 7.5 notes it holds for random variables with densities too. The density gives one formula that the proof below uses: if w has distribution $N(0, \sigma^2)$ and c is a real number, then, completing the square,

$$\mathbb{E}[e^{cw}] = \int_{-\infty}^{\infty} \frac{e^{cw - w^2/(2\sigma^2)}}{\sqrt{2\pi}\,\sigma}\,dw = e^{c^2\sigma^2/2} \int_{-\infty}^{\infty} \frac{e^{-(w-c\sigma^2)^2/(2\sigma^2)}}{\sqrt{2\pi}\,\sigma}\,dw = e^{c^2\sigma^2/2}, \tag{11.4}$$

since the last integrand is the density of $N(c\sigma^2, \sigma^2)$.

Theorem 11.5 (Loss at initialisation)**.** *In the model of Definition 11.4, fix the tokens and every parameter except W_U, and let the entries of W_U*

be independent random variables with distribution $N(0, \sigma_w^2)$, $\sigma_w > 0$. *Then at each position* t, *with* $\mathbf{r} = \mathbf{r}_t^{(2)}$ *and* $\ell_t = -\ln p_{t,\,x_{t+1}}$,

$$\ln 27 \;\le\; \mathbb{E}[\ell_t] \;\le\; \ln 27 + \frac{\sigma_w^2 \|\mathbf{r}\|^2}{2}.$$

The same bounds hold for the expected loss $\mathbb{E}[L]$ *of Equation 11.3, with* $\|\mathbf{r}\|^2$ *replaced by the average of* $\|\mathbf{r}_t^{(2)}\|^2$ *over* t.

Proof. The vector $\mathbf{r}$ does not involve W_U, so it stays fixed. Write $\mathbf{w}_i$ for row i of W_U (as a column), so that $z_i = \mathbf{w}_i \cdot \mathbf{r}$, and let $y = x_{t+1}$. Then

$$\ell_t = \ln \sum_i e^{z_i} - z_y,$$

with the sum over the 27 tokens.

Lower bound. Let $\bar{z} = \frac{1}{27}\sum_i z_i$. The tangent line to the exponential at $\bar{z}$ lies below it: $e^z \ge e^{\bar{z}}(1 + z - \bar{z})$ for all z. Summing over the 27 values z_i gives $\sum_i e^{z_i} \ge 27 e^{\bar{z}}$, so $\ell_t \ge \ln 27 + \bar{z} - z_y$. Each z_i has expectation $\sum_j r_j\,\mathbb{E}[(\mathbf{w}_i)_j] = 0$, so taking expectations gives $\mathbb{E}[\ell_t] \ge \ln 27$.

Upper bound. The tangent line to the logarithm at $m > 0$ lies above it: $\ln Y \le \ln m + (Y - m)/m$ for $Y > 0$. Take $Y = \sum_i e^{z_i}$ and $m = \mathbb{E}[Y]$; taking expectations, $\mathbb{E}[\ln Y] \le \ln \mathbb{E}[Y]$. The entries of $\mathbf{w}_i$ are independent, so by Equation 11.4

$$\mathbb{E}[e^{z_i}] = \prod_{j=1}^{16} \mathbb{E}[e^{r_j (\mathbf{w}_i)_j}] = \prod_{j=1}^{16} e^{r_j^2 \sigma_w^2/2} = e^{\sigma_w^2 \|\mathbf{r}\|^2/2},$$

and $\mathbb{E}[Y] = 27 e^{\sigma_w^2 \|\mathbf{r}\|^2/2}$. With $\mathbb{E}[z_y] = 0$ this gives the upper bound. The statement for L follows by averaging over t, since expectation is linear. □

How large is $\|\mathbf{r}_t^{(2)}\|^2$ at initialisation? By Theorem 11.2, $\|\mathbf{r}_t^{(0)}\|^2 = 16\lambda^2$ is just below 16. The two blocks add $W_O \mathbf{o}_t$ and $W_2 \mathbf{g}_t$, with $\mathbf{g}_t$ the MLP's

hidden vector. Since W_O is independent of $\mathbf{r}_t^{(0)}$ and $\mathbf{o}_t$ and has mean zero, the cross term $\mathbb{E}[\mathbf{r}_t^{(0)} \cdot W_O \mathbf{o}_t]$ vanishes, and likewise for W_2, so

$$\mathbb{E}\|\mathbf{r}_t^{(2)}\|^2 = \mathbb{E}\|\mathbf{r}_t^{(0)}\|^2 + \mathbb{E}\|W_O\mathbf{o}_t\|^2 + \mathbb{E}\|W_2\mathbf{g}_t\|^2.$$

Over 400 random initialisations, averaged over the positions of the name "isabella", the three terms are about 16.0, 0.06 and 0.33, so $\|\mathbf{r}_t^{(2)}\|^2 \approx 16.4$ and the gap $\sigma_w^2\|\mathbf{r}\|^2/2$ in Theorem 11.5 is about 0.052. The expected initial loss therefore lies between 3.296 and about 3.35. A direct simulation, averaging the loss of Equation 11.3 over 400 random initialisations and a dozen names, gives 3.35, at the top of that range and in agreement with the estimate of Example 8.5; the standard deviation of a single document's loss is about 0.14. The microgpt post reports a loss of 3.37 at the first step, consistent with this, and about 2.37 after 1000 steps (Karpathy 2026a).

11.5. Sampling and temperature

After training, the model generates a name by sampling. It starts with $x_1 = \diamond$ at position 1, computes $\mathbf{z}_1$, draws x_2 at random from a distribution built from $\mathbf{z}_1$, feeds x_2 back in at position 2, and continues until it draws $\diamond$ or has drawn 16 tokens. The letters drawn before the boundary token form the name, so a sample has at most 16 letters. The distribution is not $\operatorname{softmax}(\mathbf{z}_t)$ itself but $\operatorname{softmax}(\mathbf{z}_t/T)$ for a *temperature* $T > 0$; microgpt uses $T = 0.5$. (In this section T is the temperature, not the length of a sequence.)

Dividing the logits by T multiplies all their differences by $1/T$. For $T < 1$ this exaggerates the differences and concentrates the probability on the likeliest tokens; for $T > 1$ it flattens the distribution.

Example 11.3. Let $\mathbf{z} = (2, 1, 0)$. At $T = 1$ the probabilities are $(0.665, 0.245, 0.090)$; at $T = 0.5$ they are $\operatorname{softmax}(4, 2, 0) = (0.867, 0.117, 0.016)$; at $T = 0.1$ the first is 0.99995; and at $T = 100$ they are $(0.3367, 0.3333, 0.3300)$, close to uniform.

As T falls the distribution approaches the standard basis vector $\mathbf{e}_1$, the one-hot vector (Section 8.6) that puts probability 1 on the token with the largest logit, and as T grows it approaches the uniform distribution. The limits are exact, and the only subtlety is a tie for the largest logit.

Theorem 11.6 (Temperature limits)**.** *Let $\mathbf{z} \in \mathbb{R}^n$ and, for $T > 0$, $\mathbf{p}(T) = \mathrm{softmax}(\mathbf{z}/T)$. Let $M = \max_i z_i$, let $A = \{i : z_i = M\}$ be the set of positions where the maximum is attained, and let k be the number of elements of A.*

(a) As $T \to \infty$, $\mathbf{p}(T) \to \frac{1}{n}\mathbf{1}$, the uniform distribution.

(b) As $T \to 0$ through positive values, $p_i(T) \to 1/k$ for $i \in A$ and $p_i(T) \to 0$ for $i \notin A$. In particular, if the maximum is attained at a single position i^*, then $\mathbf{p}(T) \to \mathbf{e}_{i^*}$.

Proof.

(a) For each i, $z_i/T \to 0$, so $e^{z_i/T} \to 1$ by continuity of the exponential. The denominator $\sum_j e^{z_j/T}$ tends to n, which is not 0, so $p_i(T) \to 1/n$.

(b) Multiplying the numerator and denominator of $p_i(T)$ by $e^{-M/T}$,

$$p_i(T) = \frac{e^{(z_i - M)/T}}{\sum_{j=1}^{n} e^{(z_j - M)/T}}.$$

For $j \in A$ the term $e^{(z_j - M)/T}$ equals 1 for every T. For $j \notin A$, $z_j - M < 0$, so $(z_j - M)/T \to -\infty$ as $T \to 0$ and $e^{(z_j - M)/T} \to 0$. So the denominator tends to $k \geq 1$, and the numerator tends to 1 if $i \in A$ and to 0 otherwise.

□

When several logits tie for the maximum, the limit at low temperature is uniform over the tied tokens: the temperature cannot break a tie. At $T = 1$ the model samples from its own distribution $\mathbf{p} = \mathrm{softmax}(\mathbf{z})$, and in general $\mathrm{softmax}(\mathbf{z}/T)$ is proportional to the vector of powers $p_i^{1/T}$ (Exercise 11.6). At $T = 0.5$ the model samples token i with probability proportional to p_i^2.

11.6. Caching keys and values

Generating a name token by token seems to require running the model on the whole prefix $x_1, \dots, x_t$ at every step t. For a name of 16 tokens that means processing $1 + 2 + \cdots + 16 = 136$ positions, although only 16 of them are new: at step 3, for example, the keys and values of positions 1 and 2 are the same ones already computed at steps 1 and 2. The repeated work is unnecessary. Everything in Definition 11.4 except attention works on one position at a time, and by Theorem 10.2 causal attention at position t needs only the query of position t and the keys and values of positions 1 to t. So a program can keep the key–value cache of Section 10.3, a list of the key and value vectors of the positions seen so far, and process each new position once.

Definition 11.5 (Cached evaluation)**.** Given tokens $x_1, \dots, x_T$ in $\{0, \dots, 26\}$ with $T \le 16$, the *cached evaluation* of the model starts with an empty list C and, for $t = 1, \dots, T$ in turn:

1. computes $\mathbf{r}_t^{(0)}$, $\mathbf{u}_t$, $\mathbf{q}_t$, $\mathbf{k}_t$ and $\mathbf{v}_t$ from x_t and t by the formulas of Definition 11.4, and appends $(\mathbf{k}_t, \mathbf{v}_t)$ to C;
2. computes $\hat{\mathbf{o}}_t$ by the attention formulas of Definition 11.4, using $\mathbf{q}_t$ and every pair in C;
3. computes $\hat{\mathbf{r}}_t^{(1)}$, $\hat{\mathbf{r}}_t^{(2)}$ and $\hat{\mathbf{z}}_t$ from $\mathbf{r}_t^{(0)}$ and $\hat{\mathbf{o}}_t$ by the remaining formulas.

Theorem 11.7 (Caching computes causal attention)**.** *Take the model of Definition 11.4 with any values of its parameters. For every $T \leq 16$ and tokens $x_1, \dots, x_T$, the cached evaluation of Definition 11.5 gives $\hat{\mathbf{z}}_t = \mathbf{z}_t$ for $t = 1, \dots, T$, where $\mathbf{z}_t$ are the logits of Definition 11.4 computed for the whole sequence, and $\mathbf{z}_t$ depends only on $x_1, \dots, x_t$.*

Proof. In Definition 11.4, $\mathbf{r}_t^{(0)}$, $\mathbf{u}_t$, $\mathbf{q}_t$, $\mathbf{k}_t$ and $\mathbf{v}_t$ are functions of x_t and t alone, and $\mathbf{r}_t^{(1)}$, $\mathbf{r}_t^{(2)}$ and $\mathbf{z}_t$ are functions of $\mathbf{r}_t^{(0)}$ and $\mathbf{o}_t$ alone. Only $\mathbf{o}_t$ involves other positions. The output $\mathbf{o}_t^{(i)}$ of head i is causal self-attention of width 4 with its own rows of W_Q, W_K and W_V (Definition 10.4), applied to $\mathbf{u}_1, \dots, \mathbf{u}_T$. By Theorem 10.2, applied to each head, $\mathbf{o}_t^{(i)}$ is the unmasked attention of $\mathbf{q}_t^{(i)}$ over the pairs $(\mathbf{k}_j^{(i)}, \mathbf{v}_j^{(i)})$ with $j \leq t$, and it depends only on $\mathbf{u}_1, \dots, \mathbf{u}_t$. Hence $\mathbf{z}_t$ is a function of $x_1, \dots, x_t$, which proves the second claim.

At step t of the cached evaluation, after the append, C holds exactly the t pairs $(\mathbf{k}_j, \mathbf{v}_j)$ for $j = 1, \dots, t$, in order, each computed from x_j and j by the same formula as in the whole-sequence computation and therefore equal to it. Step 1 computes the same $\mathbf{r}_t^{(0)}$ and $\mathbf{q}_t$. Step 2 computes, in each head, the attention of $\mathbf{q}_t^{(i)}$ over these t pairs, which is $\mathbf{o}_t^{(i)}$ by the first part of the proof, so $\hat{\mathbf{o}}_t = \mathbf{o}_t$. Step 3 then gives $\hat{\mathbf{z}}_t = \mathbf{z}_t$. □

With several layers the keys and values of layer l come from the residual stream after layer $l - 1$, which depends on earlier positions, so they are no longer functions of x_t and t alone. The same argument then runs by induction on t, and within each t by induction on the layer: once the cached and whole-sequence values agree at all positions before t, they agree at position t layer by layer.

The theorem matters for generation and for training. In generation, the token drawn at step t comes from the softmax of $\mathbf{z}_t$ divided by the temperature (Section 11.5), and $\mathbf{z}_t$ is the same whether computed with the cache or from scratch, so caching changes the cost and not the distribution of the

samples. In training, microgpt computes the loss with the cache too, one position at a time. The cached and whole-sequence computations define the same function $L(\theta)$, so they have the same gradient, and reverse mode may differentiate through the cache like any other computation. Since appending tokens never changes the logits at earlier positions, one pass over a name trains all of its predictions.

11.7. What the small model leaves out

The model computes the same kind of function as production language models, at a much smaller size, and several things are missing.

Tokenisation. The model reads single characters from a vocabulary of 27. GPT-2 reads text as bytes merged into frequent sequences by byte-pair encoding (Sennrich et al. 2016), with a vocabulary of 50,257 tokens (Radford et al. 2019). A token is then often a whole word or a piece of one, and the embedding and output matrices, each $|V| \times d$, become large.

Size. The model has one layer of width 16 and a context of 16 positions. The four GPT-2 models have 12 to 48 layers, widths 768 to 1600, a context of 1024 tokens, and between 117 million and 1542 million parameters (Radford et al. 2019). More layers means more residual blocks in the stack of Corollary 11.1.

Batching. Each training step here uses one name, processed one position at a time. Production training computes all positions of a sequence at once with the causal mask, as in the array code of Section 11.3, and averages the gradient over many sequences per step (GPT-2 used 512). Theorem 11.7 is what makes the all-at-once form give the same loss.

Other components. GPT-2 uses layer normalisation with gain and bias, biases in its linear maps, the GELU activation in place of ReLU, and a final normalisation before the output matrix. Production systems also

train in lower-precision arithmetic than the 64-bit floating point used here (Appendix A).

11.8. Notes and further reading

Karpathy (2026a) walks through the code of the model line by line; the code is Karpathy (2026b). Radford et al. (2019) describes GPT-2, and Vaswani et al. (2017) the original transformer, with normalisation after each residual sum. Xiong et al. (2020) compare the two placements of normalisation and show that placing it at the input of each block, as this model does, gives better-behaved gradients at initialisation. Layer normalisation is from Ba et al. (2016) and RMS normalisation from Zhang and Sennrich (2019); residual connections are from He et al. (2016). Byte-pair encoding for text comes from Sennrich et al. (2016).

11.9. Exercises

Exercise 11.1. Let $n = 2$, $\varepsilon = 1/2$ and $\mathbf{x} = (3, 4)$. Compute $\operatorname{rms}(\mathbf{x})$ and its Jacobian at $\mathbf{x}$, and check that the Jacobian has eigenvalues s and εs^3 as Theorem 11.1 says.

Exercise 11.2. Let $\mathbf{y} = \operatorname{rms}(\mathbf{x})$ and let $\bar{\mathbf{x}} = J^\top \bar{\mathbf{y}}$, with J the Jacobian of Theorem 11.1. Show that $\mathbf{x} \cdot \bar{\mathbf{x}} = \varepsilon s^3 \, (\mathbf{x} \cdot \bar{\mathbf{y}})$. Deduce that if ε were 0 (and $\mathbf{x} \neq \mathbf{0}$) the gradient with respect to $\mathbf{x}$ would always be perpendicular to $\mathbf{x}$, and explain this from Theorem 11.2.

Exercise 11.3. Let $C = I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$, the $n \times n$ centring matrix. Show that $C\mathbf{x} = \mathbf{x} - m(\mathbf{x})\mathbf{1}$ and $v(\mathbf{x}) = \mu(C\mathbf{x})$, so that layer normalisation with $\mathbf{g} = \mathbf{1}$ and $\mathbf{b} = \mathbf{0}$ is $\operatorname{rms} \circ C$. Find its Jacobian and show that it sends $\mathbf{1}$ to $\mathbf{0}$.

Exercise 11.4.

(a) Show that $\operatorname{rms}(\operatorname{rms}(\mathbf{x})) = c\,\operatorname{rms}(\mathbf{x})$ with $c = (\mu/(\mu+\varepsilon)+\varepsilon)^{-1/2}$, where $\mu = \mu(\mathbf{x})$.

(b) Show that $c = 1$ exactly when $\mu = 1 - \varepsilon$.

(c) For $\mu = 1$ and $\varepsilon = 10^{-5}$, show that $c^2 = (1+\varepsilon)/(1+\varepsilon+\varepsilon^2)$ and that $1 - c$ is about 5×10^{-11}.

Exercise 11.5. Generalise Theorem 11.4 to a model with vocabulary size $|V|$, context length n_c, width d, K layers (each with its own attention and MLP blocks, as in Definition 11.4) and MLP width $4d$. Show that the number of parameters is $2|V|d + n_c d + 12Kd^2$ and that it does not depend on the number of heads. How many parameters does the model have with $d = 32$ and $K = 2$, other sizes unchanged?

Exercise 11.6. Let $\mathbf{p} = \operatorname{softmax}(\mathbf{z})$. Show that for every $T > 0$,

$$\operatorname{softmax}(\mathbf{z}/T)_i = \frac{p_i^{1/T}}{\sum_j p_j^{1/T}}.$$

Find the sampling distribution at $T = 0.5$ when $\mathbf{p} = (0.5, 0.3, 0.2)$.

Exercise 11.7. Let $\mathbf{z} = (1, 3, 3, 0)$. Find the limits of $\operatorname{softmax}(\mathbf{z}/T)$ as $T \to 0$ and as $T \to \infty$, and its value at $T = 1/\ln 2$.

