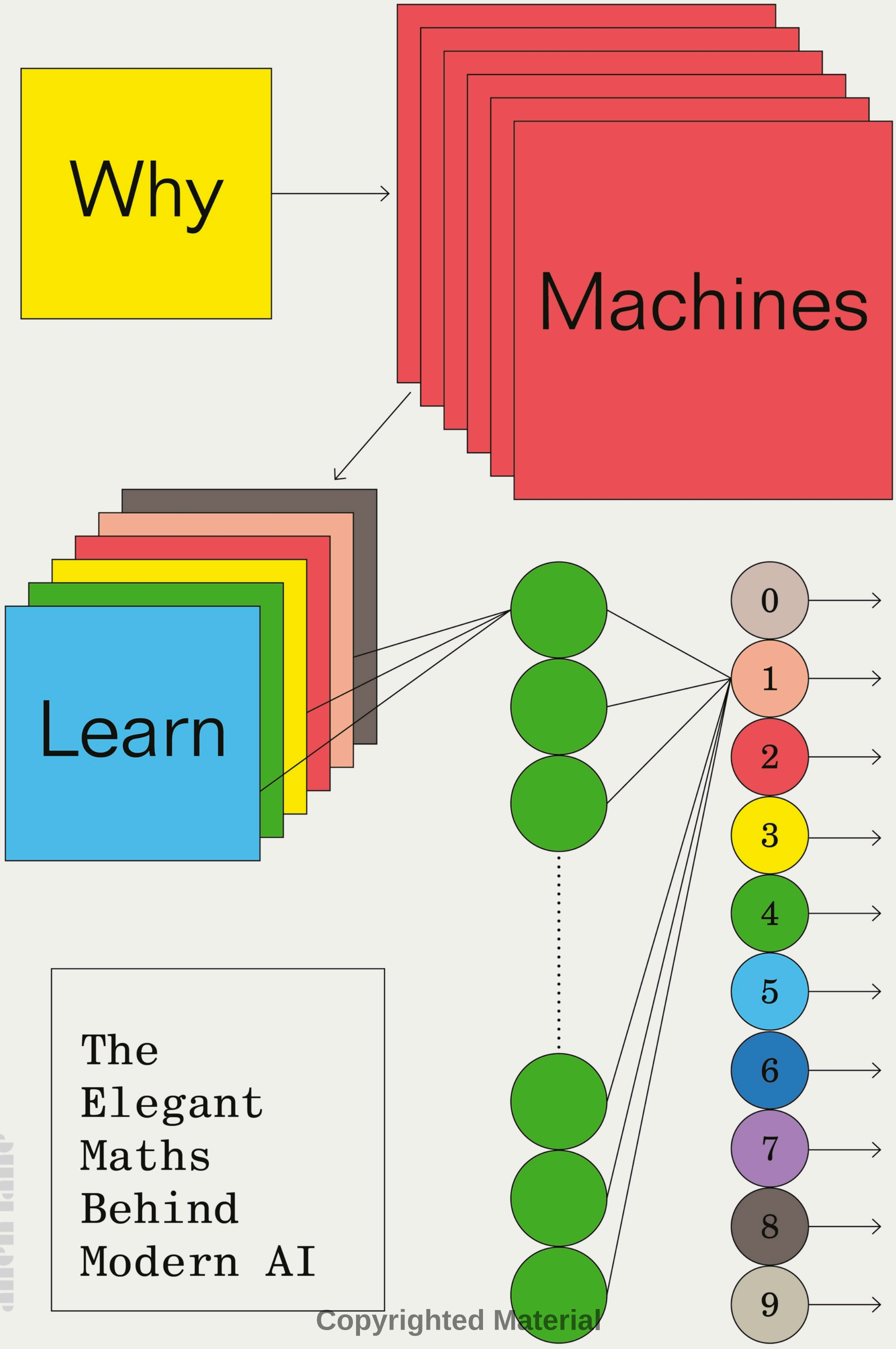
Why Machines Learn

The Elegant Maths Behind Modern AI



allen lane

Anil Ananthaswamy

Why Machines Learn

ANIL ANANTHASWAMY

Why Machines Learn

The Elegant Maths Behind Modern AI

ALLEN LANE
an imprint of
PENGUIN BOOKS

ALLEN LANE

UK | USA | Canada | Ireland | Australia
India | New Zealand | South Africa

Penguin Books is part of the Penguin Random House group of companies
whose addresses can be found at global.penguinrandomhouse.com.

Penguin
Random House
UK

First published in the United States of America by Dutton 2024
First published in Great Britain by Allen Lane 2024
001

Portions of chapter 12 and the epilogue appeared in *Quanta Magazine*.
The illustration in chapter 6 on PCA done on EEG data adapted with permission
from John Abel. The illustrations in chapter 12 on the bias-variance and double descent
curves adapted with permission from Mikhail Belkin. Illustrations about properties
of penguins in chapter 4 created courtesy of data made freely available by Kristen Gorman,
Allison Horst, and Alison Hill. The illustrations of biological neuron (p. 14),
paddy fields (p. 68), and the map of Manhattan (p. 147) by Roshan Shakeel.