Exercise 11.8. Let $F\colon \mathbb{R}^n \to \mathbb{R}^n$ be differentiable at $\mathbf{a}$ with $\|J_F(\mathbf{a})\|_F < 1$. Show that the Jacobian $I + J_F(\mathbf{a})$ of the residual block is invertible. Give an example with $\|J_F(\mathbf{a})\|_F = 1$ where it is not.

Exercise 11.9. Extend Exercise 10.5 to the model of Definition 11.4. Count only the multiplications in attention scores ($\mathbf{q}_t^{(i)} \cdot \mathbf{k}_j^{(i)}$, d_h each) and in weighted sums of values (d_h per head for each j). Show that generating

n positions with the cache costs $d\,n(n+1)$ such multiplications, while recomputing all positions from scratch at every step costs $d\,n(n+1)(n+2)/3$. Evaluate both for $d = n = 16$.

References

Axler, Sheldon. 2024. *Linear Algebra Done Right*. 4th ed. Undergraduate Texts in Mathematics. Springer. https://doi.org/10.1007/978-3-031-41026-0.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. *Layer Normalization*. https://arxiv.org/abs/1607.06450.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. 2014. *Neural Machine Translation by Jointly Learning to Align and Translate*. https://arxiv.org/abs/1409.0473.

Baydin, Atılım Güneş, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. 2018. "Automatic Differentiation in Machine Learning: A Survey." *Journal of Machine Learning Research* 18 (153): 1–43. https://jmlr.org/papers/v18/17-468.html.

Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. "A Neural Probabilistic Language Model." *Journal of Machine Learning Research* 3: 1137–55. https://www.jmlr.org/papers/v3/bengio03a.html.

Billingsley, Patrick. 1995. *Probability and Measure*. 3rd ed. Wiley. https://openlibrary.org/isbn/0471007102.

Bolte, Jérôme, and Edouard Pauwels. 2020. *A Mathematical Model for Automatic Differentiation in Machine Learning*. https://arxiv.org/abs/

2006.02080.

Bottou, Léon, Frank E. Curtis, and Jorge Nocedal. 2018. "Optimization Methods for Large-Scale Machine Learning." *SIAM Review* 60 (2): 223–311. https://doi.org/10.1137/16M1080173.

Boyd, Stephen, and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press. https://doi.org/10.1017/CBO9780511804441.

Chen, Stanley F., and Joshua Goodman. 1999. "An Empirical Study of Smoothing Techniques for Language Modeling." *Computer Speech & Language* 13 (4): 359–93. https://doi.org/10.1006/csla.1999.0128.

Chen, Tianqi, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. *Training Deep Nets with Sublinear Memory Cost*. https://arxiv.org/abs/1604.06174.

Clifford, W. K. 1871. "Preliminary Sketch of Biquaternions." *Proceedings of the London Mathematical Society* s1-4 (1): 381–95. https://doi.org/10.1112/plms/s1-4.1.381.

Cover, Thomas M., and Joy A. Thomas. 2006. *Elements of Information Theory*. 2nd ed. Wiley. https://doi.org/10.1002/047174882X.

Cybenko, G. 1989. "Approximation by Superpositions of a Sigmoidal Function." *Mathematics of Control, Signals, and Systems* 2 (4): 303–14. https://doi.org/10.1007/BF02551274.

Deisenroth, Marc Peter, A. Aldo Faisal, and Cheng Soon Ong. 2020. *Mathematics for Machine Learning*. Cambridge University Press. https://doi.org/10.1017/9781108679930.

Glorot, Xavier, and Yoshua Bengio. 2010. "Understanding the Diffi-

culty of Training Deep Feedforward Neural Networks." *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Proceedings of machine learning research, vol. 9: 249–56. https://proceedings.mlr.press/v9/glorot10a.html.

Goh, Gabriel. 2017. "Why Momentum Really Works." *Distill* 2 (4). https://doi.org/10.23915/distill.00006.

Goldberg, David. 1991. "What Every Computer Scientist Should Know about Floating-Point Arithmetic." *ACM Computing Surveys* 23 (1): 5–48. https://doi.org/10.1145/103162.103163.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. https://www.deeplearningbook.org/.

Griewank, Andreas. 1992. "Achieving Logarithmic Growth of Temporal and Spatial Complexity in Reverse Automatic Differentiation." *Optimization Methods and Software* 1 (1): 35–54. https://doi.org/10.1080/10556789208805505.

Griewank, Andreas, and Andrea Walther. 2000. "Algorithm 799: Revolve: An Implementation of Checkpointing for the Reverse or Adjoint Mode of Computational Differentiation." *ACM Transactions on Mathematical Software* 26 (1): 19–45. https://doi.org/10.1145/347837.347846.

Griewank, Andreas, and Andrea Walther. 2008. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. 2nd ed. Society for Industrial; Applied Mathematics. https://doi.org/10.1137/1.9780898717761.

Grinstead, Charles M., and J. Laurie Snell. 1997. *Introduction to Probability*. 2nd ed. American Mathematical Society. https://math.dartmouth.edu/~prob/prob/prob.pdf.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. https://arxiv.org/abs/1502.01852.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. "Deep Residual Learning for Image Recognition." *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–78. https://doi.org/10.1109/CVPR.2016.90.

Higham, Nicholas J. 2002. *Accuracy and Stability of Numerical Algorithms*. 2nd ed. Society for Industrial; Applied Mathematics. https://doi.org/10.1137/1.9780898718027.

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. 1989. "Multilayer Feedforward Networks Are Universal Approximators." *Neural Networks* 2 (5): 359–66. https://doi.org/10.1016/0893-6080(89)90020-8.

IEEE. 2019. *IEEE Standard for Floating-Point Arithmetic*. IEEE Std 754-2019. IEEE. https://doi.org/10.1109/IEEESTD.2019.8766229.

Jurafsky, Daniel, and James H. Martin. 2026. *Speech and Language Processing*. 3rd, draft of 19 August 2026. https://web.stanford.edu/~jurafsky/slp3/.

Kahn, A. B. 1962. "Topological Sorting of Large Networks." *Communications of the ACM* 5 (11): 558–62. https://doi.org/10.1145/368996.369025.

Kakade, Sham, and Jason D. Lee. 2018. *Provably Correct Automatic Subdifferentiation for Qualified Programs*. https://arxiv.org/abs/1809.08530.

Karpathy, Andrej. 2020. *Micrograd: A Tiny Scalar-Valued Autograd Engine and a Neural Net Library on Top of It*. GitHub repository.

https://github.com/karpathy/micrograd.

Karpathy, Andrej. 2026a. "Microgpt." February 12. https://karpathy.github.io/2026/02/12/microgpt/.

Karpathy, Andrej. 2026b. *Microgpt.py: Train and Run a GPT in Pure, Dependency-Free Python*. GitHub gist. https://gist.github.com/karpathy/8627fe009c40f57531cb18360106ce95.

Kingma, Diederik P., and Jimmy Ba. 2014. *Adam: A Method for Stochastic Optimization*. https://arxiv.org/abs/1412.6980.

Leshno, Moshe, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. 1993. "Multilayer Feedforward Networks with a Nonpolynomial Activation Function Can Approximate Any Function." *Neural Networks* 6 (6): 861–67. https://doi.org/10.1016/S0893-6080(05)80131-5.

Lessard, Laurent, Benjamin Recht, and Andrew Packard. 2016. "Analysis and Design of Optimization Algorithms via Integral Quadratic Constraints." *SIAM Journal on Optimization* 26 (1): 57–95. https://doi.org/10.1137/15M1009597.

Linnainmaa, Seppo. 1976. "Taylor Expansion of the Accumulated Rounding Error." *BIT* 16 (2): 146–60. https://doi.org/10.1007/BF01931367.

MacKay, David J. C. 2003. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press. https://www.inference.org.uk/itila/.

Nesterov, Yurii. 2018. *Lectures on Convex Optimization*. 2nd ed. Vol. 137. Springer Optimization and Its Applications. Springer. https://doi.org/10.1007/978-3-319-91578-4.

Nocedal, Jorge, and Stephen J. Wright. 2006. *Numerical Optimization*. 2nd

ed. Springer Series in Operations Research and Financial Engineering. Springer. https://doi.org/10.1007/978-0-387-40065-5.

Polyak, B. T. 1964. "Some Methods of Speeding up the Convergence of Iteration Methods." *USSR Computational Mathematics and Mathematical Physics* 4 (5): 1–17. https://doi.org/10.1016/0041-5553(64)90137-5.

PyTorch Contributors. 2026. "Autograd Mechanics." PyTorch documentation. https://docs.pytorch.org/docs/stable/notes/autograd.html.

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. *Language Models Are Unsupervised Multitask Learners*. OpenAI. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Reddi, Sashank J., Satyen Kale, and Sanjiv Kumar. 2018. *On the Convergence of Adam and Beyond*. https://arxiv.org/abs/1904.09237.

Robbins, Herbert, and Sutton Monro. 1951. "A Stochastic Approximation Method." *The Annals of Mathematical Statistics* 22 (3): 400–407. https://doi.org/10.1214/aoms/1177729586.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. 1986. "Learning Representations by Back-Propagating Errors." *Nature* 323 (6088): 533–36. https://doi.org/10.1038/323533a0.

Sennrich, Rico, Barry Haddow, and Alexandra Birch. 2016. "Neural Machine Translation of Rare Words with Subword Units." *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1715–25. https://doi.org/10.18653/v1/P16-1162.

Shannon, C. E. 1948. "A Mathematical Theory of Communication." *Bell System Technical Journal* 27 (3): 379–423. https://doi.org/10.1002/j.

1538-7305.1948.tb01338.x.

Stoer, J., and R. Bulirsch. 2002. *Introduction to Numerical Analysis*. 3rd ed. Texts in Applied Mathematics. Springer. https://doi.org/10.1007/978-0-387-21738-3.

Tarjan, Robert. 1972. "Depth-First Search and Linear Graph Algorithms." *SIAM Journal on Computing* 1 (2): 146–60. https://doi.org/10.1137/0201010.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, et al. 2017. "Attention Is All You Need." *Advances in Neural Information Processing Systems 30*. https://arxiv.org/abs/1706.03762.

Wasserman, Larry. 2004. *All of Statistics: A Concise Course in Statistical Inference*. Springer Texts in Statistics. Springer. https://doi.org/10.1007/978-0-387-21736-9.

Wengert, R. E. 1964. "A Simple Automatic Derivative Evaluation Program." *Communications of the ACM* 7 (8): 463–64. https://doi.org/10.1145/355586.364791.

Xiong, Ruibin, Yunchang Yang, Di He, et al. 2020. "On Layer Normalization in the Transformer Architecture." *Proceedings of the 37th International Conference on Machine Learning*. https://arxiv.org/abs/2002.04745.

Zhang, Biao, and Rico Sennrich. 2019. "Root Mean Square Layer Normalization." *Advances in Neural Information Processing Systems 32*. https://arxiv.org/abs/1910.07467.

Zorich, Vladimir A. 2015. *Mathematical Analysis I*. 2nd ed. Universitext. Springer. https://doi.org/10.1007/978-3-662-48792-1.

Zorich, Vladimir A. 2016. *Mathematical Analysis II*. 2nd ed. Universitext. Springer. https://doi.org/10.1007/978-3-662-48993-2.

A. Floating-point arithmetic

Every number in this book's computations, from a gradient checked by finite differences to the 4192 parameters of the model in Chapter 11, lives in the computer as a floating-point number: a binary fraction with a fixed number of significant bits, scaled by a power of 2. Arithmetic on such numbers rounds after every operation. This appendix states the rounding model that arithmetic following the IEEE 754 standard satisfies, and uses it to explain two things the main text relies on: why finite differences need a carefully chosen step, and why programs compute the softmax, and the logarithm of a sum of exponentials, after subtracting the largest input.

A.1. Binary floating-point numbers

A computer has to store numbers as large as 10^{300} and as small as 10^{-300} in a fixed number of bits, with the same relative precision across that range. Scientific notation does this in base 10; floating point does it in base 2. For example,

$$5.75 = 4 + 1 + \tfrac{1}{2} + \tfrac{1}{4} = (101.11)_2 = (1.0111)_2 \times 2^2 = \Big(1 + \frac{7}{16}\Big)2^2 :$$

a *significand* between 1 and 2, written in binary, times a power of 2. Storing a number then means storing the exponent (here 2) and the binary digits after the point (here 0111), together with a sign. The format that this book's computations use keeps 52 such digits.

Definition A.1 (Binary64 numbers)**.** The finite *binary64* numbers (the "double precision" numbers of IEEE 754) are 0 and the numbers

$$\pm\Big(1+\frac{f}{2^{52}}\Big)2^{e} \quad (0 \le f < 2^{52},\ -1022 \le e \le 1023), \qquad \pm\frac{f}{2^{52}}\,2^{-1022} \quad (0 < f < 2^{52}),$$

with f and e integers. Those of the first kind are *normal*, and those of the second kind are *subnormal*. A number occupies 64 bits: one for the sign, 11 for the exponent (stored as $e + 1023$, with the values 0 and 2047 reserved for zero and subnormals, and for infinities and NaN, "not a number") and 52 for f.

For 5.75, $e = 2$ and $f = (7/16)2^{52} = 7 \times 2^{48}$. The significand $1 + f/2^{52}$ has 53 significant bits, the leading 1 being implicit. Between 2^e and 2^{e+1} the normal numbers are equally spaced, 2^{e-52} apart, so the spacing doubles from one power of 2 to the next and the *relative* spacing stays between 2^{-53} and 2^{-52}. The spacing just above 1 is $2^{-52} \approx 2.22 \times 10^{-16}$, often called *machine epsilon*.

The largest finite number is $(2 - 2^{-52})\,2^{1023} \approx 1.798 \times 10^{308}$, the smallest positive normal number is $2^{-1022} \approx 2.225 \times 10^{-308}$, and the smallest positive subnormal is $2^{-1074} \approx 4.94 \times 10^{-324}$. Results too large to represent *overflow* to $\pm\infty$, and results too small *underflow* to a subnormal or to 0. For the exponential function this happens at moderate arguments: e^x overflows for $x > \ln\big((2 - 2^{-52})2^{1023}\big) \approx 709.78$, and rounds to 0 for $x < \ln 2^{-1075} \approx -745.13$.

Subnormal numbers fill the gap between 0 and the smallest positive normal number 2^{-1022}. Without them, the next number above 0 would be 2^{-1022}, while the normal numbers just above 2^{-1022} are only 2^{-1074} apart; two such numbers could then differ by an amount that is not representable, and their computed difference would be 0 although they are not equal. The subnormal numbers are the multiples of 2^{-1074} below 2^{-1022}, the same spacing as the normal numbers in $[2^{-1022}, 2^{-1021})$, and with them the difference of two distinct binary64 numbers is never computed as 0 (Goldberg 1991). Lemma A.2 below relies on them.

Most real numbers are not binary64 numbers. Even 0.1 is not, because 1/10 is not a fraction with a power of 2 as denominator (Exercise A.2). The computer stores a real number x in range as $\mathrm{fl}(x)$, the binary64 number nearest to x (with ties going to the one whose integer f is even); this is *rounding to nearest*. The stored value of 0.1 is 0.1000000000000000055511 ..., wrong by about 5.55×10^{-18}, a relative error of 5.55×10^{-17}. Near 0.1 the binary64 numbers are 2^{-56} apart and near 1000 they are 2^{-43} apart, so the absolute error of rounding grows with $|x|$, but the relative error stays below one bound.

Theorem A.1 (Rounding error)**.** *Let $u = 2^{-53}$, the* unit roundoff*. If x is real and $2^{-1022} \le |x| \le (2 - 2^{-52})2^{1023}$, then*

$$\mathrm{fl}(x) = x(1 + \delta) \quad \textit{for some } |\delta| \le u.$$

Proof. It is enough to treat $x > 0$. Let e be the integer with $2^e \le x < 2^{e+1}$; then $-1022 \le e \le 1023$. The binary64 numbers $2^e(1 + f/2^{52})$, $f = 0, \dots, 2^{52}$, lie 2^{e-52} apart and run from 2^e to 2^{e+1} (the last of them is the first number of the next range, or, for $e = 1023$, lies beyond the largest finite number, which x does not exceed). So x lies between two consecutive binary64 numbers that are 2^{e-52} apart, and the nearer of them is within half that distance: $|\mathrm{fl}(x) - x| \le 2^{e-53} \le 2^{-53}x$. Put $\delta = (\mathrm{fl}(x) - x)/x$. □

IEEE 754 requires the basic operations $+$, $-$, $\times$, $\div$ and $\sqrt{\ }$ to be *correctly rounded*: the computed result is fl of the exact result (IEEE 2019). By Theorem A.1, for binary64 numbers x and y and any of these operations $\circ$,

$$\mathrm{fl}(x \circ y) = (x \circ y)(1 + \delta), \qquad |\delta| \le u, \tag{A.1}$$

as long as the exact result lies in the normal range. This *standard model* is the starting point of rounding-error analysis (Higham 2002, ch. 2; Stoer and Bulirsch 2002, sec. 1.2). The standard does not require correct rounding of exp and log, and library implementations of them typically guarantee an error of about one *unit in the last place* (the spacing between consecutive

binary64 numbers at the size of the result), a relative error of a small multiple of u.

A.2. Cancellation

Subtracting two nearby binary64 numbers introduces no new rounding error (Lemma A.2 below), but the two numbers usually carry errors from earlier steps, and the subtraction can make those errors large compared with the result.

Example A.1. For small x, $1 - \cos x \approx x^2/2$. Computed directly, the result is poor: the computed $\cos x$ is a number near 1 with an error of up to about u, and that error is not small compared with $x^2/2$. For $x = 10^{-7}$ the direct computation gives 4.996×10^{-15} against the true 5.000×10^{-15}, a relative error of 8×10^{-4}. For $x = 10^{-8}$, $\cos x$ rounds to exactly 1 and the computed $1 - \cos x$ is 0, against the true value 5×10^{-17}. Rewriting with the identity $1 - \cos x = 2\sin^2(x/2)$ removes the subtraction, and gives 5.000×10^{-15} and 5.000×10^{-17}, correct to the precision shown.

In the example, the computed $\cos x$ is the true value times $1 + \delta$ with $|\delta|$ of the order of u, and the subtraction itself is exact. The general situation is two numbers known only through approximations with relative errors at most u, as Theorem A.1 provides.