Printed and bound in Great Britain by Clays Ltd, Elcograf S.p.A.

The authorized representative in the EEA is Penguin Random House Ireland,
Morrison Chambers, 32 Nassau Street, Dublin D02 YH68

A CIP catalogue record for this book is available from the British Library

ISBN: 978–0–241–58648–8

www.greenpenguin.co.uk

MIX
Paper | Supporting
responsible forestry
FSC® C018179
FSC
www.fsc.org

Penguin Random House is committed to a
sustainable future for our business, our readers
and our planet. This book is made from Forest
Stewardship Council® certified paper.

to teachers everywhere, sung and unsung

Whatever we do, we have to make our life vectors.
Lines with force and direction.

—LIAM NEESON AS FBI AGENT MARK FELT
IN THE 2017 MOVIE OF THE SAME NAME

The author acknowledges with gratitude the support
of the Alfred P. Sloan Foundation in the
research and writing of this book.

CONTENTS

Prologue

Buried on page 25 of the July 8, 1958, issue of *The New York Times* was a rather extraordinary story. The headline read, "New Navy Device Learns by Doing: Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser." The opening paragraph raised the stakes: "The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

With hindsight, the hyperbole is obvious and embarrassing. But *The New York Times* wasn't entirely at fault. Some of the over-the-top talk also came from Frank Rosenblatt, a Cornell University psychologist and project engineer. Rosenblatt, with funding from the U.S. Office of Naval Research, had invented the perceptron, a version of which was presented at a press conference the day before the *New York Times* story about it appeared in print. According to Rosenblatt, the perceptron would be the "first device to think as the human brain" and such machines might even be sent to other planets as "mechanical space explorers."

None of this happened. The perceptron never lived up to the hype. Nonetheless, Rosenblatt's work was seminal. Almost every lecturer on artificial intelligence (AI) today will harken back to the perceptron. And that's justified. This moment in history—the arrival of large language models (LLMs) such as ChatGPT and its ilk and our response to it—which some have likened to what it must have felt like in the 1910s and '20s, when physicists were confronted with the craziness of quantum mechanics, has its roots in research initiated by Rosenblatt. There's a line in the *New York Times* story that only hints at the revolution the perceptron set in motion: "Dr. Rosenblatt said he could explain *why the machine learned* only in highly technical terms" (italics mine). The story, however, had none of the "highly technical" details.

This book does. It tackles the technical details. It explains the elegant mathematics and algorithms that have, for decades, energized and excited researchers in "machine learning," a type of AI that involves building machines that can learn to discern patterns in data without being explicitly programmed to do so. Trained machines can then detect similar patterns in new, previously unseen data, making possible applications that range from recognizing pictures of cats and dogs to creating, potentially, autonomous cars and other technology. Machines can learn because of the extraordinary confluence of math and computer science, with more than a dash of physics and neuroscience added to the mix.

Machine learning (ML) is a vast field populated by algorithms that leverage relatively simple math that goes back centuries, math one learns in high school or early in college. There's, of course, elementary algebra. Another extremely important cornerstone of machine learning is calculus, co-invented by no less a polymath than Isaac Newton. The field also relies heavily on the work of Thomas Bayes, the eighteenth-century English statistician and minister who gave us the eponymous Bayes's theorem, a key contribution to the field of probability and statistics. The work of German mathematician Carl Friedrich Gauss on the Gaussian distribution (and the bell-shaped curve) also permeates machine learning. Then there's linear algebra, which forms the backbone of machine learning. The earliest exposition of this branch of mathematics appears in a two-thousand-year-old Chinese text, *Nine Chapters on the Mathematical Art.* The modern version of linear algebra has its roots in the work of many mathematicians, but mainly Gauss, Gottfried Wilhelm Leibniz, Wilhelm Jordan, Gabriel Cramer, Hermann Günther Grassmann, James Joseph Sylvester, and Arthur Cayley.

By the mid-1850s, some of the basic math that would prove necessary to building learning machines was in place, even as other mathematicians continued developing more relevant mathematics and birthed and advanced the field of computer science. Yet, few could have dreamed that such early mathematical work would be the basis for the astounding developments in AI over the past half century, particularly over the last decade, some of which may legitimately allow us to envision a semblance of the kind of future Rosenblatt was overoptimistically foreshadowing in the 1950s.

This book tells the story of this journey, from Rosenblatt's perceptron to modern-day deep neural networks, elaborate networks of computational units called artificial neurons, through the lens of key mathematical ideas underpinning the field of machine learning. It eases gently into the math and then, ever so slowly, ratchets up the difficulty, as we go from the relatively simple ideas of the 1950s to the somewhat more involved math and algorithms that power today's machine learning systems.

Hence, we will unabashedly embrace equations and concepts from at least four major fields of mathematics—linear algebra, calculus, probability and statistics, and optimization theory—to acquire the minimum theoretical and conceptual knowledge necessary to appreciate the awesome power we are bestowing on machines. It is only when we understand the inevitability of learning machines that we will be prepared to tackle a future in which AI is ubiquitous, for good and for bad.

Getting under the mathematical skin of machine learning is crucial to our understanding of not just the power of the technology, but also its limitations. Machine learning systems are already making life-altering decisions for us: approving credit card applications and mortgage loans, determining whether a tumor is cancerous, predicting the prognosis for someone in cognitive decline (will they go on to get Alzheimer's?), and deciding whether to grant someone bail. Machine learning has permeated science, too: It is influencing chemistry, biology, physics, and everything in between. It's being used in the study of genomes, extrasolar planets, the intricacies of quantum systems, and much more. And as of this writing, the world of AI is abuzz with the advent of large language models such as ChatGPT. The ball has only just gotten rolling.

We cannot leave decisions about how AI will be built and deployed solely to its practitioners. If we are to effectively regulate this extremely useful, but disruptive and potentially threatening, technology, another layer of society—educators, politicians, policymakers, science communicators, or even interested consumers of AI—must come to grips with the basics of the mathematics of machine learning.

In her book *Is Math Real?,* mathematician Eugenia Cheng writes about the gradual process of learning mathematics: "It can . . . seem like we're taking very small steps and not getting anywhere, before suddenly we look behind us and discover we've climbed a giant mountain. All these things can be disconcerting, but accepting a little intellectual discomfort (or sometimes a lot of it) is an important part of making progress in math."

Fortunately, the "intellectual discomfort" in store for us is eminently endurable and more than assuaged by the intellectual payoff, because underlying modern ML is some relatively simple and elegant math—a notion that's best illustrated with an anecdote about Ilya

Sutskever. Today, Sutskever is best known as the co-founder of OpenAI, the company behind ChatGPT. More than a decade ago, as a young undergraduate student looking for an academic advisor at the University of Toronto, Sutskever knocked on Geoffrey Hinton's door. Hinton was already a well-known name in the field of "deep learning," a form of machine learning, and Sutskever wanted to work with him. Hinton gave Sutskever some papers to read, which he devoured. He remembers being perplexed by the simplicity of the math, compared to the math and physics of his regular undergrad coursework. He could read these papers on deep learning and understand powerful concepts. "How can it be that it's so simple . . . so simple that you can explain it to high school students without too much effort?" he told me. "I think that's actually miraculous. This is also, to me, an indication that we are probably on the right track. [It can't] be a coincidence that such simple concepts go so far."

Of course, Sutskever already had sophisticated mathematical chops, so what seemed simple to him may not be so for most of us, including me. But let's see.

This book aims to communicate the conceptual simplicity underlying ML and deep learning. This is not to say that everything we are witnessing in AI now—in particular, the behavior of deep neural networks and large language models—is amenable to being analyzed using simple math. In fact, the denouement of this book leads us to a place that some might find disconcerting, though others will find it exhilarating: These networks and AIs seem to flout some of the fundamental ideas that have, for decades, underpinned machine learning. It's as if empirical evidence has broken the theoretical camel's back in the same way experimental observations of the material world in the early twentieth century broke classical physics; we need something new to make sense of the brave new world awaiting us.

As I did the research for this book, I observed a pattern to my learning that reminded me of the way modern artificial neural networks learn: With each pass the algorithm makes through data, it learns more about the patterns that exist in that data. One pass may not be enough; nor ten; nor a hundred. Sometimes, neural networks learn over tens of thousands of iterations through the data. This is indeed the way I grokked the subject in order to write about it. Each pass through some corner of this vast base of knowledge caused some neurons in my brain to make connections, literally and metaphorically. Things that didn't make sense the first or second time around eventually did upon later passes.

I have used this technique to help readers make similar connections: I found myself repeating ideas and concepts over the course of writing this book, sometimes using the same phrasing or, at times, a different take on the same concept. These repetitions and rephrasings are intentional: They are one way that most of us who are not mathematicians or practitioners of ML can come to grips with a paradoxically simple yet complex subject. Once an idea is exposed, our brains might see patterns and make connections when encountering that idea elsewhere, making more sense of it than would have been possible at first blush.

I hope your neurons enjoy this process as much as mine did.