Lemma A.1 (Cancellation)**.** *Let a and b be real numbers with $a \neq b$, let $u = 2^{-53}$, and let $\hat{a} = a(1 + \delta_a)$ and $\hat{b} = b(1 + \delta_b)$ with $|\delta_a|, |\delta_b| \leq u$. Then*

$$\frac{|(\hat{a} - \hat{b}) - (a - b)|}{|a - b|} \leq u \, \frac{|a| + |b|}{|a - b|}.$$

Proof. $(\hat{a} - \hat{b}) - (a - b) = a\delta_a - b\delta_b$, whose absolute value is at most $u(|a| + |b|)$ by the triangle inequality. □

When a and b are close, the factor $(|a|+|b|)/|a-b|$ is large, and the relative error of the difference can be far larger than u, even though the subtraction itself adds no error. The digits that agree cancel, and what remains is mostly the earlier rounding error. This is *cancellation*. In Example A.1, $a = 1$ and $b = \cos x \approx 1 - x^2/2$, so the factor is about $4/x^2$, which is 4×10^{14} for $x = 10^{-7}$.

The same effect appears in a difference quotient, where $f(x+h)$ and $f(x)$ agree in more and more digits as h shrinks. One more fact lets the analysis below ignore the subtraction itself. The stored values of 0.3 and 0.2 are binary64 numbers within a factor 2 of each other, and their difference $0.09999999999999997779\ldots$ is computed exactly: it is itself a binary64 number. It differs from the stored value of 0.1 only because storing 0.3 and 0.2 rounded them.

Lemma A.2 (Sterbenz's lemma)**.** *If a and b are binary64 numbers with $b/2 \le a \le 2b$, then $a-b$ is a binary64 number, so floating-point subtraction computes it exactly.*

Proof. If $b = 0$ then $a = 0$. Otherwise both numbers are positive, and swapping them changes only the sign of the difference, so suppose $b \le a \le 2b$. Let e be the exponent of b (with $e = -1022$ if b is subnormal). Every binary64 number at least 2^e is an integer multiple of 2^{e-52}, and so are b and $a \ge b$. Hence $a - b = D\,2^{e-52}$ for an integer D with $0 \le D \le b/2^{e-52} < 2^{53}$, because $a - b \le b$. If $D\,2^{e-52} \ge 2^{-1022}$, let $e' \le e$ be its exponent; then $D\,2^{e-52} = (D\,2^{e-e'})\,2^{e'-52}$ with $2^{52} \le D\,2^{e-e'} < 2^{53}$, a normal number. Otherwise it is an integer multiple of 2^{-1074} below 2^{-1022}, a subnormal number or 0. □

A.3. The error of finite differences

The forward difference $(f(x+h) - f(x))/h$ estimates the derivative $f'(x)$, and the hidden checks in this book use such estimates to test gradients.

How small should h be? In exact arithmetic the error falls as h falls. In floating point it does not. For $f(y) = 1/y$ at $x = 3$, where $f'(3) = -1/9$, the computed forward difference with $h = 2^{-10}$ is wrong by 3.6×10^{-5}; with $h = 2^{-27}$ by 8.3×10^{-10}; and with $h = 2^{-40}$ by 2.7×10^{-5} again. The function values carry rounding errors, the difference cancels their leading digits, and the division by h magnifies what is left. The total error has two parts that pull in opposite directions, which Section 3.7 introduced: the truncation error, the error of the exact difference quotient, which comes from cutting off the Taylor series of f and shrinks with h; and the rounding error, which comes from the computed values of f and grows as h shrinks.

Theorem A.2 (Error of finite differences)**.** *Let $u = 2^{-53}$, $h > 0$, and let $M_0 > 0$ bound $|f|$ at the points where the computation evaluates f. Suppose the computed values $\hat{f}(y)$ satisfy $|\hat{f}(y) - f(y)| \le uM_0$ at those points, and that the subtraction and the division by h (or $2h$) are exact.*

(a) If f is twice differentiable on an open interval containing $[x, x+h]$ and $|f''| \le M_2$ on $[x, x+h]$, with $M_2 > 0$, then

$$\left|\frac{\hat{f}(x+h) - \hat{f}(x)}{h} - f'(x)\right| \le \frac{M_2 h}{2} + \frac{2uM_0}{h}.$$

The bound is smallest at $h^* = 2\sqrt{uM_0/M_2}$, where it equals $2\sqrt{uM_0M_2}$.

(b) If f is three times differentiable on an open interval containing $[x-h, x+h]$ and $|f'''| \le M_3$ there, with $M_3 > 0$, then

$$\left|\frac{\hat{f}(x+h) - \hat{f}(x-h)}{2h} - f'(x)\right| \le \frac{M_3 h^2}{6} + \frac{uM_0}{h}.$$

The bound is smallest at $h^* = (3uM_0/M_3)^{1/3}$, where it equals $M_3(h^*)^2/2$.

Proof.

(a) By Taylor's theorem with the Lagrange form of the remainder, $f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(\xi)$ for some ξ between x and $x+h$, so the exact difference quotient satisfies

$$\Big|\frac{f(x+h)-f(x)}{h} - f'(x)\Big| = \frac{h}{2}|f''(\xi)| \le \frac{M_2 h}{2}.$$

The computed quotient differs from the exact one by $((\hat{f}(x+h) - f(x+h)) - (\hat{f}(x) - f(x)))/h$, whose absolute value is at most $2uM_0/h$. The triangle inequality adds the two bounds.

The function $g(h) = M_2 h/2 + 2uM_0/h$ has $g'(h) = M_2/2 - 2uM_0/h^2$, which is negative for $h < h^*$ and positive for $h > h^*$, so h^* gives the minimum, and $g(h^*) = \sqrt{uM_0M_2} + \sqrt{uM_0M_2}$.

(b) Taylor's theorem to third order at $x+h$ and at $x-h$ gives

$$f(x \pm h) = f(x) \pm hf'(x) + \frac{h^2}{2}f''(x) \pm \frac{h^3}{6}f'''(\xi_\pm),$$

with ξ_+ between x and $x+h$ and ξ_- between $x-h$ and x. Subtracting, the terms in $f(x)$ and $f''(x)$ cancel:

$$\frac{f(x+h)-f(x-h)}{2h} - f'(x) = \frac{h^2}{12}(f'''(\xi_+) + f'''(\xi_-)),$$

of absolute value at most $M_3h^2/6$. The rounding errors contribute at most $2uM_0/(2h) = uM_0/h$. The function $g(h) = M_3h^2/6 + uM_0/h$ has $g'(h) = M_3h/3 - uM_0/h^2$, which changes sign from negative to positive at h^* with $(h^*)^3 = 3uM_0/M_3$. There $uM_0/h^* = M_3(h^*)^2/3$, so $g(h^*) = M_3(h^*)^2/6 + M_3(h^*)^2/3 = M_3(h^*)^2/2$.

□

So the best step for a forward difference is of order $\sqrt{u} \approx 10^{-8}$ and gives about 8 correct digits, and the best step for a central difference is of order

$u^{1/3} \approx 5 \times 10^{-6}$ and gives about 11. Taking h smaller than the best step makes the estimate worse, not better. This is why the hidden checks in this book use central differences with steps near 10^{-6} when they compare a gradient with finite differences, and why exact derivatives computed by automatic differentiation (Chapter 3, Chapter 4) are preferable to finite differences whenever they are available.

The hypotheses of Theorem A.2 hold exactly for $f(y) = 1/y$ at $x = 3$ with $h = 2^{-k}$: $3 \pm h$ is a binary64 number for $k \leq 51$; each value $1/(3 \pm h)$ is a single correctly rounded division, so its error is at most $u|f| \leq uM_0$; the two values differ by less than a factor of 2, so by Lemma A.2 their difference is exact; and dividing by a power of 2 is exact. Here $f'(3) = -1/9$. For the forward difference $M_0 = 1/3$ and $M_2 = 2/27$, since $|f''(y)| = 2/y^3$ on $[3, 3 + h]$; for the central difference $M_0 = 1/(3 - h)$ and $M_3 = 6/(3 - h)^4$, since $|f'''(y)| = 6/y^4$ on $[3 - h, 3 + h]$.

Figure A.1 shows the observed errors for $k = 1, \ldots, 50$ against the bounds. For large h the forward-difference error follows the truncation term $h/27$; for small h rounding dominates, and the error grows like $1/h$. The bound predicts a best step $h^* = 2\sqrt{u(1/3)/(2/27)} \approx 4.5 \times 10^{-8}$ with error at most 3.3×10^{-9}. The smallest observed error is 8.3×10^{-10}, at $h = 2^{-27} \approx 7.5 \times 10^{-9}$; the bound is a worst case, and the actual rounding errors partly cancel. For the central difference the predicted best step is about 1.1×10^{-5} with error at most 4.9×10^{-12}, and the smallest observed error is 1.6×10^{-12}, at $h = 2^{-18} \approx 3.8 \times 10^{-6}$.

A.4. Softmax and log-sum-exp

The softmax of $\mathbf{z} \in \mathbb{R}^n$ is the vector with components $\text{softmax}(\mathbf{z})_i = e^{z_i} / \sum_j e^{z_j}$ (Definition 8.5), so computing it exponentiates every input. In binary64, e^z overflows for z above about 709.78. The 16-bit format binary16 (IEEE 2019) has the same structure as binary64 with 10 fraction bits and 5 exponent bits; its largest finite number is $(2 - 2^{-10})2^{15} =$

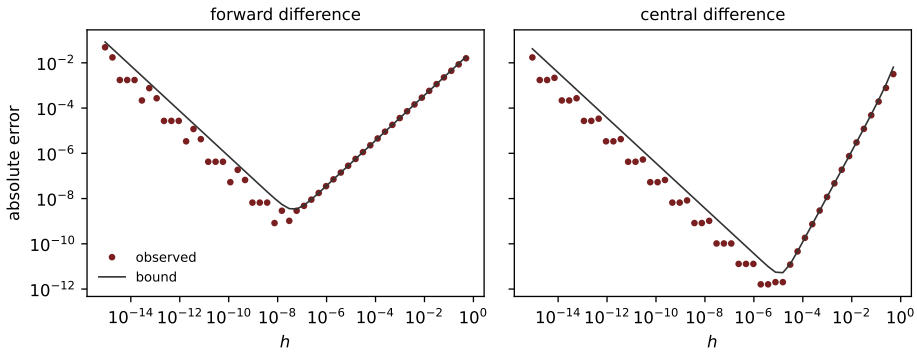

Figure A.1.: Error of the forward difference (left) and the central difference (right) for $f(y) = 1/y$ at $y = 3$, with $h = 2^{-k}$. Dots are observed errors (zero errors are not shown); the line is the bound of Theorem A.2.

65504, and e^z overflows there for z above about 11.09. The inputs to a softmax in a language model, its logits (Chapter 11), reach that size in practice. The remedy rests on an observation that a small case makes clear: $\mathrm{softmax}(1,2,3) \approx (0.090, 0.245, 0.665)$, and $\mathrm{softmax}(101,102,103)$ and $\mathrm{softmax}(-2,-1,0)$ are the same vector, because adding a constant to every input multiplies every exponential by the same factor, which cancels in the quotient.

Theorem A.3 (Softmax is unchanged by a shift)**.** *For every* $\mathbf{z} \in \mathbb{R}^n$ *and every real* c, $\mathrm{softmax}(\mathbf{z} + c\mathbf{1}) = \mathrm{softmax}(\mathbf{z})$. *Consequently, for every real* m,

$$\ln \sum_{j=1}^{n} e^{z_j} = m + \ln \sum_{j=1}^{n} e^{z_j - m}. \tag{A.2}$$

Proof. $e^{z_i + c} = e^c e^{z_i}$, so

$$\mathrm{softmax}(\mathbf{z} + c\mathbf{1})_i = \frac{e^c e^{z_i}}{\sum_j e^c e^{z_j}} = \frac{e^{z_i}}{\sum_j e^{z_j}} = \mathrm{softmax}(\mathbf{z})_i.$$

For Equation A.2, $\sum_j e^{z_j} = e^m \sum_j e^{z_j - m}$, and the logarithm of a product of positive numbers is the sum of their logarithms. □

The function $\mathrm{lse}(\mathbf{z}) = \ln \sum_j e^{z_j}$ is the log-sum-exp function of Definition 8.6. It appears in the cross-entropy loss, because $-\ln \mathrm{softmax}(\mathbf{z})_y = \mathrm{lse}(\mathbf{z}) - z_y$ (Chapter 8). In words, Equation A.2 says that log-sum-exp may be computed after subtracting any number m from every input, provided m is added back at the end.

Corollary A.1 (Stable evaluation)**.** *Let* $\mathbf{z} \in \mathbb{R}^n$*,* $m = \max_j z_j$ *and* $S = \sum_j e^{z_j - m}$*. Then every term* $e^{z_j - m}$ *lies in* $(0, 1]$*, at least one equals* 1*, and* $1 \le S \le n$*. Hence*

$$\mathrm{softmax}(\mathbf{z})_i = \frac{e^{z_i - m}}{S}, \qquad \mathrm{lse}(\mathbf{z}) = m + \ln S, \qquad -\ln \mathrm{softmax}(\mathbf{z})_y = m + \ln S - z_y,$$

and computed this way, no exponential exceeds 1*, and every denominator and every argument of the logarithm lies between* 1 *and* n*.*

Proof. $z_j - m \le 0$, with equality where the maximum is attained, so $0 < e^{z_j - m} \le 1$ with equality there, and summing n such terms gives $1 \le S \le n$. The three formulas are Theorem A.3 with $c = -m$ and Equation A.2. □

In floating point, a term $e^{z_j - m}$ may underflow to 0 when $z_j - m < -745$; since $S \ge 1$, that changes the computed probabilities by less than 2^{-1074}, far below the rounding error. The model of Chapter 11 computes its softmax in this way, and its hidden checks do the same.

Example A.2. Let $\mathbf{z} = (1000, 999, 998)$. Computed directly, e^{1000} overflows to ∞, so every quotient is ∞/∞ and every component of the result is NaN. Subtracting $m = 1000$ gives $(0, -1, -2)$, $S = 1 + e^{-1} + e^{-2} \approx 1.5032$, and

$$\mathrm{softmax}(\mathbf{z}) \approx (0.6652, 0.2447, 0.0900), \qquad \mathrm{lse}(\mathbf{z}) \approx 1000.4076.$$

For $\mathbf{z} = (-1000, -1001, -1002)$ the direct computation underflows instead: every e^{z_j} is 0, so the quotients are $0/0$ and the logarithm of the sum is $-\infty$. The shifted computation gives the same probabilities as before, since the shifted vector is again $(0, -1, -2)$, and $\operatorname{lse}(\mathbf{z}) \approx -999.5924$.

A.5. Notes and further reading

The IEEE 754 standard (IEEE 2019) defines the binary64 and binary16 formats and the rounding rules, and requires the basic operations to be correctly rounded. Goldberg (1991) is a widely read introduction to the consequences for programmers, with proofs. Higham (2002) develops rounding-error analysis in depth; its chapter 2 covers the standard model and its chapter 1 discusses cancellation. Stoer and Bulirsch (2002, ch. 1) treats floating-point numbers and error propagation in a first course on numerical analysis.

A.6. Exercises

Exercise A.1. What is the spacing between consecutive binary64 numbers near 1000? What is the largest power of 2 whose sum with 1000 rounds back to exactly 1000?

Exercise A.2. Show that 0.1 is not a binary64 number. Then explain why the computed sum $0.1 + 0.2$ is not equal to the computed 0.3, using Theorem A.1.

Exercise A.3. For large $x > 0$, the difference $\sqrt{x+1} - \sqrt{x}$ suffers from cancellation. Show that it equals $1/(\sqrt{x+1} + \sqrt{x})$, and use Lemma A.1 to explain why the second form is better. Compare the two in binary64 for $x = 10^{12}$.

Exercise A.4. Show that for $\mathbf{z} \in \mathbb{R}^n$, $\max_j z_j \le \operatorname{lse}(\mathbf{z}) \le \max_j z_j + \ln n$, with equality on the right exactly when all components are equal.

Exercise A.5. For $f = \sin$, with $M_0 = M_2 = 1$, find the step h^* and the error bound of Theorem A.2 (a) in binary64, and the corresponding step and bound for the central difference with $M_3 = 1$.

B. Solutions to the exercises

B.1. Chapter 1

Exercise 1.1. Holding the other variables fixed,

$$\frac{\partial f}{\partial x} = e^{yz}, \qquad \frac{\partial f}{\partial y} = xz\, e^{yz}, \qquad \frac{\partial f}{\partial z} = xy\, e^{yz}.$$

At $(1, 0, 2)$, $e^{yz} = 1$, so $\nabla f(1, 0, 2) = (1, 2, 0)$.

Exercise 1.2. Expanding with the properties of the dot product, $\|\mathbf{x}+\mathbf{y}\|^2 = \|\mathbf{x}\|^2 + 2\,\mathbf{x}\cdot\mathbf{y} + \|\mathbf{y}\|^2$ and $\|\mathbf{x}-\mathbf{y}\|^2 = \|\mathbf{x}\|^2 - 2\,\mathbf{x}\cdot\mathbf{y} + \|\mathbf{y}\|^2$. Adding, the dot products cancel and leave $2\|\mathbf{x}\|^2 + 2\|\mathbf{y}\|^2$.

Exercise 1.3. Both sides of the equation are non-negative, so it holds exactly when their squares are equal. By the expansion in the proof of Corollary 1.1, that means $\mathbf{x}\cdot\mathbf{y} = \|\mathbf{x}\|\,\|\mathbf{y}\|$.

If $\mathbf{x} = \mathbf{0}$ this holds. If $\mathbf{y} = c\mathbf{x}$ with $c \geq 0$, then $\mathbf{x}\cdot\mathbf{y} = c\|\mathbf{x}\|^2 = \|\mathbf{x}\|\,\|c\mathbf{x}\|$.

Conversely, suppose $\mathbf{x}\cdot\mathbf{y} = \|\mathbf{x}\|\,\|\mathbf{y}\|$ and $\mathbf{x} \neq \mathbf{0}$. Then equality holds in Theorem 1.1, so one vector is a multiple of the other. If $\mathbf{y} = \mathbf{0}$ then $\mathbf{y} = 0\mathbf{x}$. Otherwise $\mathbf{x} = t\mathbf{y}$ with $t \neq 0$, because $\mathbf{x} \neq \mathbf{0}$, and then $\mathbf{y} = (1/t)\mathbf{x}$. Either way $\mathbf{y} = c\mathbf{x}$ for some c, and $c\|\mathbf{x}\|^2 = \mathbf{x}\cdot\mathbf{y} = \|\mathbf{x}\|\,\|\mathbf{y}\| = |c|\,\|\mathbf{x}\|^2$, so $c = |c| \geq 0$.

Exercise 1.4. The Jacobian matrix and its determinant are

$$J_P(r, \theta) = \begin{pmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{pmatrix}, \qquad \det J_P = r\cos^2\theta + r\sin^2\theta = r.$$

Exercise 1.5. (a) $f(\mathbf{x}) = \mathbf{x}^\top I\mathbf{x}$ is a quadratic form, so by Example 1.8 it is differentiable everywhere with $\nabla f(\mathbf{x}) = (I + I^\top)\mathbf{x} = 2\mathbf{x}$.

(b) For $\mathbf{x} \neq \mathbf{0}$, write $g(\mathbf{x}) = (x_1^2 + \cdots + x_n^2)^{1/2}$. Differentiating with respect to x_j by the chain rule for one variable, $\partial g/\partial x_j = x_j/\|\mathbf{x}\|$. These partial derivatives are continuous on the open set of non-zero vectors, so by Theorem 1.6 and Theorem 1.5, g is differentiable there with $\nabla g(\mathbf{x}) = \mathbf{x}/\|\mathbf{x}\|$.

At $\mathbf{0}$, $(g(t\mathbf{e}_1) - g(\mathbf{0}))/t = |t|/t$, which is 1 for $t > 0$ and -1 for $t < 0$, so it has no limit as $t \to 0$. The partial derivative $\partial g/\partial x_1(\mathbf{0})$ does not exist, so by Theorem 1.5 g is not differentiable at $\mathbf{0}$.

Exercise 1.6. $\nabla f(1,1) = (2,6)$, so by Theorem 1.7 the directional derivative in the direction $(3,4)/5$ is $(2\cdot 3 + 6\cdot 4)/5 = 6$. A unit vector gives a zero directional derivative exactly when it is perpendicular to $(2,6)$, that is, a multiple of $(3,-1)$. The two such unit vectors are $\pm(3,-1)/\sqrt{10}$.

Exercise 1.7. (a) Since $x^2 \leq x^2 + y^2$, $|g(x,y)| \leq |y| \leq \|(x,y)\|$, which tends to 0 as $(x,y) \to (0,0)$.

(b) For $t \neq 0$ and $u_1^2 + u_2^2 = 1$, $g(t\mathbf{u}) = t^3 u_1^2 u_2/t^2 = t\,u_1^2 u_2$, so $(g(t\mathbf{u}) - g(\mathbf{0}))/t = u_1^2 u_2$ for every $t \neq 0$, and the limit is $u_1^2 u_2$.

(c) If g were differentiable at the origin, Theorem 1.7 would give $D_{\mathbf{u}} g(\mathbf{0}) = \nabla g(\mathbf{0}) \cdot \mathbf{u}$. Taking $\mathbf{u} = \mathbf{e}_1$ and $\mathbf{u} = \mathbf{e}_2$ in

(d) gives both partial derivatives as 0, so $\nabla g(\mathbf{0})$ would be $\mathbf{0}$ and every directional derivative would be 0. But $\mathbf{u} = (1,1)/\sqrt{2}$ gives $D_{\mathbf{u}} g(\mathbf{0}) = \frac{1}{2} \cdot \frac{1}{\sqrt{2}} \neq 0$.

Exercise 1.8. The rotation matrix is $R = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$, and by Example 1.6, $J_F = R$ everywhere. Next, $F(\mathbf{x}) - F(\mathbf{y}) = R\mathbf{v}$ with $\mathbf{v} = \mathbf{x} - \mathbf{y}$, and

$$\|R\mathbf{v}\|^2 = (v_1 \cos\theta - v_2 \sin\theta)^2 + (v_1 \sin\theta + v_2 \cos\theta)^2 = v_1^2 + v_2^2,$$

because the cross terms cancel and $\cos^2\theta + \sin^2\theta = 1$.

B.2. Chapter 2

Exercise 2.1. First, $f(1,2) = (2,3)$ and

$$J_f(x,y) = \begin{pmatrix} y & x \\ 1 & 1 \end{pmatrix}, \qquad J_g(u,v) = \begin{pmatrix} 2u & 1 \\ \cos u & 0 \end{pmatrix}.$$

By Theorem 2.1,

$$J_{g\circ f}(1,2) = J_g(2,3)\,J_f(1,2) = \begin{pmatrix} 4 & 1 \\ \cos 2 & 0 \end{pmatrix}\begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 9 & 5 \\ 2\cos 2 & \cos 2 \end{pmatrix}.$$

Directly, $(g\circ f)(x,y) = (x^2y^2 + x + y,\ \sin(xy))$, with partial derivatives $(2xy^2+1,\ 2x^2y+1)$ in the first row and $(y\cos(xy),\ x\cos(xy))$ in the second. At $(1,2)$ these are $(9,5)$ and $(2\cos 2, \cos 2)$.

Exercise 2.2. Fix $\mathbf{x}$ and let $\varphi(t) = f(t\mathbf{x})$ for $t > 0$. The curve $t \mapsto t\mathbf{x}$ has velocity $\mathbf{x}$, so by Corollary 2.1, $\varphi'(1) = \nabla f(\mathbf{x})\cdot\mathbf{x}$. By homogeneity $\varphi(t) = t^k f(\mathbf{x})$, whose derivative at $t=1$ is $kf(\mathbf{x})$. The two expressions for $\varphi'(1)$ are equal.

For $f(x,y) = x^2y - 4y^3$, $f(tx,ty) = t^3 f(x,y)$, so $k=3$, and

$$x\cdot 2xy + y(x^2 - 12y^2) = 3x^2y - 12y^3 = 3f(x,y).$$

Exercise 2.3. (a) Write $h = \sum_{i=1}^{m} f_i g_i$. By Lemma 2.1 each f_i and g_i is differentiable at $\mathbf{a}$, with $\nabla f_i(\mathbf{a})^\top$ equal to row i of $J_{\mathbf{f}}(\mathbf{a})$, and similarly for $\mathbf{g}$. By Theorem 2.3 each $f_i g_i$ is differentiable. A sum of functions differentiable at $\mathbf{a}$ is differentiable there, with derivative the sum of the derivatives, because the remainders add and the triangle inequality bounds the norm of their sum. So

$$J_h(\mathbf{a}) = \sum_i \big(g_i(\mathbf{a})\nabla f_i(\mathbf{a})^\top + f_i(\mathbf{a})\nabla g_i(\mathbf{a})^\top\big) = \mathbf{g}(\mathbf{a})^\top J_{\mathbf{f}}(\mathbf{a}) + \mathbf{f}(\mathbf{a})^\top J_{\mathbf{g}}(\mathbf{a}),$$

because a row vector times a matrix is the combination of the rows of the matrix with the entries of the vector as weights.

(b) Take $\mathbf{f} = \mathbf{g} = \gamma$, with $n = 1$. Then $h(t) = \|\gamma(t)\|^2 = 1$ for every t, so $0 = h'(t) = 2\gamma(t)^\top \gamma'(t) = 2\,\gamma(t) \cdot \gamma'(t)$.

Exercise 2.4. (a) The arrows are $v_{s-1} \to p_s$, $v_{s-1} \to q_s$, $p_s \to v_s$ and $q_s \to v_s$ for each s. The only arrows leaving v_{s-1} go to p_s and q_s, and the only arrow leaving either of these goes to v_s. So a path from $x = v_0$ to v_k passes through $v_1, v_2, \dots, v_{k-1}$ in turn, and between v_{s-1} and v_s it goes through exactly one of p_s and q_s. Each of the k stages offers two choices, so there are 2^k paths.

(b) The local partial derivatives are $2v_{s-1}$ on $v_{s-1} \to p_s$, $\cos v_{s-1}$ on $v_{s-1} \to q_s$, and 1 on the arrows into v_s. The product along a path is therefore $c_1 c_2 \cdots c_k$, where each c_s is $2v_{s-1}$ or $\cos v_{s-1}$ according to the choice made at stage s. By Theorem 2.2, dv_k/dx is the sum of these 2^k products over all choices, and expanding $\prod_{s=1}^{k} (2v_{s-1} + \cos v_{s-1})$ by the distributive law produces exactly that sum.

Exercise 2.5. (a) For $y \neq 0$, $f(0, y) = 0$ and

$$\frac{f(t,y) - f(0,y)}{t} = \frac{y(t^2 - y^2)}{t^2 + y^2} \to \frac{y(-y^2)}{y^2} = -y \quad \text{as } t \to 0.$$

For $y = 0$, $f(t, 0) = 0$ for every t, so $\partial f/\partial x(0,0) = 0 = -0$. In the same way, for $x \neq 0$, $(f(x,t) - f(x,0))/t = x(x^2 - t^2)/(x^2 + t^2) \to x$, and $\partial f/\partial y(0,0) = 0$.

(b) By (a), $t \mapsto \partial f/\partial x(0,t)$ is the function $-t$, whose derivative at 0 is -1; this is $\partial^2 f/\partial y\,\partial x(0,0)$. Likewise $t \mapsto \partial f/\partial y(t,0)$ is t, so $\partial^2 f/\partial x\,\partial y(0,0) = 1$. The hypothesis that fails is the continuity of the mixed partial derivatives at the origin. Away from the origin both equal $(x^2 - y^2)(x^4 + 10x^2y^2 + y^4)/(x^2 + y^2)^3$, which is 1 on the x-axis and -1 on the y-axis, so it has no limit at the origin.

Exercise 2.6. (a) The map $\mathbf{x} \mapsto z = \mathbf{w} \cdot \mathbf{x} + b$ is affine with Jacobian $\mathbf{w}^\top$ (Example 1.6), so Theorem 2.1 gives $J_f(\mathbf{x}) = \sigma'(z)\mathbf{w}^\top$, that is, $\partial f/\partial x_j = \sigma'(z)w_j$. This is again σ' composed with the same affine map, times the constant w_j, so $\partial^2 f/\partial x_i\,\partial x_j = \sigma''(z)w_i w_j$, which is entry (i,j) of $\sigma''(z)\mathbf{w}\mathbf{w}^\top$.

(b) Suppose $\mathbf{w}\mathbf{w}^\top\mathbf{v} = \lambda\mathbf{v}$ with $\mathbf{v} \neq \mathbf{0}$. The left side is $(\mathbf{w}\cdot\mathbf{v})\mathbf{w}$. If $\mathbf{w}\cdot\mathbf{v} = 0$ then $\lambda = 0$; otherwise $\mathbf{v}$ is a multiple of $\mathbf{w}$, and $\mathbf{w}\mathbf{w}^\top\mathbf{w} = \|\mathbf{w}\|^2\mathbf{w}$ gives $\lambda = \|\mathbf{w}\|^2$. Both values occur when $n \geq 2$: $\mathbf{w}$ is an eigenvector for $\|\mathbf{w}\|^2$, and the single equation $\mathbf{w}\cdot\mathbf{v} = 0$ in $n \geq 2$ unknowns has a non-zero solution, which is an eigenvector for 0. So the Hessian has eigenvalues $\sigma''(z)\|\mathbf{w}\|^2$ and 0. The neuron's output does not change at all along directions perpendicular to $\mathbf{w}$, and the zero eigenvalue records that.

Exercise 2.7. The gradient is $\nabla f = (4x^3 - 4y,\ 4y^3 - 4x)$, so a critical point has $y = x^3$ and $x = y^3$. Then $x = x^9$, so $x(x^8 - 1) = 0$ and x is 0, 1 or -1, giving the critical points $(0,0)$, $(1,1)$ and $(-1,-1)$. The Hessian is

$$\nabla^2 f(x,y) = \begin{pmatrix} 12x^2 & -4 \\ -4 & 12y^2 \end{pmatrix}.$$

At the origin $f_{xx}f_{yy} - f_{xy}^2 = -16 < 0$, so by Corollary 2.5 the origin is a saddle point. At $(1,1)$ and at $(-1,-1)$ the Hessian is $\begin{pmatrix} 12 & -4 \\ -4 & 12 \end{pmatrix}$, with $f_{xx}f_{yy} - f_{xy}^2 = 128 > 0$ and $f_{xx} = 12 > 0$ (its eigenvalues are 8 and 16), so both are strict local minima, where $f = -2$.

Exercise 2.8. The gradients are $\nabla f = (2x, 4y^3)$ and $\nabla g = (2x, -4y^3)$, both $\mathbf{0}$ at the origin. The Hessians are $\mathrm{diag}(2, 12y^2)$ and $\mathrm{diag}(2, -12y^2)$, both equal to $\mathrm{diag}(2,0)$ at the origin. For $(x,y) \neq (0,0)$, $f(x,y) = x^2 + y^4 > 0 = f(0,0)$, so the origin is a strict local minimum of f. For $t \neq 0$, $g(t,0) = t^2 > 0$ and $g(0,t) = -t^4 < 0$, so every disc around the origin contains points where g is above and below $g(0,0) = 0$, and the origin is a saddle point of g. The eigenvalues of the common Hessian are 2 and 0.

The eigenvalue 0 is neither positive nor negative, which rules out parts (a) and (b) of Theorem 2.10, and no eigenvalue is negative, which rules out part (c). The quadratic term $\frac{1}{2}\mathbf{h}^\top \nabla^2 f(\mathbf{0})\mathbf{h} = h_1^2$ vanishes along the y-axis, and there the fourth-order terms decide.

B.3. Chapter 3

Exercise 3.1. One trace is

variable	operation	value at $(0,1,2)$
v_1	x_1	0
v_2	x_2	1
v_3	x_3	2
v_4	$v_1 + v_2$	1
v_5	$v_2 + v_3$	3
v_6	$v_4 v_5$	3
v_7	$\exp v_1$	1
v_8	$v_6 v_7$	3

so $f(0,1,2) = 3$. The graph has the eight nodes and the edges $1 \to 4$, $2 \to 4$, $2 \to 5$, $3 \to 5$, $4 \to 6$, $5 \to 6$, $1 \to 7$, $6 \to 8$ and $7 \to 8$. The input x_1 has two children, v_4 and v_7, and so does x_2.

Exercise 3.2. With $\dot v_1 = 1$, $\dot v_2 = -1$, $\dot v_3 = 2$, the tangent rules give

$$\dot v_4 = \dot v_1 + \dot v_2 = 0, \quad \dot v_5 = \dot v_2 + \dot v_3 = 1, \quad \dot v_6 = v_5 \dot v_4 + v_4 \dot v_5 = 1,$$
$$\dot v_7 = e^{v_1}\dot v_1 = 1, \quad \dot v_8 = v_7 \dot v_6 + v_6 \dot v_7 = 1 + 3 = 4.$$

From the formula, by the product rule, $\partial f/\partial x_1 = (x_2 + x_3)e^{x_1} + (x_1 + x_2)(x_2 + x_3)e^{x_1}$, $\partial f/\partial x_2 = (x_1 + 2x_2 + x_3)e^{x_1}$ and $\partial f/\partial x_3 = (x_1 + x_2)e^{x_1}$, so $\nabla f(0,1,2) = (6,4,1)$ and $\nabla f(0,1,2)\cdot \mathbf{u} = 6 - 4 + 2 = 4$, as Theorem 3.2 says.

Exercise 3.3. The inputs 1 and 2 take the first two places, in either of 2 orders. Node 7 has parents 4 and 6, and the other two nodes, 3 and 5, must precede 4 and 6 respectively, so 7 comes last. The remaining four places hold $3, 4, 5, 6$. Their parents other than 3 and 5 are inputs, so the only conditions are that 3 comes before 4 and 5 before 6. An arrangement is fixed by choosing which 2 of the 4 places hold 3 and 4 (in that order), with 5 and 6 in the other two, so there are $\binom{4}{2} = 6$ arrangements, and $2 \times 6 = 12$ orders in all.

Exercise 3.4. (a) The inverse of $c + d\varepsilon$ is $1/c - (d/c^2)\varepsilon$, so

$$\frac{a + b\varepsilon}{c + d\varepsilon} = (a + b\varepsilon)\Big(\frac{1}{c} - \frac{d}{c^2}\varepsilon\Big) = \frac{a}{c} + \Big(\frac{b}{c} - \frac{ad}{c^2}\Big)\varepsilon = \frac{a}{c} + \frac{bc - ad}{c^2}\,\varepsilon.$$

This is also what Equation 3.2 gives for the operation $(t_1, t_2) \mapsto t_1/t_2$, whose partial derivatives are $1/t_2$ and $-t_1/t_2^2$.

(b) By Theorem 3.3 (a), $p(a + b\varepsilon) = p(a) + p'(a)b\varepsilon$ and $q(a + b\varepsilon) = q(a) + q'(a)b\varepsilon$, with $q(a) \neq 0$. By (a),

$$\frac{p(a + b\varepsilon)}{q(a + b\varepsilon)} = \frac{p(a)}{q(a)} + \frac{p'(a)q(a) - p(a)q'(a)}{q(a)^2}\, b\,\varepsilon = r(a) + r'(a)\, b\,\varepsilon$$

by the quotient rule, which is Equation 3.2 for $g = r$.

Exercise 3.5. The derivative of $\sqrt{t}$ is $1/(2\sqrt{t})$, so $\sqrt{a + b\varepsilon} = \sqrt{a} + \dfrac{b}{2\sqrt{a}}\,\varepsilon$. Squaring with Definition 3.5,

$$\Big(\sqrt{a} + \frac{b}{2\sqrt{a}}\varepsilon\Big)^2 = a + 2\sqrt{a} \cdot \frac{b}{2\sqrt{a}}\,\varepsilon = a + b\varepsilon.$$

In particular $\sqrt{4 + \varepsilon} = 2 + \frac{1}{4}\varepsilon$.

Exercise 3.6. Multiplication in $\mathbb{R}[\varepsilon]_3$ is multiplication of polynomials in ε followed by dropping ε^3 and higher powers, so it is associative and commutative, and the binomial theorem holds. For $k \geq 2$,

$$(a + \varepsilon)^k = \sum_{j=0}^{k} \binom{k}{j} a^{k-j}\varepsilon^j = a^k + ka^{k-1}\varepsilon + \binom{k}{2} a^{k-2}\varepsilon^2,$$

because every term with $j \geq 3$ contains $\varepsilon^3 = 0$. Since $\frac{d}{dt}t^k = kt^{k-1}$ and $\frac{d^2}{dt^2}t^k = k(k-1)t^{k-2} = 2\binom{k}{2}t^{k-2}$, this is $t^k + (t^k)'\varepsilon + \frac{1}{2}(t^k)''\varepsilon^2$ at $t = a$. The same formula holds for $k = 0$ and $k = 1$, where the derivatives that appear are 0. Multiplying by the coefficients of p and adding gives the result. For $p(t) = t^4 - 3t^2$,

$$p(1+\varepsilon) = (1 + 4\varepsilon + 6\varepsilon^2) - 3(1 + 2\varepsilon + \varepsilon^2) = -2 - 2\varepsilon + 3\varepsilon^2,$$

so $\frac{1}{2}p''(1) = 3$ and $p''(1) = 6$. Directly, $p''(t) = 12t^2 - 6$, which is 6 at $t = 1$.