CHAPTER 1

Desperately Seeking Patterns

When he was a child, the Austrian scientist Konrad Lorenz, enamored by tales from a book called *The Wonderful Adventures of Nils*—the story of a boy's adventures with wild geese written by the Swedish novelist and winner of the Nobel Prize for Literature, Selma Lagerlöf—"yearned to become a wild goose." Unable to indulge his fantasy, the young Lorenz settled for taking care of a day-old duckling his neighbor gave him. To the boy's delight, the duckling began following him around: It had imprinted on him. "Imprinting" refers to the ability of many animals, including baby ducks and geese (goslings), to form bonds with the first moving thing they see upon hatching. Lorenz would go on to become an ethologist and would pioneer studies in the field of animal behavior, particularly imprinting. (He got ducklings to imprint on him; they followed him around as he walked, ran, swam, and even paddled away in a canoe.) He won the Nobel Prize for Physiology or Medicine in 1973, jointly with fellow ethologists Karl von Frisch and Nikolaas Tinbergen. The three were celebrated "for their discoveries concerning organization and elicitation of individual and social behavior *patterns*."

Patterns. While the ethologists were discerning them in the behavior of animals, the animals were detecting patterns of their own. Newly hatched ducklings must have the ability to make out or tell apart the properties of things they see moving around them. It turns out that ducklings can imprint not just on the first living creature they see moving, but on inanimate things as well. Mallard ducklings, for example, can imprint on a pair of moving objects that are similar in shape or color. Specifically, they imprint on the relational concept embodied by the objects. So, if upon birth the ducklings see two moving red objects, they will later follow two objects of the same color (even if those latter objects are blue, not red), but not two objects of different colors. In this case, the ducklings imprint on the *idea* of similarity. They also show the ability to discern *dis*similarity. If the first moving objects the ducklings see are, for example, a cube and a rectangular prism, they will recognize that the objects have different shapes and will later follow two objects that are different in shape (a pyramid and a cone, for example), but they will ignore two objects that have the same shape.

Ponder this for a moment. Newborn ducklings, with the briefest of exposure to sensory stimuli, detect patterns in what they see, form abstract notions of similarity/dissimilarity, and then will recognize those abstractions in stimuli they see later and act upon them. Artificial intelligence researchers would offer an arm and a leg to know just how the ducklings pull this off.

While today's AI is far from being able to perform such tasks with the ease and efficiency of ducklings, it does have something in common with the ducklings, and that's the ability to pick out and learn about patterns in data. When Frank Rosenblatt invented the perceptron in the late 1950s, one reason it made such a splash was because it was the first formidable "brain-inspired" algorithm that could learn about patterns in data simply by examining the data. Most important, given certain assumptions about the data, researchers proved that Rosenblatt's perceptron will always find the pattern hidden in the data in a finite amount of time; or, put differently, the perceptron will converge upon a solution without fail. Such certainties in computing are like gold dust. No wonder the perceptron learning algorithm created such a fuss.

But what do these terms mean? What are "patterns" in data? What does "learning about these patterns" imply? Let's start by examining this table:

x1	**x2**	**y**
4	2	8
1	2	5
0	5	10
2	1	4

Each row in the table is a triplet of values for variables *x1, x2,* and *y.* There's a simple pattern hidden in this data: In each row, the value of *y* is related to the corresponding values of *x1* and *x2.* See if you can spot it before reading further.

In this case, with a pencil, paper, and a little effort one can figure out that *y* equals *x1* plus two times *x2*.

$$y = x1 + 2x2$$

A small point about notation: We are going to dispense with the multiplication sign ("×") between two variables or between a constant and a variable. For example, we'll write

$$2 \times x2 \text{ as } 2x2 \text{ and } x1 \times x2 \text{ as } x1x2$$

Ideally, we should write $2x2$ as $2x_2$ and $x1x2$ as x_1x_2, with the variables subscripted. But we'll dispense with the subscripts, too, unless it becomes absolutely necessary to use them. (Purists will cringe, but this method helps keep our text less cluttered and easy on the eye;

when we do encounter subscripts, read x_i as "x sub-i.") So, keep this in mind: If there's a symbol such as "x" followed by a digit such as "2," giving us $x2$, take the entire symbol to mean one thing. If a symbol (say, x or $x2$) is preceded by a number (say, 9), or by another symbol (say, $w1$), then the number and the symbol, or the two symbols, are being multiplied. So:

$$2x2 = 2 \times x2$$

$$x1x2 = x1 \times x2$$

$$w2x1 = w2 \times x1$$

Getting back to our equation $y = x1 + 2x2$, more generally, we can write this as:

$$y = w1x1 + w2x2, \text{ where } w1 = 1 \text{ and } w2 = 2$$

To be clear, we have found one of the many possible relationships between y and $x1$ and $x2$. There can be others. And indeed, for this example, there are, but we don't need to worry about them for our purposes here. Finding patterns is nowhere near as simple as this example is suggesting, but it gets us going.

We identified what's called a linear relationship between y, on the one hand, and $x1$ and $x2$, on the other. ("Linear" means that y depends only on $x1$ and $x2$, and not on $x1$ or $x2$ raised to some power, or on any product of $x1$ and $x2$.) Also, I'm using the words "equation" and "relationship" interchangeably here.

The relationship between y, $x1$, and $x2$ is defined by the constants $w1$ and $w2$. These constants are called the coefficients, or weights, of the linear equation connecting y to $x1$ and $x2$. In this simple case, assuming such a linear relationship exists, we figured out the values for *w1* and *w2* after inspecting the data. But often, the relationship between *y* and (*x1, x2, . . .*) is not so straightforward, especially when it extends to more values on the right side of the equation.

For example, consider:

$$y = w1x1 + w2x2 + w3x3 + \cdots + w9x9$$

Or, more generally, for a set of *n* weights, and using formal mathematical notation:

$$y = w1x1 + w2x2 + w3x3 + \cdots + wnxn = \sum_{i=1}^{n} wixi$$

The expression on the right, using the sigma notation, is shorthand for summing all *wixi,* where *i* takes on values from 1 to *n.*

In the case of 9 inputs, you'd be hard-pressed to extract the values of *w1* to *w9* just by visually inspecting the data and doing some mental arithmetic. That's where learning comes in. If there's a way to algorithmically figure out the weights, then the algorithm is "learning" the weights. But what's the point of doing that?

Well, once you have learned the weights—say, *w1* and *w2* in our simple, toy example—then given some value of *x1* and *x2* that wasn't in our initial dataset, we can calculate the value of *y.* Say, $x1 = 5$ and $x2 = 2$. Plug these values into the equation $y = x1 + 2x2$ and you get a value of $y = 9$.

What's all this got to do with real life? Take a very simple, practical, and some would say utterly boring problem. Let's say *x1* represents the number of bedrooms in a house, and *x2* represents the total square footage, and *y* represents the price of the house. Let's assume that there exists a linear relationship between (*x1, x2*) and *y.* Then, by learning the weights of the linear equation from some existing data about houses and their prices, we have essentially built a very simple model with which to predict the price of a house, given the number of bedrooms and the square footage.

The above example—a teeny, tiny baby step, really—is the beginning of machine learning. What we just did is a simplistic form of something called supervised learning. We were given samples of data that had hidden in them some correlation between a set of inputs and a set of outputs. Such data are said to be annotated, or labeled; they are also called the training data. Each input (*x1,* x2, . . . , *xn*) has a label *y* attached to it. So, in our earlier numerical table, the pair of numbers (4, 2) is labeled with $y = 8$, the pair (1, 2) with 5, and so on. We figured out the correlation. Once it is learned, we can use it to make predictions about new inputs that weren't part of the training data.

Also, we did a very particular kind of problem solving called regression, where given some independent variables (*x1, x2*), we built a model (or equation) to predict the value of a dependent variable (*y*). There are many other types of models we could have built, and we'll come to them in due course.

In this case, the correlation, or pattern, was so simple that we needed only a small amount of labeled data. But modern ML requires orders of magnitude more—and the availability of such data has been one of the factors fueling the AI revolution. (The ducklings, for their part, likely indulge in a more sophisticated form of learning. No parent duck sits around labeling the data for its ducklings, and yet the babies learn. How do they do it? Spoiler alert: We don't know, but maybe by understanding *why* machines learn, we can one day fully understand how ducklings and, indeed, humans learn.)

It may seem implausible, but this first step we took using a laughably simple example of supervised learning sets us on a path toward understanding modern deep neural networks—one step at a time, of course (with small, gentle, and occasionally maybe not so gentle dollops of vectors, matrices, linear algebra, calculus, probability and statistics, and optimization theory served, as needed, along the way).

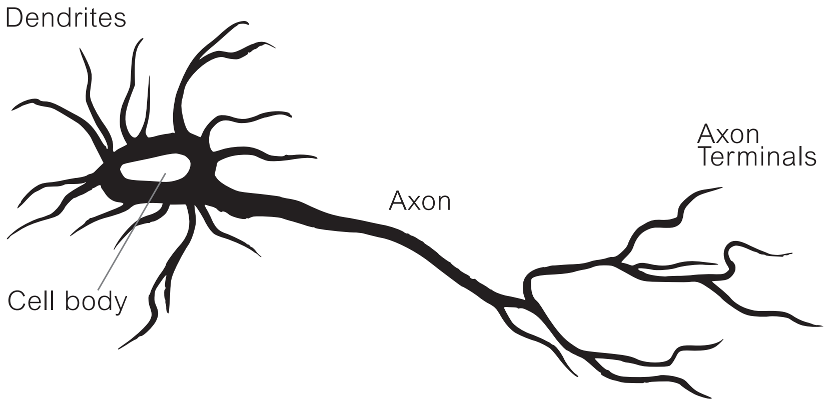
Rosenblatt's perceptron, which we briefly encountered in the prologue, was for its time an astonishing example of one such learning algorithm. And because it was modeled on how neuroscientists thought human neurons worked, it came imbued with mystique and the promise that, one day, perceptrons would indeed make good on the promise of AI.