Exercise 3.7. (a) The formula for s_k contains s_{k-1} twice and one more x, so its count L_k satisfies $L_k = 2L_{k-1} + 1$ with $L_0 = 1$. Then $L_k + 1 = 2(L_{k-1} + 1)$, so $L_k + 1 = 2^{k+1}$ and $L_k = 2^{k+1} - 1$.

(b) The formula for s_k' contains s_{k-1}' twice and s_{k-1} twice, so $D_k = 2D_{k-1} + 2L_{k-1} = 2D_{k-1} + 2^{k+1} - 2$, with $D_0 = 0$. Dividing by 2^k, $D_k/2^k = D_{k-1}/2^{k-1} + 2 - 2^{1-k}$, and summing from 1 to k,

$$\frac{D_k}{2^k} = 2k - \sum_{i=1}^{k} 2^{1-i} = 2k - 2 + 2^{1-k}, \qquad D_k = (k-1)\,2^{k+1} + 2.$$

For example $s_1' = 1 \cdot x + x \cdot 1 + 1$ contains x twice, and $D_1 = 0 + 2 = 2$.

(c) Each step is two lines, $w = v^2$ and $v_{\text{new}} = w + x$, so the trace has $2k$ operations, and forward mode adds one tangent line for each, $2k$ in all.

Exercise 3.8. By Taylor's theorem with the Lagrange remainder, there are ξ_+ in $(a, a+h)$ and ξ_- in $(a-h, a)$ with

$$f(a \pm h) = f(a) \pm f'(a)h + \tfrac{1}{2}f''(a)h^2 \pm \tfrac{1}{6}f'''(\xi_\pm)h^3.$$

Subtracting, the $f(a)$ and $f''(a)$ terms cancel: $f(a+h) - f(a-h) = 2f'(a)h + \frac{1}{6}(f'''(\xi_+) + f'''(\xi_-))h^3$. Dividing by $2h$,

$$\left|\frac{f(a+h) - f(a-h)}{2h} - f'(a)\right| = \frac{|f'''(\xi_+) + f'''(\xi_-)|}{12}\,h^2 \leq \frac{2M}{12}h^2 = \frac{Mh^2}{6}.$$

B.4. Chapter 4

Exercise 4.1. Write $c = \cos 1$. The accumulators start at 0, except $\bar{v}_7 = 1$, and the sweep processes the lines from 7 down to 3:

line	operation	contributions
7	$v_4 + v_6$	$\bar{v}_4$ += 1, $\bar{v}_6$ += 1
6	$\sin v_5$	$\bar{v}_5$ += $\bar{v}_6 \cos v_5 = c$
5	$v_1 v_2$	$\bar{v}_1$ += $\bar{v}_5 v_2 = 0.5c$,
		$\bar{v}_2$ += $\bar{v}_5 v_1 = 2c$
4	$v_3 v_2$	$\bar{v}_3$ += $\bar{v}_4 v_2 = 0.5$,
		$\bar{v}_2$ += $\bar{v}_4 v_3 = 4$
3	v_1^2	$\bar{v}_1$ += $\bar{v}_3 \cdot 2v_1 = 2$

So $\bar{v}_7 = \bar{v}_6 = \bar{v}_4 = 1$, $\bar{v}_5 = c$, $\bar{v}_3 = 0.5$, $\bar{v}_2 = 4 + 2c$ and $\bar{v}_1 = 2 + 0.5c$. The last two are the components of $\nabla f(2, 0.5)$ found by two forward passes in Example 3.1; the sweep found both at once.

Exercise 4.2. (a) The sweep starts with `bar[1]` = 0 and `bar[2]` = 1. Line 2 has the pairs $(1, x)$ and $(1, x)$, so the inner loop runs twice, and each time adds `bar[2] * x` = x to `bar[1]`, which started at 0. Line 1 is an input and has no pairs. So `bar[1]` is $2x$.

(b) With `=`, each pass of the inner loop sets `bar[1]` to x, so the result is x, half the true derivative. In Example 4.1, the variable a receives two contributions, 1 from line 4, $v_4 = v_3 + v_1$, and 3 from line 3, $v_3 = v_1 v_2$. The sweep processes line 4 before line 3, so with `=` the second assignment wins and $\partial L/\partial a$ comes out as 3 instead of 4.

Exercise 4.3. (a) The tape is

line	operation	value	pairs (i, d_{ki})
v_1	a	2	
v_2	b	3	
v_3	$v_1 v_2$	6	$(1,3)$, $(2,2)$
v_4	$\exp v_2$	$e^3 \approx 20.0855$	$(2, e^3)$
v_5	$v_3 + v_1$	8	$(3,1)$, $(1,1)$

The list `bar` starts as $(0,0,0,0,1)$ for lines 1 to 5. Line 5 adds 1 to the entries of lines 3 and 1, giving $(1,0,1,0,1)$. Line 4 is processed, but `bar[4]` is 0, so it adds $0 \cdot e^3 = 0$ to `bar[2]`. Line 3 adds $1 \cdot 3$ to `bar[1]` and $1 \cdot 2$ to `bar[2]`, giving $(4,2,1,0,1)$. The gradient is $(4,2)$, as in Example 4.1, and line 4 has adjoint 0 because L does not depend on it.

(b) Now `bar` starts as $(0,0,1,0,0)$. Lines 5 and 4 have adjoint 0, and the sweep processes both without changing anything. Line 3 then adds 3 to `bar[1]` and 2 to `bar[2]`, so the sweep returns $(3,2,1,0,0)$ and the gradient $(3,2)$ of ab. This is the first row of $J_f(2,3)$ in Section 4.5. Lines 4 and 5, on which u does not depend, cost a pass of the loop each and contribute nothing.

(c) From L the search visits the nodes of L, u, a and b, and lists them as a, b, u, L. From u it visits u, a and b. It never reaches the node of e^b, since no path leads from it to either output, so the sweep over the depth-first order skips it.

Exercise 4.4. The parents are (v_1) for v_3, (v_3, v_2) for v_4, (v_1, v_2) for v_5, (v_5) for v_6 and (v_4, v_6) for v_7. The visit to v_7 visits v_4, which visits v_3, which visits v_1: the list becomes v_1, v_3. The visit to v_4 then visits v_2 and finishes: v_1, v_3, v_2, v_4. The visit to v_7 then visits v_6, which visits v_5, whose parents are both marked, so v_5, v_6 and finally v_7 are appended. The result is

$$v_1,\ v_3,\ v_2,\ v_4,\ v_5,\ v_6,\ v_7.$$

Every edge, $1 \to 3$, $1 \to 5$, $2 \to 4$, $2 \to 5$, $3 \to 4$, $5 \to 6$, $4 \to 7$ and $6 \to 7$, goes from an earlier to a later entry, so this is a topological order. It does not put the inputs first, and the sweep does not need it to.

Exercise 4.5. The Jacobian and the product are

$$J_f(x,y) = \begin{pmatrix} y & x \\ 1 & 1 \\ 2x & 0 \end{pmatrix}, \qquad J_f(1,2) = \begin{pmatrix} 2 & 1 \\ 1 & 1 \\ 2 & 0 \end{pmatrix}, \qquad \mathbf{w}^\top J_f(1,2) = (2{-}1{+}4,\ 1{-}1{+}0) = (5,0).$$

For the sweep, take the trace $v_1 = x$, $v_2 = y$, $v_3 = v_1 v_2$, $v_4 = v_1 + v_2$, $v_5 = v_1^2$, with outputs v_3, v_4, v_5, and start the accumulators of the outputs at $1, -1, 2$. Line 5 adds $2 \cdot 2v_1 = 4$ to $\bar{v}_1$. Line 4 adds -1 to $\bar{v}_1$ and to $\bar{v}_2$. Line 3 adds $1 \cdot v_2 = 2$ to $\bar{v}_1$ and $1 \cdot v_1 = 1$ to $\bar{v}_2$. So $\bar{v}_1 = 4 - 1 + 2 = 5$ and $\bar{v}_2 = -1 + 1 = 0$, as Theorem 4.4 says.

Exercise 4.6. J_1 is $p \times n$, J_2 is $p \times p$ and J_3 is $1 \times p$. Forming $J_2 J_1$ takes $p \cdot p \cdot n$ multiplications and then $J_3(J_2 J_1)$ takes $1 \cdot p \cdot n$, so $J_3(J_2 J_1)$ costs $p^2 n + pn$. Forming $J_3 J_2$ takes p^2 and then $(J_3 J_2) J_1$ takes pn, so $(J_3 J_2) J_1$ costs $p^2 + pn$. For $n = p = 1000$ these are 1.001×10^9 and 2×10^6, a factor of about 500. The order $J_3(J_2 J_1)$ starts at the inputs and pushes the n columns of J_1 through the later Jacobians, as n forward passes do. The order $(J_3 J_2) J_1$ starts at the output and pulls the single row J_3 back through the earlier Jacobians, as one reverse sweep does.

Exercise 4.7. (a) Write $\bar{p}_k$ for the adjoint of p_k and $\bar{x}_k$ for that of x_k. The sweep starts with $\bar{p}_n = 1$ and processes $p_k = p_{k-1} x_k$ for $k = n, \dots, 3$ by

$$\bar{p}_{k-1} \mathrel{+}= \bar{p}_k \, x_k, \qquad \bar{x}_k \mathrel{+}= \bar{p}_k \, p_{k-1},$$

and finally $p_2 = x_1 x_2$ by $\bar{x}_1 \mathrel{+}= \bar{p}_2 x_2$, $\bar{x}_2 \mathrel{+}= \bar{p}_2 x_1$. Each of the $n-1$ lines costs two multiplications, $2(n-1)$ in all. By induction downwards, $\bar{p}_k = x_{k+1} \cdots x_n$, so $\bar{x}_k = p_{k-1} \, x_{k+1} \cdots x_n = \prod_{j \neq k} x_j$ for $k \geq 3$, and similarly for x_1, x_2.

(b) Each partial derivative is a product of $n-1$ numbers, which takes $n-2$ multiplications, so $n(n-2)$ in all. Forward mode runs n passes.

Each pass has $n-1$ value lines with one multiplication each and $n-1$ tangent lines $\dot{p}_k = \dot{p}_{k-1}x_k + p_{k-1}\dot{x}_k$ with two each, so $3n(n-1)$ multiplications in all. Both grow like n^2, against $3(n-1)$ for one evaluation and one sweep.

Exercise 4.8. (a) A path from x to L visits exactly one of u_i, w_i for each $i = 0, \dots, k$, since every edge goes from layer $i-1$ to layer i (with x before layer 0 and L after layer k), and both nodes of each layer are joined to both nodes of the next. Each of the 2^{k+1} choices of one node per layer is a path, so there are 2^{k+1} paths.

(b) Step 3 of the sweep updates one accumulator for each edge $i \to k$, adding $\bar{v}_k d_{ki}$ to the accumulator of v_i. There are 2 edges from x, 4 into each of the layers $1, \dots, k$, and 2 into L, so $4k+4$ in all.

Exercise 4.9. (a) $T/K + K \geq 2\sqrt{T} = 200$, with equality at $K = \sqrt{T} = 100$: 100 checkpoints and 100 states of one segment, 200 states in all, against 10 000 without checkpointing.

(b) The forward pass stores T/K_1 checkpoints. In the sweep, each segment of length K_1 is recomputed once from its checkpoint, keeping every K_2th state, K_1/K_2 of them. Each sub-segment of length K_2 is then recomputed from its stored state, and its K_2 states are kept only while the sweep passes through it. At most $T/K_1 + K_1/K_2 + K_2$ states are stored at once. Every step is recomputed at most once as part of a segment and at most once as part of a sub-segment, so the extra work is at most two forward passes. With $K_1 = T^{2/3}$ and $K_2 = T^{1/3}$ each of the three terms is $T^{1/3}$, giving $3T^{1/3}$ states: 300 for $T = 10^6$ instead of the 2000 of one level.

Exercise 4.10. Write $p = e^t + e^{-t}$ and $q = e^t - e^{-t}$, so $p' = q$, $q' = p$ and $\tanh t = q/p$. By the quotient rule,

$$\frac{d}{dt}\tanh t = \frac{q'p - qp'}{p^2} = \frac{p^2 - q^2}{p^2} = 1 - \tanh^2 t.$$

The primitive records one line, computing tanh of its argument for the value and $1-\tanh^2$ for the local derivative, through `elementary`:

```
tanh = elementary("tanh", math.tanh, lambda t: 1 - math.tanh(t) **
↪  2)
```

For $L = \tanh(xy)$ at $(0.5, -1)$, with $s = \tanh(-0.5)$, the engine returns $\partial L/\partial x = (1-s^2)\,y \approx -0.786448$ and $\partial L/\partial y = (1-s^2)\,x \approx 0.393224$. Its tape has 4 lines: x, y, $t = xy$ and $\tanh t$. Derived from `exp`, with e^t and e^{-t} each computed once, the tape would record e^t, then $-t$ (as $-1 \cdot t$) and e^{-t}, then the difference, the sum and the quotient: 6 lines in place of one, 9 in all.

B.5. Chapter 5

Exercise 5.1. As in Example 5.1, $\theta_k = ((1-2\eta)^k, (1-6\eta)^k)$.

(a) Both coordinates tend to 0 exactly when $|1-2\eta| < 1$ and $|1-6\eta| < 1$, that is, when $0 < \eta < 1$ and $0 < \eta < \frac{1}{3}$. So gradient descent converges exactly when $0 < \eta < \frac{1}{3}$, as Theorem 5.7 predicts with $L = 6$.

(b) $y_k = (1-6\eta)^k$ changes sign at every step exactly when $1 - 6\eta < 0$, that is, $\eta > \frac{1}{6}$.

(c) For $\eta = 0.3$ the factors are 0.4 and -0.8, so $\theta_3 = (0.064, -0.512)$. For $\eta = \frac{1}{3}$ the factors are $\frac{1}{3}$ and -1: $x_k = 3^{-k} \to 0$, but $y_k = (-1)^k$ jumps between 1 and -1 for ever, and $f(\theta_k) \to 3$.

Exercise 5.2. (a) If f is L-smooth then $|f'(x) - f'(0)| \le L|x|$ for every x, so $|f'(x) - f'(0)|/|x| \le L$ for $x \neq 0$. As $x \to 0$ the left side tends to $f''(0) = \frac{1}{4}$, so $L \ge \frac{1}{4}$.

(b) $x_1 = 0 - 4f'(0) = -4 \cdot \frac{1}{2} = -2$. Since $f'(x) > 0$ for every x, $x_{k+1} = x_k - 4f'(x_k) < x_k$. Suppose the decreasing sequence x_k were bounded below. Then it would converge to some $\bar{x}$, and $4f'(x_k) = x_k - x_{k+1} \to 0$. By continuity of f', $f'(\bar{x}) = 0$, which is impossible. So $x_k \to -\infty$. Each step lowers f (Corollary 5.1 with $\eta = 1/L$), and $f(x_k) \to 0$, the infimum, which is never attained.

Exercise 5.3. (a) Here $\nabla f(1,1) = (2,6)$ and

$$\varphi(t) = (1-2t)^2 + 3(1-6t)^2, \qquad \varphi'(t) = -4(1-2t) - 36(1-6t) = 224t - 40.$$

So $t_0 = 40/224 = \frac{5}{28}$, a minimum since $\varphi'' = 224 > 0$, and $\theta_1 = (1 - \frac{10}{28}, 1 - \frac{30}{28}) = (\frac{9}{14}, -\frac{1}{14})$. Then $\nabla f(\theta_1) = (\frac{9}{7}, -\frac{3}{7})$, and $(\frac{9}{7}, -\frac{3}{7}) \cdot (2,6) = \frac{18}{7} - \frac{18}{7} = 0$.

(b) Write $\mathbf{g} = \nabla f(\theta_k)$. By Lemma 5.1 with $\mathbf{d} = -\mathbf{g}$, $\varphi'(t) = -\nabla f(\theta_k - t\mathbf{g}) \cdot \mathbf{g}$. At the minimiser $t_k > 0$, $\varphi'(t_k) = 0$, which says $\nabla f(\theta_{k+1}) \cdot \mathbf{g} = 0$.

Exercise 5.4. (a) Expanding, $f(\theta) = \frac{1}{2}\theta^\top X^\top X\theta - (X^\top \mathbf{y})^\top \theta + \frac{1}{2}\|\mathbf{y}\|^2$. This is the quadratic of Example 5.2 with the symmetric matrix $A = X^\top X$ and $\mathbf{b} = X^\top \mathbf{y}$, plus a constant. So $\nabla f(\theta) = X^\top X\theta - X^\top \mathbf{y} = X^\top(X\theta - \mathbf{y})$, and the Hessian is $X^\top X$.

(b) If $X^\top X\mathbf{v} = \lambda \mathbf{v}$ with $\|\mathbf{v}\| = 1$, then $\lambda = \mathbf{v}^\top X^\top X\mathbf{v} = \|X\mathbf{v}\|^2 \geq 0$. By Theorem 5.4 with $\mu = 0$, f is convex, and by Example 5.2 it is L-smooth with L the largest eigenvalue.

(c) $X^\top X = \begin{pmatrix} 3 & 3 \\ 3 & 5 \end{pmatrix}$, with characteristic equation $\lambda^2 - 8\lambda + 6 = 0$ and eigenvalues $4 \pm \sqrt{10}$, about 0.838 and 7.162. Both are positive, so f is also $(4 - \sqrt{10})$-strongly convex, and

$$\kappa = \frac{4 + \sqrt{10}}{4 - \sqrt{10}} = \frac{(4 + \sqrt{10})^2}{6} = \frac{13 + 4\sqrt{10}}{3} \approx 8.55.$$

Exercise 5.5. By Example 1.8, $\nabla g(\mathbf{x}) = \nabla f(\mathbf{x}) - \mu\mathbf{x}$. So

$$g(\mathbf{y})-g(\mathbf{x})-\nabla g(\mathbf{x})\cdot(\mathbf{y}-\mathbf{x}) = f(\mathbf{y})-f(\mathbf{x})-\nabla f(\mathbf{x})\cdot(\mathbf{y}-\mathbf{x})-\frac{\mu}{2}(\|\mathbf{y}\|^2-\|\mathbf{x}\|^2-2\mathbf{x}\cdot(\mathbf{y}-\mathbf{x})),$$

and the bracket equals $\|\mathbf{y}\|^2-2\mathbf{x}\cdot\mathbf{y}+\|\mathbf{x}\|^2 = \|\mathbf{y}-\mathbf{x}\|^2$. So the left side is non-negative for all $\mathbf{x}$, $\mathbf{y}$ exactly when $f(\mathbf{y}) \geq f(\mathbf{x})+\nabla f(\mathbf{x})\cdot(\mathbf{y}-\mathbf{x})+\frac{\mu}{2}\|\mathbf{y}-\mathbf{x}\|^2$ for all $\mathbf{x}$, $\mathbf{y}$. By Theorem 5.3, the first condition says g is convex, and the second says f is μ-strongly convex.