THE FIRST ARTIFICIAL NEURON

The perceptron's roots lie in a 1943 paper by an unlikely combination of a philosophically minded neuroscientist in his mid-forties and a homeless teenager. Warren McCulloch was an American neurophysiologist trained in philosophy, psychology, and medicine. During the 1930s, he worked on neuroanatomy, creating maps of the connectivity of parts of monkey brains. While doing so, he also obsessed over the "logic of the brain." By then, the work of mathematicians and philosophers like Alan Turing, Alfred North Whitehead, and Bertrand Russell was suggesting a deep connection between computation and logic. The statement "If P is true AND Q is true, then S is true" is an example of a logical proposition. The assertion was that all computation could be reduced to such logic. Given this way of thinking about computation, the question bothering McCulloch was this: If the brain is a computational device, as many think it is, how does it implement such logic?

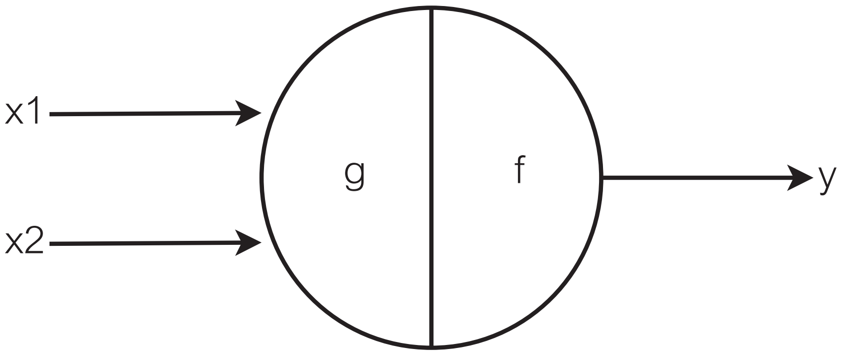
With these questions in mind, McCulloch moved in 1941 from Yale University to the University of Illinois, where he met a prodigiously talented teenager named Walter Pitts. The youngster, already an accomplished logician ("a protégé of the eminent mathematical logician Rudolf Carnap"), was attending seminars run by Ukrainian mathematical physicist Nicolas Rashevsky in Chicago. Pitts, however, was a "mixed-up adolescent, essentially a runaway from a family that could not appreciate his genius." McCulloch and his wife, Rook, gave Walter a home. "There followed endless evenings sitting around the McCulloch kitchen table trying to sort out how the brain worked, with the McCullochs' daughter Taffy sketching little pictures," wrote computer scientist Michael Arbib. Taffy's drawings would later illustrate McCulloch and Pitts's 1943 paper, "A Logical Calculus of the Ideas Immanent in Nervous Activity."

In that work, McCulloch and Pitts proposed a simple model of a biological neuron. First, here's an illustration of a generic biological neuron:



The neuron's cell body receives inputs via its treelike projections, called dendrites. The cell body performs some computation on these inputs. Then, based on the results of that computation, it may send an electrical signal spiking along another, longer projection, called the axon. That signal travels along the axon and reaches its branching terminals, where it's communicated to the dendrites of neighboring neurons. And so it goes. Neurons interconnected in this manner form a biological neural network.

McCulloch and Pitts turned this into a simple computational model, an artificial neuron. They showed how by using one such artificial neuron, or neurode (for "neuron" + "node"), one could implement certain basic Boolean logical operations such as AND, OR, NOT, and so on, which are the building blocks of digital computation. (For some Boolean operations, such as exclusive-OR, or XOR, you need more than one neurode, but more on this later.) What follows is an image of a single neurode. (Ignore the "*g*" and "*f*" inside the neuron for now; we'll come to those in a moment.)



In this simple version of the McCulloch-Pitts model, *x1* and *x2* can be either 0 or 1. In formal notation, we can say:

$$x1, x2 \in \{0,1\}$$

That should be read as *x1* is an element of the set {0, 1} and *x2* is an element of the set {0, 1}; *x1* and *x2* can take on only values 0 or 1 and nothing else. The neurode's output *y* is calculated by first summing the inputs and then checking to see if that sum is greater than or equal to some threshold, *theta* (θ). If so, *y* equals 1; if not, *y* equals 0.

$$sum = x1 + x2$$

$$If\ sum \geq \theta\text{: } y = 1$$

$$Else\text{: } y = 0$$

Generalizing this to an arbitrary sequence of inputs, *x1, x2, x3, . . . , xn,* one can write down the formal mathematical description of the simple neurode. First, we define the function $g(x)$—read that as "g of x," where x here is the set of inputs $(x1, x2, x3, \ldots, xn)$—which sums up the inputs. Then we define the function $f(g(x))$—again, read that as "f of g of x"—which takes the summation and performs the thresholding to generate the output, *y:* It is zero if $g(x)$ is less than some θ and 1 if $g(x)$ is greater than or equal to θ.

$$g(x) = x1 + x2 + x3 + \cdots + xn = \sum_{i=1}^{n} xi$$

$$f(z) = \begin{cases} 0, & z < \theta \\ 1, & z \geq \theta \end{cases}$$

$$y = f(g(x)) = \begin{cases} 0, & g(x) < \theta \\ 1, & g(x) \geq \theta \end{cases}$$

With one artificial neuron as described, we can design some of the basic Boolean logic gates (AND & OR, for example). In an AND logic gate, the output *y* should be 1 if both *x1* and *x2* are equal to 1; otherwise, the output should be 0. In this case, $\theta = 2$ does the trick. Now, the output *y* will be 1 only when *x1* and *x2* are both 1 (only then will $x1 + x2$ be greater than or equal to 2). You can play with the value of θ to design the other logic gates. For example, in an OR gate, the output should be 1 if either *x1* or *x2* is 1; otherwise, the output should be 0. What should θ be?

The simple MCP model can be extended. You can increase the number of inputs. You can let inputs be "inhibitory," meaning *x1* or *x2* can be multiplied by −1. If one of the inputs to the neurode is inhibitory and you set the threshold appropriately, then the neurode will always output a 0, regardless of the value of all the other inputs. This allows you to build more complex logic. As does interconnecting multiple neurodes such that the output of one neurode serves as the input to another.

All this was amazing, and yet limited. The McCulloch-Pitts (MCP) neuron is a unit of computation, and you can use combinations of it to create any type of Boolean logic. Given that all digital computation at its most basic is a sequence of such logical operations, you can essentially mix and match MCP neurons to carry out any computation. This was an extraordinary statement to make in 1943. The mathematical roots of McCulloch and Pitts's paper were apparent. The paper had only three references—Carnap's *The Logical Syntax of Language;* David Hilbert and Wilhelm Ackermann's *Foundations of Theoretical Logic;* and Whitehead and Russell's *Principia Mathematica*—and none of them had to do with biology. There was no doubting the rigorous results derived in the McCulloch-Pitts paper. And yet, the upshot was simply a machine that could compute, not learn. In particular, the value of θ had to be hand-engineered; the neuron couldn't examine the data and figure out θ.

It's no wonder Rosenblatt's perceptron made such a splash. It could learn its weights from data. The weights encoded some knowledge, however minimal, about patterns in the data and remembered them, in a manner of speaking.

LEARNING FROM MISTAKES

Rosenblatt's scholarship often left his students floored. George Nagy, who came to Cornell University in Ithaca, New York, in 1960 to do his Ph.D. with Rosenblatt, recalled a walk the two of them took, during which they talked about stereo vision. Rosenblatt blew Nagy away with his mastery of the topic. "It was difficult not to feel naïve talking to him in general," said Nagy, now professor emeritus at Rensselaer Polytechnic Institute in Troy, New York; Rosenblatt's evident erudition was accentuated by his relative youth. (He was barely ten years older than Nagy.)

Rosenblatt's youthfulness almost got the two of them into trouble during a road trip. He and Nagy had to go from Ithaca to Chicago for a conference. Rosenblatt hadn't yet written the paper he wanted to present, so he asked Nagy to drive while he worked. Nagy had never owned a car and barely knew how to drive, but he agreed nonetheless. "Unfortunately, I drove in several lanes at once, and a policeman stopped us," Nagy said. Rosenblatt told the cop that he was a professor and had asked his student to drive. "The cop laughed and said, 'You are not a professor, you are a student.'" Fortunately, Rosenblatt had enough papers on him to convince the cop of his credentials, and the cop let the two go. Rosenblatt drove the rest of the way to Chicago, where he stayed up all night typing his paper, which he presented the next day. "He was able to do these things," Nagy told me.

By the time Nagy arrived at Cornell, Rosenblatt had already built the Mark I Perceptron; we saw in the prologue that Rosenblatt had done so in 1958, leading to the coverage in *The New York Times*. Nagy began working on the next machine, called Tobermory (named after the talking cat created by H. H. Munro, aka Saki), a hardware neural network designed for speech recognition. Mean-while, the Mark I Perceptron and Rosenblatt's ideas had already garnered plenty of attention.

In the summer of 1958, the editor of the Cornell Aeronautical Laboratory's *Research Trends* magazine had devoted an entire issue to Rosenblatt ("because of the unusual significance of Dr. Rosenblatt's article," according to the editor). The article was titled "The Design of an Intelligent Automaton: Introducing the Perceptron—A Machine that Senses, Recognizes, Remembers, and Responds Like the Human Mind." Rosenblatt would eventually rue choosing the term "perceptron" to describe his work. "It became one of Rosenblatt's great regrets that he used a word that sounds like a machine," Nagy told me. By "perceptron," Rosenblatt really meant a class of models of the nervous system for perception and cognition.