Exercise 5.6. Take any $\mathbf{x}$ and any $\mathbf{d} \neq \mathbf{0}$, and put $\mathbf{y} = \mathbf{x} + \mathbf{d}$. By the definition of strong convexity and Lemma 5.4,

$$\frac{\mu}{2}\|\mathbf{d}\|^2 \leq f(\mathbf{y}) - f(\mathbf{x}) - \nabla f(\mathbf{x}) \cdot \mathbf{d} \leq \frac{L}{2}\|\mathbf{d}\|^2.$$

Dividing by $\frac{1}{2}\|\mathbf{d}\|^2 > 0$ gives $\mu \leq L$.

Exercise 5.7. (a) The characteristic equation $(3-\lambda)^2-1 = 0$ gives $\lambda = 2$ and $\lambda = 4$, with unit eigenvectors $\mathbf{u}_1 = (1,-1)/\sqrt{2}$ and $\mathbf{u}_2 = (1,1)/\sqrt{2}$. The minimiser is

$$\theta^* = A^{-1}\mathbf{b} = \frac{1}{8}\begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = (\tfrac{1}{4}, \tfrac{1}{4}).$$

By Theorem 5.7 with $L = 4$, gradient descent converges from every start exactly when $0 < \eta < \frac{1}{2}$.

(b) $\eta = 2/(2+4) = \frac{1}{3}$ and $\rho = (4-2)/(4+2) = \frac{1}{3}$.

(c) $\theta_1 = \mathbf{0} - \frac{1}{3}(A\mathbf{0} - \mathbf{b}) = (\frac{1}{3}, \frac{1}{3})$ and $\theta_2 = (\frac{1}{3}, \frac{1}{3}) - \frac{1}{3}((\frac{4}{3}, \frac{4}{3}) - (1,1)) = (\frac{2}{9}, \frac{2}{9})$. The starting error $\mathbf{e}_0 = -(\frac{1}{4}, \frac{1}{4}) = -\frac{\sqrt{2}}{4}\mathbf{u}_2$ is an eigenvector for $\lambda = 4$, so by Theorem 5.7 $\mathbf{e}_k = (1 - \frac{4}{3})^k\mathbf{e}_0 = (-\frac{1}{3})^k\mathbf{e}_0$. Its length shrinks by exactly $\frac{1}{3}$ at every step, and it changes sign each time: θ_1 overshoots θ^* and θ_2 falls short.

Exercise 5.8. (a) The flow is $\theta' = -\lambda\theta$, and since $\frac{d}{dt}(e^{\lambda t}\theta(t)) = e^{\lambda t}(\theta' + \lambda\theta) = 0$, the solution is $\theta(t) = \theta_0 e^{-\lambda t}$. Gradient descent gives $\theta_{k+1} = (1 - \eta\lambda)\theta_k$, so $\theta_k = (1 - \eta\lambda)^k\theta_0$.

(b) With $\eta = t/k$, $\theta_k = (1-\lambda t/k)^k \theta_0$. For $k > \lambda t$ the base is positive, and with $s = \lambda t/k$,

$$\ln\Big(1-\frac{\lambda t}{k}\Big)^k = k\ln(1-s) = -\lambda t \cdot \frac{\ln(1-s)}{-s}.$$

As $k \to \infty$, $s \to 0$ and $\ln(1-s)/(-s) \to 1$, the derivative of ln at 1. So the logarithm tends to $-\lambda t$, and by continuity of the exponential, $\theta_k \to \theta_0 e^{-\lambda t} = \theta(t)$.

(c) $\theta(1) = e^{-6} \approx 0.00248$. With $k = 6$, $\eta = \frac{1}{6}$ and $\theta_6 = (1-1)^6 = 0$: the first step jumps straight to the minimum. With $k = 60$, $\theta_{60} = 0.9^{60} \approx 0.00180$, and with $k = 600$, $\theta_{600} = 0.99^{600} \approx 0.00241$.

B.6. Chapter 6

Exercise 6.1. Let X_1 and X_2 be the two scores, so $S = X_1 + X_2$. By Example 6.1 each has mean $\frac{7}{2}$ and variance $\frac{35}{12}$. By Theorem 6.1, $\mathbb{E}[S] = \frac{7}{2} + \frac{7}{2} = 7$. The 36 outcomes are equally likely, so $P(X_1 = a \text{ and } X_2 = b) = \frac{1}{36} = \frac{1}{6}\cdot\frac{1}{6}$ and the scores are independent; by Theorem 6.3, $\mathrm{Var}[S] = 2 \cdot \frac{35}{12} = \frac{35}{6}$.

Listing the outcomes, the totals $2, 3, \dots, 12$ occur $1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1$ times. So $\mathbb{E}[S] = \frac{252}{36} = 7$ and $\mathbb{E}[S^2] = \frac{1974}{36}$, and by Lemma 6.2 $\mathrm{Var}[S] = \frac{1974}{36} - 49 = \frac{210}{36} = \frac{35}{6}$.

Exercise 6.2. (a) The six pairs give $G = -1.5$ for $(1,2)$ and $(2,1)$, -3.5 for $(1,6)$ and $(6,1)$, and -4 for $(2,6)$ and $(6,2)$. The mean is $-\frac{2(1.5+3.5+4)}{6} = -3$, still $f'(0)$, and the variance is $\frac{2(1.5^2+0.5^2+1^2)}{6} = \frac{7}{6}$, half the $\frac{7}{3}$ of sampling with replacement.

(b) There are $N(N-1)$ ordered pairs of different indices, all equally likely, so $\mathbb{E}[\mathbf{d}_I \cdot \mathbf{d}_J] = \frac{1}{N(N-1)} \sum_{a\neq b} \mathbf{d}_a \cdot \mathbf{d}_b$. Expanding $\|\sum_a \mathbf{d}_a\|^2$,

$$0 = \Big\|\sum_a \mathbf{d}_a\Big\|^2 = \sum_a \|\mathbf{d}_a\|^2 + \sum_{a\neq b} \mathbf{d}_a \cdot \mathbf{d}_b,$$

so $\sum_{a\neq b}\mathbf{d}_a\cdot\mathbf{d}_b=-N\sigma^2$ and $\mathbb{E}[\mathbf{d}_I\cdot\mathbf{d}_J]=-\sigma^2/(N-1)$.

(c) Let the outcomes be the lists $(i_1,\dots,i_B)$ of B different examples, all equally likely, and put $\mathbf{d}_a=\nabla f_a(\theta)-\nabla f(\theta)$, so $\sum_a\mathbf{d}_a=\mathbf{0}$. Each entry I_j takes every value in the same number of lists, so it is uniform on the N examples, and each pair of entries (I_i,I_j) with $i\neq j$ is uniform on the ordered pairs of different examples. The batch mean minus $\nabla f(\theta)$ is $\frac{1}{B}\sum_j\mathbf{d}_{I_j}$, whose expectation is $\mathbf{0}$, so the estimate is unbiased, and its variance is

$$\frac{1}{B^2}\Big(\sum_j\mathbb{E}[\|\mathbf{d}_{I_j}\|^2]+\sum_{i\neq j}\mathbb{E}[\mathbf{d}_{I_i}\cdot\mathbf{d}_{I_j}]\Big)=\frac{1}{B^2}\Big(B\sigma^2-\frac{B(B-1)\sigma^2}{N-1}\Big)=\frac{\sigma^2}{B}\cdot\frac{N-B}{N-1},$$

by (b). For $N=3$ and $B=2$ this is $\frac{14}{3}\cdot\frac{1}{2}\cdot\frac{1}{2}=\frac{7}{6}$, as in (a). For $B=N$ the variance is 0: the batch is the whole training set.

Exercise 6.3. (a) The step is $\theta_{k+1}=\theta_k-\eta(\theta_k-y_{I_k})$. Subtracting 3, $e_{k+1}=e_k-\eta(e_k+3-y_{I_k})=(1-\eta)e_k+\eta(y_{I_k}-3)$.

(b) Squaring,

$$e_{k+1}^2=(1-\eta)^2e_k^2+2\eta(1-\eta)\,e_k\,(y_{I_k}-3)+\eta^2(y_{I_k}-3)^2.$$

Fix the earlier draws, so that e_k is fixed, and average over I_k: the values y_i-3 are -2, -1 and 3, with average 0 and mean square $\sigma^2=\frac{14}{3}$. So the average over I_k is $(1-\eta)^2e_k^2+\eta^2\sigma^2$, and by Lemma 6.3 $\mathbb{E}[e_{k+1}^2]=(1-\eta)^2\,\mathbb{E}[e_k^2]+\eta^2\sigma^2$.

(c) Write $u_k=\mathbb{E}[e_k^2]$ and $a=(1-\eta)^2$. The constant solution of $u_{k+1}=au_k+\eta^2\sigma^2$ is $u^*=\eta^2\sigma^2/(1-a)=\eta^2\sigma^2/(\eta(2-\eta))=\eta\sigma^2/(2-\eta)$, and u_k-u^* satisfies $u_{k+1}-u^*=a(u_k-u^*)$, so

$$u_k=u^*+(1-\eta)^{2k}(u_0-u^*).$$

For $0<\eta<2$, $0\le a<1$ and $u_k\to u^*$. With $\eta=0.1$, $u^*=\frac{0.1\cdot14/3}{1.9}\approx0.246$.

Exercise 6.4. (a) Let $\theta_{-1} = \theta_0$. We show by induction that $\mathbf{w}_k = \theta_k - \theta_{k-1}$ and that the points agree with Equation 6.3. For $k = 0$, $\mathbf{w}_0 = \mathbf{0} = \theta_0 - \theta_{-1}$. If $\mathbf{w}_k = \theta_k - \theta_{k-1}$, then

$$\theta_{k+1} = \theta_k + \beta \mathbf{w}_k - \eta \nabla f(\theta_k) = \theta_k - \eta \nabla f(\theta_k) + \beta(\theta_k - \theta_{k-1}),$$

which is Equation 6.3, and $\mathbf{w}_{k+1} = \theta_{k+1} - \theta_k$ by definition.

(b) The recurrence $\mathbf{w}_k = \beta \mathbf{w}_{k-1} - \eta \nabla f(\theta_{k-1})$ has the same form as Definition 6.6, and the induction in the proof of Theorem 6.8 gives $\mathbf{w}_k = -\eta \sum_{j=0}^{k-1} \beta^{k-1-j} \nabla f(\theta_j)$. Let m_k be the exponential moving average of the gradients $\mathbf{g}_j = \nabla f(\theta_{j-1})$. By Theorem 6.8, $m_k = (1-\beta) \sum_{j=0}^{k-1} \beta^{k-1-j} \nabla f(\theta_j)$, so $\mathbf{w}_k = -\frac{\eta}{1-\beta} m_k$: the heavy-ball step is a moving average of past gradients, scaled by $\eta/(1-\beta)$.

Exercise 6.5. (a) By Theorem 6.6, every solution tends to 0 exactly when $\eta < 2(1 + \frac{1}{4}) = \frac{5}{2}$.

(b) With $\sqrt{\beta} = \frac{1}{2}$, the roots are complex exactly when $(1 - \frac{1}{2})^2 < \eta < (1 + \frac{1}{2})^2$, that is, $\frac{1}{4} < \eta < \frac{9}{4}$; at the two ends there is a repeated root. For $\eta = 1$ the characteristic equation is $r^2 - \frac{1}{4} r + \frac{1}{4} = 0$, with roots

$$r = \frac{\frac{1}{4} \pm \sqrt{\frac{1}{16} - 1}}{2} = \frac{1 \pm i\sqrt{15}}{8}, \qquad |r| = \frac{\sqrt{1 + 15}}{8} = \frac{1}{2} = \sqrt{\beta}.$$

Exercise 6.6. (a) We need $0.9^k \le 0.01$, that is, $k \ge \ln 0.01 / \ln 0.9 \approx 43.7$, so $k = 44$.

(b) Let $k = 2h$. The weights on $g_1, \dots, g_h$ are $(1-\beta)\beta^{k-j}$ for $j = 1, \dots, h$, with exponents running from $k-1$ down to h. Their sum is

$$(1-\beta)(\beta^h + \beta^{h+1} + \cdots + \beta^{2h-1}) = \beta^h (1-\beta)(1 + \beta + \cdots + \beta^{h-1}) = \beta^h (1 - \beta^h).$$

For $\beta = 0.9$ and $k = 20$ the older half carries $0.9^{10}(1 - 0.9^{10}) \approx 0.227$ of the weight and the newer half $1 - 0.9^{10} \approx 0.651$.

Exercise 6.7. (a) Write primes for the quantities when Adam runs on cf. We show by induction on k that $\theta'_k = \theta_k$, $\mathbf{m}'_k = c\,\mathbf{m}_k$ and $\mathbf{v}'_k = c^2\mathbf{v}_k$. All hold for $k = 0$. If they hold for $k-1$, then $\mathbf{g}'_k = c\nabla f(\theta'_{k-1}) = c\,\mathbf{g}_k$, so the recurrences give $\mathbf{m}'_k = c\,\mathbf{m}_k$ and $\mathbf{v}'_k = c^2\mathbf{v}_k$. Then $\hat{m}'_{k,i} = c\,\hat{m}_{k,i}$ and $\sqrt{\hat{v}'_{k,i}} = c\sqrt{\hat{v}_{k,i}}$ since $c > 0$, and with $\epsilon = 0$ the factor c cancels in the step, so $\theta'_k = \theta_k$.

(b) By Theorem 6.9, if $g_{k,i} = g$ for every k then $m_{k,i} = (1-\beta_1^k)g$ and $v_{k,i} = (1-\beta_2^k)g^2$, so $\hat{m}_{k,i} = g$ and $\sqrt{\hat{v}_{k,i}} = |g|$. The step is $-\eta_k\, g/(|g| + \epsilon)$, of size just under η_k, whatever the size of g.

Exercise 6.8. With $\eta_k = \eta_0(1 - k/K)$,

$$\sum_{k=0}^{K-1}\Big(1-\frac{k}{K}\Big) = K - \frac{1}{K}\cdot\frac{(K-1)K}{2} = \frac{K+1}{2}, \qquad \sum_{k=0}^{K-1}\Big(1-\frac{k}{K}\Big)^2 = \frac{1}{K^2}\sum_{j=1}^{K} j^2 = \frac{(K+1)(2K+1)}{6K},$$

where the second sum substitutes $j = K - k$ and uses $\sum_{j=1}^{K} j^2 = K(K+1)(2K+1)/6$. Multiplying by η_0 and η_0^2 gives the two sums. Then

$$\frac{2(f(\theta_0) - f^*) + L\sigma^2\eta_0^2\frac{(K+1)(2K+1)}{6K}}{\eta_0\frac{K+1}{2}} = \frac{4(f(\theta_0) - f^*)}{\eta_0(K+1)} + \frac{L\sigma^2\eta_0(2K+1)}{3K}.$$

The theorem needs $\eta_0 \le 1/L$, so that every $\eta_k \le 1/L$.

B.7. Chapter 7

Exercise 7.1. (a) Layer l has an $n_l \times n_{l-1}$ weight matrix and n_l biases, so $n_l(n_{l-1}+1)$ parameters. For widths $3, 4, 4, 2$ the total is $4\cdot 4 + 4\cdot 5 + 2\cdot 5 = 46$.

(b) The forward pass computes $W_1\mathbf{x}$ and $W_2\mathbf{a}_1$, with $64\cdot 16 + 16\cdot 64 = 2048$ multiplications. The backward pass computes the outer products

$\delta_2 \mathbf{a}_1^\top$ and $\delta_1 \mathbf{x}^\top$ (1024 each), $W_2^\top \delta_2$ (1024), the product with $\sigma'(\mathbf{z}_1)$ (64), and the input gradient $W_1^\top \delta_1$ (1024): 4160 multiplications, about 2.03 times the forward pass.

Exercise 7.2. At $\mathbf{x} = \mathbf{0}$ the pre-activation is $\mathbf{z} = \mathbf{b} = (0, 1)$, and $\tanh' = 1 - \tanh^2$. By Equation 7.1,

$$J_F(\mathbf{0}) = \begin{pmatrix} 1 & 0 \\ 0 & 1 - \tanh^2 1 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 0 & -(1 - \tanh^2 1) \end{pmatrix} \approx \begin{pmatrix} 1 & 2 \\ 0 & -0.420 \end{pmatrix}.$$

Exercise 7.3. Forward: $\mathbf{z}_1 = (1, -2)^\top + (1, 1)^\top = (2, -1)^\top$, $\mathbf{a}_1 = (2, 0)^\top$, $z_2 = 3 \cdot 2 + 0 = 6$, and $L = \frac{1}{2}(6 - 2)^2 = 8$. Backward: $\delta_2 = z_2 - 2 = 4$,

$$\delta_1 = (W_2^\top \delta_2) \odot \mathrm{relu}'(\mathbf{z}_1) = (12, 4)^\top \odot (1, 0)^\top = (12, 0)^\top.$$

By Equation 7.4, $\partial L / \partial W_2 = \delta_2 \mathbf{a}_1^\top = (8, 0)$, $\partial L / \partial b_2 = 4$, $\partial L / \partial W_1 = \delta_1 x = (12, 0)^\top$ and $\partial L / \partial \mathbf{b}_1 = (12, 0)^\top$. The second hidden unit is inactive, so its weight and bias get partial derivative 0, and so does the output weight that reads it, because its input $(\mathbf{a}_1)_2$ is 0.

Exercise 7.4. (a) For f_1 the engine multiplies along each path from x to the output and adds: $1 \cdot g \cdot 1 + (-1) \cdot g \cdot (-1) = 2g$. For f_2, the path through $\mathrm{relu}(\mathrm{relu}(x))$ meets two ReLUs whose inputs are both 0, giving $g \cdot g$, and the path through $-\mathrm{relu}(-x)$ gives $(-1) \cdot g \cdot (-1) = g$. The total is $g^2 + g$.

(b) For $x \geq 0$, $f_2(x) = \mathrm{relu}(x) - 0 = x$; for $x < 0$, $f_2(x) = \mathrm{relu}(0) - (-x) = x$. Example 7.3 showed $f_1(x) = x$. So both derivatives at 0 are 1. The engine is right for f_1 only if $2g = 1$, that is $g = \frac{1}{2}$, and right for f_2 only if $g^2 + g = 1$, whose roots are $(-1 \pm \sqrt{5})/2$, neither of which is $\frac{1}{2}$.

Exercise 7.5. The function has breakpoints 0, 1, 2 and slopes 0, 1, -1, 0 on the four pieces of the real line, so the slope changes by $+1$, -2 and $+1$:

$$h(x) = \mathrm{relu}(x) - 2\,\mathrm{relu}(x - 1) + \mathrm{relu}(x - 2).$$

For $x \le 0$ all terms are 0. For $x \ge 2$ the sum is $x - 2(x-1) + (x-2) = 0$. Between, the argument of Lemma 7.3 shows h is affine on $[0,1]$ and $[1,2]$ with the right values at 0, 1, 2.

Exercise 7.6. (a) For each x, one of $\mathrm{relu}(x)$ and $\mathrm{relu}(-x)$ is $|x|$ and the other is 0. If $x \ge y$, $y + \mathrm{relu}(x-y) = y + x - y = x$; if $x < y$, it is $y + 0 = y$.

(b) Write $y = \mathrm{relu}(y) - \mathrm{relu}(-y)$, as in Example 7.3. Then

$$\max(x, y) = \mathrm{relu}(x - y) + \mathrm{relu}(y) - \mathrm{relu}(-y),$$

a network with hidden weight matrix $\begin{pmatrix} 1 & -1 \\ 0 & 1 \\ 0 & -1 \end{pmatrix}$, zero hidden biases, output weights $(1, 1, -1)$ and output bias 0.

Exercise 7.7. (a) The error is $1/(4N^2) \le 10^{-6}$ when $N^2 \ge 250000$, so the least N is 500.

(b) On $[0, 1/N]$ the interpolant is the chord from $(0,0)$ to $(1/N, 1/\sqrt{N})$, so $\hat{f}(x) = \sqrt{N}\,x$ there. At $x = 1/(4N)$, $\sqrt{x} = 1/(2\sqrt{N})$ and $\hat{f}(x) = 1/(4\sqrt{N})$, so $f(x) - \hat{f}(x) = 1/(4\sqrt{N})$. For this to be at most 10^{-6} we need $\sqrt{N} \ge 250000$, so $N \ge 6.25 \times 10^{10}$. The function $\sqrt{x}$ has an infinite slope at 0, and the interpolant is worst there.

Exercise 7.8. (a) By Equation 7.6 with $m = n$, $\mathbb{E}\|\mathrm{relu}(W\mathbf{x})\|^2 = \frac{ns^2}{2}\|\mathbf{x}\|^2$, which equals $\|\mathbf{x}\|^2$ when $s = \sqrt{2/n}$: $s \approx 0.354$ for $n = 16$ and $s \approx 0.177$ for $n = 64$.

(b) The factor is $64 \times 0.08^2/2 = 0.2048$. Since $0.2048^8 \approx 3.1 \times 10^{-6}$ and $0.2048^9 \approx 6.3 \times 10^{-7}$, nine layers are needed.

B.8. Chapter 8

Exercise 8.1. (a) If $p(v) = 1$, every other value is 0, and $H(p) = -1 \cdot \ln 1 = 0$. Conversely, each term $-p(v) \ln p(v)$ is non-negative and is 0 only when $p(v)$ is 0 or 1. If $H(p) = 0$ every term is 0, so every $p(v)$ is 0 or 1, and since the values sum to 1, exactly one of them is 1.

(b) $H(p) = \frac{1}{2} \ln 2 + 2 \cdot \frac{1}{4} \ln 4 = \frac{3}{2} \ln 2 \approx 1.040$. By the proof of Corollary 8.1, $D_{\mathrm{KL}}(p \,\|\, u) = \ln 3 - \frac{3}{2} \ln 2 \approx 0.0589$. And

$$D_{\mathrm{KL}}(u \,\|\, p) = \tfrac{1}{3}\Big(\ln \frac{1/3}{1/2} + 2 \ln \frac{1/3}{1/4}\Big) = \tfrac{1}{3} \ln \frac{32}{27} \approx 0.0566.$$

Exercise 8.2. For $0 < \theta < 1$ the log-likelihood is $k \ln \theta + (N - k) \ln(1 - \theta)$, with derivative $k/\theta - (N - k)/(1 - \theta)$. This is 0 only at $\theta = k/N$; it is positive for smaller θ and negative for larger, so $\theta = k/N$ is the maximum on $(0, 1)$. At $\theta = 0$ and $\theta = 1$ the likelihood is 0, because $0 < k < N$. With $V = \{\text{heads}, \text{tails}\}$, Theorem 8.2 gives the same answer, $\hat{p}(\text{heads}) = k/N$. When $k = 0$ the likelihood $(1 - \theta)^N$ is decreasing, so the maximum is at $\theta = 0 = k/N$, again as the theorem says; the calculus argument fails there because the maximum is at an endpoint.

Exercise 8.3. (a) If $\mathbf{z} - \mathbf{z}' = c\mathbf{1}$, the two softmaxes agree by Equation 8.4. Conversely, let $\mathbf{p}$ be the common value, $S = \sum_j e^{z_j}$ and $S' = \sum_j e^{z'_j}$. Then $z_i = \ln p_i + \ln S$ and $z'_i = \ln p_i + \ln S'$ for every i, so $\mathbf{z} - \mathbf{z}' = (\ln S - \ln S')\mathbf{1}$.

(b) By (a) and $\operatorname{softmax}(\ln \mathbf{p}) = \mathbf{p}$, the solutions are $z_i = \ln p_i + c$. The condition $z_1 = 0$ gives $c = \ln 2$, so $\mathbf{z} = (0, \ln \frac{2}{3}, \ln \frac{1}{3}) \approx (0, -0.405, -1.099)$.

Exercise 8.4. The exponentials are $1, 2, 3$, so $\mathbf{p} = (\frac{1}{6}, \frac{1}{3}, \frac{1}{2})$ and

$$J = \operatorname{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top = \begin{pmatrix} \frac{5}{36} & -\frac{1}{18} & -\frac{1}{12} \\ -\frac{1}{18} & \frac{2}{9} & -\frac{1}{6} \\ -\frac{1}{12} & -\frac{1}{6} & \frac{1}{4} \end{pmatrix}.$$

The first row sums to $\frac{5}{36} - \frac{2}{36} - \frac{3}{36} = 0$, and the others likewise. For $\mathbf{v} = \mathbf{e}_1$, $\mathbf{v}^\top J \mathbf{v} = J_{11} = \frac{5}{36} = p_1(1 - p_1)$, the variance of the indicator of outcome 1. For $\mathbf{v} = \mathbf{1}$ it is 0, by (b) of Theorem 8.4.

Exercise 8.5. (a) $\sigma'(t) = e^{-t}/(1 + e^{-t})^2 = \sigma(t) \cdot e^{-t}/(1 + e^{-t})$, and $e^{-t}/(1 + e^{-t}) = 1 - \sigma(t)$. Multiplying numerator and denominator by e^t, $1 - \sigma(t) = 1/(e^t + 1) = \sigma(-t)$.

(b) By (a), $(\ln \sigma)' = 1 - \sigma$ and $(\ln(1 - \sigma))' = -\sigma$, so $L'(t) = -y(1 - \sigma(t)) + (1 - y)\sigma(t) = \sigma(t) - y$. With logits $\mathbf{z} = (t, 0)$, $\operatorname{softmax}(\mathbf{z})_1 = \sigma(t)$ (the case $n = 2$ after Definition 8.5), and L is the cross-entropy loss with target $\mathbf{e}_1$ if $y = 1$ and $\mathbf{e}_2$ if $y = 0$. By Theorem 8.5 its partial derivative with respect to $z_1 = t$ is $p_1 - y = \sigma(t) - y$.

Exercise 8.6. (a) By Corollary 8.1, $L(\mathbf{z}) = H(\mathbf{y}, \mathbf{p}) \geq H(\mathbf{y})$, with equality only if $\mathbf{p} = \mathbf{y}$. Since every $y_i > 0$, there are logits with $\operatorname{softmax}(\mathbf{z}) = \mathbf{y}$, and by Exercise 8.3 they are exactly $\mathbf{z} = \ln \mathbf{y} + c\mathbf{1}$ for real c, where $\ln \mathbf{y}$ is taken componentwise.

(b) For $\mathbf{y} = \mathbf{e}_y$, $L(\mathbf{z}) = -\ln p_y > 0$ because $p_y < 1$. Along $\mathbf{z} = t\mathbf{e}_y$, $p_y = e^t/(e^t + n - 1) \to 1$ as $t \to \infty$, so $L \to 0$. The infimum is 0 and it is not attained.

(c) The minimiser has $z_y - z_i = \ln(y_y/y_i)$ with $y_y = 0.9 + 0.1/27$ and $y_i = 0.1/27$, so $z_y - z_i = \ln(1 + 0.9 \cdot 270) = \ln 244 \approx 5.50$. Label smoothing keeps the optimal logits finite.

Exercise 8.7. The negative log-likelihood is

$$\frac{1}{2s^2} \sum_{i=1}^{N} (y_i - \mu)^2 + \frac{N}{2} \ln(2\pi) + N \ln s.$$

(a) For fixed s, this is a quadratic in μ with positive leading coefficient $N/(2s^2)$. Its derivative, $-\frac{1}{s^2} \sum_i (y_i - \mu)$, is 0 only at $\mu = \bar{y}$, which is therefore the minimiser.

(b) Let $Q = \sum_i (y_i - \bar{y})^2$, which is positive because the y_i are not all equal. At $\mu = \bar{y}$ the function of s is $g(s) = Q/(2s^2) + N \ln s + \text{const}$, with $g'(s) = -Q/s^3 + N/s = (Ns^2 - Q)/s^3$. This is negative for $s^2 < Q/N$ and positive for $s^2 > Q/N$, so g is smallest at $s^2 = Q/N$.

Exercise 8.8. Put $w_i = z_i - \bar{z}$, so that $\sum_i w_i = 0$. By Lemma 8.1 with $x = e^{w_i}$, $w_i \le e^{w_i} - 1$, so $\frac{1}{n}\sum_i e^{w_i} \ge \frac{1}{n}\sum_i (1 + w_i) = 1$. Hence

$$\operatorname{lse}(\mathbf{z}) = \bar{z} + \ln n + \ln\Big(\frac{1}{n}\sum_i e^{w_i}\Big) \ge \ln n + \bar{z},$$

and by Theorem 8.5, $-\ln \operatorname{softmax}(\mathbf{z})_y = \operatorname{lse}(\mathbf{z}) - z_y \ge \ln n + \bar{z} - z_y$. If all z_i are equal, every $w_i = 0$ and both sides equal $\ln n$. Taking expectations, linearity gives $\mathbb{E}[L] \ge \ln n + \frac{1}{n}\sum_i \mathbb{E}[z_i] - \mathbb{E}[z_y] = \ln n$.

B.9. Chapter 9

Exercise 9.1. (a) With $\mathtt{a} = 0$, $\mathtt{d} = 3$ and $\diamond = 26$, `ada` is $(26, 0, 3, 0, 26)$.

(b) A name with n letters is wrapped into $n + 2$ tokens, and every token except the first is predicted, so it gives $n + 1$ predictions. Adding over the m names gives $M + m$. The five names have $M = 4{+}3{+}4{+}3{+}3 = 17$ letters and $m = 5$, so 22 predictions.

Exercise 9.2. Two names, `ava` and `eva`, have second letter `v`, so $P(X_2 = \mathtt{v}) = 2/5$. Of these two, one starts with `a`, so

$$P(X_1 = \mathtt{a} \mid X_2 = \mathtt{v}) = \frac{P(X_1 = \mathtt{a}, X_2 = \mathtt{v})}{P(X_2 = \mathtt{v})} = \frac{1/5}{2/5} = \frac{1}{2},$$

and $P(X_2 = \mathtt{v})\, P(X_1 = \mathtt{a} \mid X_2 = \mathtt{v}) = \frac{2}{5} \cdot \frac{1}{2} = \frac{1}{5}$, as before. The chain rule holds in either order because both orders come from the same definition Equation 9.1.

Exercise 9.3. Multiplying the transition probabilities of the pairs,

$$P(\texttt{anna}) = p(\texttt{a} \mid \diamond)\, p(\texttt{n} \mid \texttt{a})\, p(\texttt{n} \mid \texttt{n})\, p(\texttt{a} \mid \texttt{n})\, p(\diamond \mid \texttt{a}) = \tfrac{3}{5} \cdot \tfrac{1}{8} \cdot \tfrac{1}{2} \cdot \tfrac{1}{2} \cdot \tfrac{5}{8} = \tfrac{3}{256},$$
$$P(\texttt{emma}) = \tfrac{2}{5} \cdot \tfrac{1}{2} \cdot \tfrac{1}{2} \cdot \tfrac{1}{2} \cdot \tfrac{5}{8} = \tfrac{1}{32}.$$

The other names have $P(\texttt{ava}) = P(\texttt{ada}) = \tfrac{3}{5} \cdot \tfrac{1}{8} \cdot 1 \cdot \tfrac{5}{8} = \tfrac{3}{64}$ and $P(\texttt{eva}) = \tfrac{1}{8}$ from Example 9.2, so `eva` is the most probable. It is short, and after its first letter every step except the last has probability $1/2$ or 1.

Exercise 9.4. (a) Summing the numerators of row a gives $\sum_b (n_{ab} + 1) = n_a + |V|$, which is the denominator.

(b) The counts $m_{ab} = n_{ab} + 1$ have row totals $m_a = \sum_b m_{ab} = n_a + |V| > 0$. By Theorem 9.2, the maximum-likelihood model for them is m_{ab}/m_a, which is Equation 9.5.

(c) $p(\diamond \mid \texttt{a}) = (5 + 1)/(8 + 7) = 2/5$, smaller than $5/8$: smoothing takes probability from the observed pairs and gives it to the unobserved ones.

Exercise 9.5. (a) Each p_i is a probability, so $0 < p_i \leq 1$ and $-\ln p_i \geq 0$. So $L \geq 0$ and $e^L \geq e^0 = 1$. Equality holds exactly when $L = 0$, that is, when every $-\ln p_i$ is 0, so every $p_i = 1$.

(b) $\mathrm{PPL}_A \,/\, \mathrm{PPL}_B = e^{L_A - L_B} = e^{-0.1} \approx 0.905$: model B's perplexity is about 1.105 times model A's.

Exercise 9.6. (a) For a string $w = (w_1, \ldots, w_k)$ of letters, with $w_0 = \diamond$, write $\pi(w) = \prod_{t=1}^{k} p(w_t \mid w_{t-1})$ for the probability that generation produces $w_1, \ldots, w_k$ first. Then $r_k = \sum_w \pi(w)$ over letter strings of length k, and by Equation 9.3 the name w has probability $\pi(w)\, p(\diamond \mid w_k)$, so $q_k = \sum_w \pi(w)\, p(\diamond \mid w_k)$. A string of $k + 1$ letters is a string w of k letters followed by a letter c, so

$$r_{k+1} = \sum_w \pi(w) \sum_{c \neq \diamond} p(c \mid w_k) = \sum_w \pi(w)(1 - p(\diamond \mid w_k)) = r_k - q_k,$$

and since $1 - p(\diamond \mid w_k) \leq 1 - \varepsilon$, also $r_{k+1} \leq (1 - \varepsilon) r_k$.

(b) Adding $q_k = r_k - r_{k+1}$ for $k = 0, \dots, K$, the sum telescopes: $\sum_{k=0}^{K} q_k = r_0 - r_{K+1} = 1 - r_{K+1}$. By (a) and induction, $0 \le r_{K+1} \le (1-\varepsilon)^{K+1}$, which tends to 0, so the total probability of all names is $\sum_{k \ge 0} q_k = 1$.

Exercise 9.7. Call the tokens 1 and 2. With $d = 1$ the embeddings are numbers u_1, u_2, and $W = (w_1, w_2)^\top$, so the logits after token a are $(w_1 u_a, w_2 u_a)$ and

$$p(1 \mid a) = \frac{e^{w_1 u_a}}{e^{w_1 u_a} + e^{w_2 u_a}} = \frac{1}{1 + e^{-(w_1 - w_2) u_a}}.$$

Given a bigram model with positive entries, take $w_1 = 1$, $w_2 = 0$ and $u_a = \ln(p(1 \mid a)/p(2 \mid a))$. Then

$$\frac{1}{1 + e^{-u_a}} = \frac{1}{1 + p(2 \mid a)/p(1 \mid a)} = \frac{p(1 \mid a)}{p(1 \mid a) + p(2 \mid a)} = p(1 \mid a),$$

and the probability of token 2 is $1 - p(1 \mid a) = p(2 \mid a)$ because both rows sum to 1.

Exercise 9.8. Write $\mathbf{g} = \mathbf{p} - \mathbf{e}_b$, so that $\partial \ell / \partial z_i = g_i$ by Theorem 8.5. Write ϵ_{kj} for entry (k, j) of E. The logits are $z_i = \sum_j w_{ij} \, (E^\top \mathbf{e}_a)_j = \sum_j w_{ij} \, \epsilon_{aj}$, so ϵ_{kj} appears only if $k = a$, and then $\partial z_i / \partial \epsilon_{aj} = w_{ij}$. By the chain rule,

$$\frac{\partial \ell}{\partial \epsilon_{kj}} = [k = a] \sum_i g_i w_{ij} = [k = a] \, (W^\top \mathbf{g})_j,$$

which is entry (k, j) of $\mathbf{e}_a (W^\top \mathbf{g})^\top$. All rows other than row a are zero, so a gradient step changes only $\mathbf{u}_a$.

Exercise 9.9. Each prediction adds one to a count in the row of its context, the $n-1$ tokens before it. N predictions touch at most N different rows, so at most N rows are non-zero. A 4-gram table over 27 tokens has $27^3 = 19{,}683$ rows, so at least $1 - 1000/19{,}683 \approx 0.949$ of them are empty.

B.10. Chapter 10

Exercise 10.1. The scores are $qk_j/\sqrt{1} = 2, 0, -2$, so the weights are

$$\mathbf{a} = \frac{(e^2, 1, e^{-2})}{e^2 + 1 + e^{-2}} \approx (0.867, 0.117, 0.016),$$

and the output is

$$y = \frac{e^2 + 2 + 3e^{-2}}{e^2 + 1 + e^{-2}} \approx 1.149.$$

The first key matches the positive query best, so the output is close to $v_1 = 1$.

Exercise 10.2. (a) The new scores are $\mathbf{q} \cdot (\mathbf{k}_j + \mathbf{b})/\sqrt{d_h} = s_j + c$ with $c = \mathbf{q} \cdot \mathbf{b}/\sqrt{d_h}$, the same for every j. Then

$$\frac{e^{s_j+c}}{\sum_r e^{s_r+c}} = \frac{e^c e^{s_j}}{e^c \sum_r e^{s_r}} = \frac{e^{s_j}}{\sum_r e^{s_r}},$$

so the weights do not change (this is Theorem A.3).

(b) $\sum_j a_j(\mathbf{v}_j + \mathbf{w}) = \sum_j a_j \mathbf{v}_j + (\sum_j a_j)\mathbf{w} = \mathbf{y} + \mathbf{w}$, because the weights sum to 1.

Exercise 10.3. Let s^* be the largest score. Dividing numerator and denominator by $e^{\lambda s^*}$,

$$a_j = \frac{e^{\lambda(s_j - s^*)}}{\sum_r e^{\lambda(s_r - s^*)}}.$$

For $r \in M$ the exponent is 0; for $r \notin M$ it is λ times a negative number, so the term tends to 0 as $\lambda \to \infty$. The denominator tends to $|M|$, and the numerator to 1 if $j \in M$ and 0 otherwise. The output $\sum_j a_j \mathbf{v}_j$ therefore tends to $\frac{1}{|M|} \sum_{j \in M} \mathbf{v}_j$. This is the exact lookup of Section 10.1 when one key matches best. As $\lambda \to 0$ every $e^{\lambda s_j} \to 1$, so the weights tend to $1/T$ and the output to the plain average of all the values.

Exercise 10.4. (a) The proof of Theorem 10.1 goes through with $\mathbb{E}[q_i^2] = \sigma^2$ and $\mathbb{E}[k_i^2] = \tau^2$: the mean is still 0, the terms with $i \neq j$ still vanish, and each of the d_h diagonal terms is $\mathbb{E}[q_i^2]\,\mathbb{E}[k_i^2] = \sigma^2\tau^2$.

(b) Component i of $\mathbf{q}$ is $q_i = \sum_{j=1}^{16} w_{ij}x_j$, where the w_{ij} are independent with mean 0 and variance $0.08^2 = 0.0064$. So $\mathbb{E}[q_i] = 0$, and expanding the square,

$$\mathbb{E}[q_i^2] = \sum_{j,l} x_j x_l\,\mathbb{E}[w_{ij}w_{il}] = \sum_j x_j^2\,(0.0064) = 16 \cdot 0.0064 = 0.1024,$$

because $\mathbb{E}[w_{ij}w_{il}] = \mathbb{E}[w_{ij}]\,\mathbb{E}[w_{il}] = 0$ for $j \neq l$ by Theorem 6.2, and $\mathbb{E}[w_{ij}^2] = 0.0064$ by Lemma 6.2. So $\mathrm{Var}[q_i] = 0.1024$, and the same holds for each k_i. The hypotheses of (a) hold: $\mathbf{q}$ is a function of W_Q and $\mathbf{k}$ of W_K, whose entries are independent of each other, and different components of $\mathbf{q}$ use different rows of W_Q, which are disjoint sets of independent entries (the general facts cited after Theorem 10.1). By (a) with $d_h = 4$, $\mathrm{Var}[\mathbf{q}\cdot\mathbf{k}] = 4 \cdot 0.1024^2 \approx 0.0419$, and the score $\mathbf{q}\cdot\mathbf{k}/2$ has variance $0.1024^2 \approx 0.0105$, so standard deviation 0.1024. Scores of size about 0.1 change e^s by about 10%, so at the start of training each head spreads its weight nearly evenly over the positions it can see, and attention starts as an approximate average of the earlier values.

Exercise 10.5. (a) With a cache, step t computes the new query against the t keys so far, t dot products, for a total of $\sum_{t=1}^{T} t = T(T+1)/2$. (b) Without a cache, step t redoes attention for every position $r \leq t$, which costs $\sum_{r=1}^{t} r = t(t+1)/2$ dot products, for a total of

$$\sum_{t=1}^{T} \frac{t(t+1)}{2} = \frac{T(T+1)(T+2)}{6}.$$

(By induction: the case $T = 1$ is $1 = 1$, and adding $(T+1)(T+2)/2$ to $T(T+1)(T+2)/6$ gives $(T+1)(T+2)(T+3)/6$.) For $T = 16$ these are 136 and 816. The cache turns a cost that grows like T^3 into one that grows like T^2.

Exercise 10.6. The weights do not depend on the values, and $\mathbf{y} = \sum_i a_i \mathbf{v}_i$ is linear in $\mathbf{v}_j$ with coefficient a_j, so its Jacobian with respect to $\mathbf{v}_j$ is $a_j I$.

The key $\mathbf{k}_j$ enters only the score $s_j = \mathbf{q} \cdot \mathbf{k}_j / \sqrt{d_h}$, whose derivative with respect to $\mathbf{k}_j$ is the row $\mathbf{q}^\top / \sqrt{d_h}$. By Equation 10.3,

$$\frac{\partial \mathbf{y}}{\partial s_j} = \sum_i \mathbf{v}_i \frac{\partial a_i}{\partial s_j} = \sum_i \mathbf{v}_i \, a_i([i = j] - a_j) = a_j \mathbf{v}_j - a_j \mathbf{y} = a_j(\mathbf{v}_j - \mathbf{y}).$$

By the chain rule, the Jacobian with respect to $\mathbf{k}_j$ is the column $a_j(\mathbf{v}_j - \mathbf{y})$ times the row $\mathbf{q}^\top / \sqrt{d_h}$.

Exercise 10.7. For $(x_1, x_2) = (1, 2)$: position 1 attends only to itself, so $y_1 = v_1 = 1$. Position 2 has query 2 and scores $2 \cdot 1 = 2$ and $2 \cdot 2 = 4$, so

$$y_2 = \frac{e^2 \cdot 1 + e^4 \cdot 2}{e^2 + e^4} = \frac{1 + 2e^2}{1 + e^2} \approx 1.881.$$

For $(2, 1)$: $y'_1 = 2$, and position 2 has query 1 and scores 2 and 1, so

$$y'_2 = \frac{e^2 \cdot 2 + e \cdot 1}{e^2 + e} = \frac{2e + 1}{e + 1} \approx 1.731.$$

The swapped outputs would be $(y_2, y_1) \approx (1.881, 1)$, but the outputs for the swapped input are $(2, 1.731)$.

Exercise 10.8. The product $W_O \mathbf{o}_t$ is the combination of the columns of W_O with weights the components of $\mathbf{o}_t$. Grouping the columns into the h blocks of d_h, block i of columns is multiplied by block i of $\mathbf{o}_t$, which is $\mathbf{o}_t^{(i)}$, so $W_O \mathbf{o}_t = \sum_i W_O^{(i)} \mathbf{o}_t^{(i)}$. Head i together with $W_O^{(i)}$ is a single-head attention whose output matrix is $d \times d_h$, and the layer adds the h results.

B.11. Chapter 11

Exercise 11.1. Here $\mu = (9 + 16)/2 = 12.5$, so $\mu + \varepsilon = 13$ and $s = 1/\sqrt{13}$. Then $\mathrm{rms}(\mathbf{x}) = (3, 4)/\sqrt{13} \approx (0.8321, 1.1094)$. With $\|\mathbf{x}\|^2 + n\varepsilon = 26$,

Equation 11.1 gives

$$J = \frac{1}{\sqrt{13}}\Big(I - \frac{1}{26}\begin{pmatrix} 9 & 12 \\ 12 & 16 \end{pmatrix}\Big) = \frac{1}{26\sqrt{13}}\begin{pmatrix} 17 & -12 \\ -12 & 10 \end{pmatrix}.$$

Then $J(3,4)^\top = \frac{1}{26\sqrt{13}}(3,4)^\top$, an eigenvalue $1/(26\sqrt{13})$, which equals $\varepsilon s^3 = \frac{1}{2}\cdot 13^{-3/2}$. And $J(4,-3)^\top = \frac{1}{26\sqrt{13}}(104,-78)^\top = \frac{1}{\sqrt{13}}(4,-3)^\top$, an eigenvalue s on the vector perpendicular to $\mathbf{x}$.

Exercise 11.2. J is symmetric and, by Theorem 11.1, $J\mathbf{x} = \varepsilon s^3\mathbf{x}$, so

$$\mathbf{x}\cdot\bar{\mathbf{x}} = \mathbf{x}\cdot J\bar{\mathbf{y}} = (J\mathbf{x})\cdot\bar{\mathbf{y}} = \varepsilon s^3\,(\mathbf{x}\cdot\bar{\mathbf{y}}).$$

With $\varepsilon = 0$ the right side is 0 for every $\bar{\mathbf{y}}$. This is what Theorem 11.2 predicts: for $\varepsilon = 0$ the map is unchanged by positive scaling, so any loss L computed from $\mathrm{rms}(\mathbf{x})$ satisfies $L(c\mathbf{x}) = L(\mathbf{x})$ for $c > 0$. Differentiating with respect to c at $c = 1$ by the chain rule gives $\nabla L(\mathbf{x})\cdot\mathbf{x} = 0$.

Exercise 11.3. $C\mathbf{x} = \mathbf{x} - \frac{1}{n}\mathbf{1}(\mathbf{1}^\top\mathbf{x}) = \mathbf{x} - m(\mathbf{x})\mathbf{1}$, because $\mathbf{1}^\top\mathbf{x} = \sum_i x_i$. The components of $C\mathbf{x}$ are $x_i - m(\mathbf{x})$, so $\mu(C\mathbf{x}) = \frac{1}{n}\sum_i (x_i - m(\mathbf{x}))^2 = v(\mathbf{x})$. Hence

$$\frac{x_i - m(\mathbf{x})}{\sqrt{v(\mathbf{x}) + \varepsilon}} = \frac{(C\mathbf{x})_i}{\sqrt{\mu(C\mathbf{x}) + \varepsilon}} = \mathrm{rms}(C\mathbf{x})_i.$$

The map C is linear, so its Jacobian is C (Example 1.6), and by Theorem 2.1 the Jacobian of layer normalisation is $J_{\mathrm{rms}}(C\mathbf{x})\,C$. Since $C\mathbf{1} = \mathbf{1} - \frac{1}{n}\mathbf{1}\cdot n = \mathbf{0}$, the Jacobian sends $\mathbf{1}$ to $\mathbf{0}$: adding the same small amount to every component does not change the output to first order (in fact it does not change it at all).

Exercise 11.4. (a) Write $\mathbf{y} = \mathrm{rms}(\mathbf{x}) = s\mathbf{x}$ with $s = (\mu + \varepsilon)^{-1/2}$. Then $\mu(\mathbf{y}) = s^2\mu = \mu/(\mu + \varepsilon)$, and $\mathrm{rms}(\mathbf{y}) = \mathbf{y}/\sqrt{\mu(\mathbf{y}) + \varepsilon} = c\,\mathbf{y}$.

(b) $c = 1$ exactly when $\mu/(\mu + \varepsilon) = 1 - \varepsilon$, that is, $\mu = (1 - \varepsilon)(\mu + \varepsilon)$. Expanding, this is $0 = \varepsilon(1 - \varepsilon - \mu)$, and since $\varepsilon > 0$ it holds exactly when $\mu = 1 - \varepsilon$.

(c) With $\mu = 1$, $c^{-2} = \frac{1}{1+\varepsilon} + \varepsilon = \frac{1+\varepsilon+\varepsilon^2}{1+\varepsilon}$, which gives c^2. Then $c^2 = 1 - \varepsilon^2/(1+\varepsilon+\varepsilon^2)$, and for $\varepsilon = 10^{-5}$, $1 - c \approx \varepsilon^2/2 = 5 \times 10^{-11}$.

Exercise 11.5. The embeddings contribute $|V|d$ (token) and $n_c d$ (position), and the output matrix another $|V|d$. Each layer has four $d \times d$ attention matrices, $4d^2$ entries, and two MLP matrices of shapes $4d \times d$ and $d \times 4d$, $8d^2$ entries, so $12d^2$ per layer. The total is $2|V|d + n_c d + 12Kd^2$. The heads only decide how to group the rows of W_Q, W_K and W_V, not the shapes of the matrices, so the count does not depend on them. For $|V| = 27$, $n_c = 16$, $d = 16$ and $K = 1$ the formula gives $864 + 256 + 3072 = 4192$, as in Theorem 11.4. With $d = 32$ and $K = 2$ it gives $1728 + 512 + 24576 = 26816$.

Exercise 11.6. Let $S = \sum_j e^{z_j}$, so that $p_i = e^{z_i}/S$ and $p_i^{1/T} = e^{z_i/T}/S^{1/T}$. Then

$$\frac{p_i^{1/T}}{\sum_j p_j^{1/T}} = \frac{e^{z_i/T}/S^{1/T}}{\sum_j e^{z_j/T}/S^{1/T}} = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}} = \operatorname{softmax}(\mathbf{z}/T)_i.$$

At $T = 0.5$ the powers are the squares $(0.25, 0.09, 0.04)$, with sum 0.38, so the sampling distribution is $(25, 9, 4)/38 \approx (0.658, 0.237, 0.105)$.

Exercise 11.7. The maximum 3 is attained at positions 2 and 3, so by Theorem 11.6 the limit as $T \to 0$ is $(0, \frac{1}{2}, \frac{1}{2}, 0)$, and the limit as $T \to \infty$ is $(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$. At $T = 1/\ln 2$, $e^{z_i/T} = e^{z_i \ln 2} = 2^{z_i}$, which gives $(2, 8, 8, 1)$ with sum 19, so the probabilities are $(2, 8, 8, 1)/19$.

Exercise 11.8. Write $J = J_F(\mathbf{a})$. Suppose $(I + J)\mathbf{v} = \mathbf{0}$. Then $\mathbf{v} = -J\mathbf{v}$, and by Lemma 1.1 $\|\mathbf{v}\| = \|J\mathbf{v}\| \le \|J\|_F \|\mathbf{v}\|$. Since $\|J\|_F < 1$, this forces $\|\mathbf{v}\| = 0$. So the only solution of $(I + J)\mathbf{v} = \mathbf{0}$ is $\mathbf{v} = \mathbf{0}$, and a square matrix with this property is invertible. For the example take $F(\mathbf{x}) = -x_1 \mathbf{e}_1$, which is linear with $J_F = -\mathbf{e}_1 \mathbf{e}_1^\top$, of Frobenius norm 1. Then $I + J_F$ has first column $\mathbf{0}$, so it is not invertible.

Exercise 11.9. At position t, each head computes t scores of d_h multiplications and a weighted sum of t value vectors of d_h components, so $2td_h$ multiplications, and the h heads give $2td_h h = 2td$. With the cache the model

processes each position once, for a total of $\sum_{t=1}^{n} 2td = d\,n(n+1)$. Without it, step t recomputes positions $1, \ldots, t$, at a cost of $\sum_{j=1}^{t} 2jd = d\,t(t+1)$, and the total is

$$d \sum_{t=1}^{n} t(t+1) = \frac{d\,n(n+1)(n+2)}{3},$$

where the sum formula holds for $n = 1$ ($2 = 1 \cdot 2 \cdot 3/3$) and passes from $n-1$ to n because $\frac{(n-1)n(n+1)}{3} + n(n+1) = \frac{n(n+1)(n+2)}{3}$. For $d = n = 16$ the two counts are 4352 and 26112, a ratio of $(n+2)/3 = 6$.

B.12. Appendix A

Exercise A.1. Since $2^9 = 512 \le 1000 < 1024 = 2^{10}$, the binary64 numbers near 1000 lie $2^{9-52} = 2^{-43} \approx 1.137 \times 10^{-13}$ apart. Adding 2^{-43} gives the next binary64 number, so the sum does not round back. The sum $1000 + 2^{-44}$ lies exactly halfway between 1000 and $1000 + 2^{-43}$. Writing $1000 = 2^9(1 + f/2^{52})$ gives $f = 488 \times 2^{43}$, which is even, so the tie goes to 1000. Any smaller power of 2 is nearer to 1000. The answer is 2^{-44}.

Exercise A.2. Every binary64 number has the form $M\,2^k$ with integers M and k. If $1/10 = M\,2^k$ with $k \ge 0$, the right side is an integer, which $1/10$ is not. If $k < 0$, then $2^{-k} = 10M$, so 5 divides a power of 2, which is impossible.

So 0.1, 0.2 and 0.3 are each stored with a rounding error, by Theorem A.1 of relative size at most u. The stored values are $\hat{a} \approx 0.1 + 5.55 \times 10^{-18}$ and $\hat{b} \approx 0.2 + 1.11 \times 10^{-17}$, and their exact sum $0.3 + 1.67 \times 10^{-17}$ lies exactly halfway between two consecutive binary64 numbers, which are 2^{-54} apart near 0.3. The tie goes to the one with even f, the upper one, $0.3 + 4.44 \times 10^{-17}$, which Python prints as `0.30000000000000004`. The stored 0.3 is the lower one, $0.3 - 1.11 \times 10^{-17}$. The two computed numbers are neighbours, so they are not equal.

Exercise A.3. Multiplying and dividing by $\sqrt{x+1}+\sqrt{x}$,

$$\sqrt{x+1}-\sqrt{x}=\frac{(x+1)-x}{\sqrt{x+1}+\sqrt{x}}=\frac{1}{\sqrt{x+1}+\sqrt{x}}.$$

In the first form, $a=\sqrt{x+1}$ and $b=\sqrt{x}$ are each computed with relative error at most u, and by Lemma A.1 the difference can have relative error up to $u(a+b)/(a-b)\approx u\cdot 2\sqrt{x}\cdot 2\sqrt{x}=4ux$, about 4.4×10^{-4} for $x=10^{12}$. The second form adds two positive numbers and divides, so by Equation A.1 each step adds a relative error of at most u and the result is accurate to a few units of u. In binary64 the first form gives 5.00003807×10^{-7} and the second $4.99999999999875\times10^{-7}$, which is correct to the digits shown; the first has relative error 7.6×10^{-6}, within the bound.

Exercise A.4. Let $m=\max_j z_j$ and $S=\sum_j e^{z_j-m}$, so that $\operatorname{lse}(\mathbf{z})=m+\ln S$ by Equation A.2. By Corollary A.1, $1\le S\le n$, so $m\le\operatorname{lse}(\mathbf{z})\le m+\ln n$. Equality on the right means $S=n$; since each of the n terms is at most 1, that happens exactly when every term is 1, that is, when $z_j=m$ for every j.

Exercise A.5. With $M_0=M_2=1$, Theorem A.2 (a) gives $h^*=2\sqrt{u}=2^{-25.5}\approx2.1\times10^{-8}$, and the bound $2\sqrt{u}\approx2.1\times10^{-8}$ on the error. For the central difference with $M_0=M_3=1$, $h^*=(3u)^{1/3}\approx6.9\times10^{-6}$ and the bound is $(3u)^{2/3}/2\approx2.4\times10^{-11}$. These assume that the library computes sin with error at most u; a library whose sin is accurate to within one unit in the last place doubles the rounding term at worst, which moves h^* by a factor of $\sqrt{2}$ or $2^{1/3}$ and changes the bounds by similar factors.