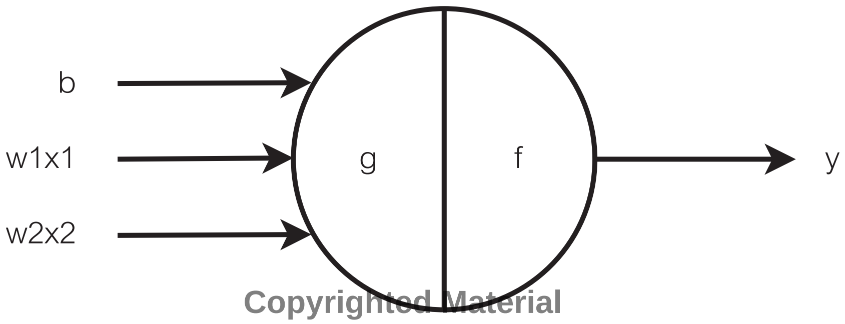
His emphasis on the brain wasn't a surprise. Rosenblatt had studied with James Gibson, one of the giants in the field of visual perception. He also looked up to McCulloch and Pitts and to Donald Hebb, a Canadian psychologist who in 1949 introduced a model for how biological neurons learn—to be clear, "learning" here refers to learning about patterns in data and not to the kind of learning we usually associate with high-level human cognition. "He'd always talk highly of them," Nagy said.

While McCulloch and Pitts had developed models of the neuron, networks of these artificial neurons could not learn. In the context of biological neurons, Hebb had proposed a mechanism for learning that is often succinctly, but somewhat erroneously, put as "Neurons that fire together wire together." More precisely, according to this way of thinking, our brains learn because connections between neurons strengthen when one neuron's output is consistently involved in the firing of another, and they weaken when this is not so. The process is called Hebbian learning. It was Rosenblatt who took the work of these pioneers and synthesized it into a new idea:

artificial neurons that reconfigure as they learn, embodying information in the strengths of their connections.

As a psychologist, Rosenblatt didn't have access to the kind of computer power he needed to simulate his ideas in hardware or software. So, he borrowed time on the Cornell Aeronautical Laboratory's IBM 704, a five-ton, room-size behemoth. The collaboration proved fruitful when Rosenblatt's work caught the attention of physicists, resulting in papers in journals of psychology and of the American Physical Society. Rosenblatt eventually built the Mark I Perceptron. The device had a camera that produced a 20x20-pixel image. The Mark I, when shown these images, could recognize letters of the alphabet. But saying that the Mark I "recognized" characters is missing the point, Nagy said. After all, optical character recognition systems, which had the same abilities, were commercially available by the mid-1950s. "The point is that Mark I *learned* to recognize letters by being zapped when it made a mistake!" Nagy would say in his talks.

But what exactly is a perceptron, and how does it learn? In its simplest form, a perceptron is an augmented McCulloch-Pitts neuron imbued with a learning algorithm. What follows is an example with two inputs. Note that each input is being multiplied by its corresponding weight. (There is also an extra input, *b,* the reason for which will soon become clear.)



The computation carried out by the perceptron goes like this:

$$sum = w1x1 + w2x2 + b$$

$$If\ sum > 0\text{: } y = 1$$

$$Else\text{: } y = -1$$

More generally and in mathematical notation:

$$g(x) = w1x1 + w2x2 + \cdots + wnxn + b = \sum_{i=1}^{n} wixi + b$$

$$f(z) = \begin{cases} -1, z \leq 0 \\ 1, z > 0 \end{cases}$$

$$y = f(g(x)) = \begin{cases} -1, g(x) \leq 0 \\ 1, g(x) > 0 \end{cases}$$

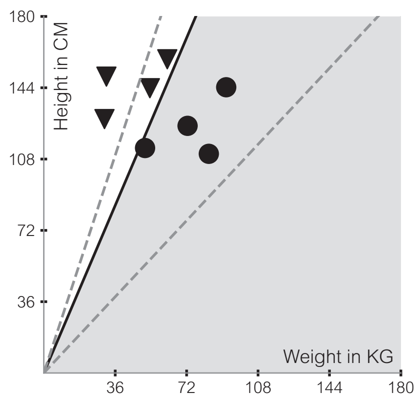
The main difference from the MCP model presented earlier is that the perceptron's inputs don't have to be binary (0 or 1), but can take on any value. Also, these inputs are multiplied by their corresponding weights, so we now have a weighted sum. Added to that is an additional term *b,* the bias. The output, *y,* is either −1 or +1 (instead of 0 or 1, as in the MCP neuron). Crucially, unlike with the MCP neuron, the perceptron can learn the correct value for the weights and the bias for solving some problem.

To understand how this works, consider a perceptron that seeks to classify someone as obese, $y = +1$, or not-obese, $y = -1$. The inputs are a person's body weight, *x1,* and height, *x2.* Let's say that the dataset contains a hundred entries, with each entry comprising a person's body weight and height and a label saying whether a doctor thinks the person is obese according to guidelines set by the National Heart, Lung, and Blood Institute. A perceptron's task is to learn the values for *w1* and *w2* and the value of the bias term *b,* such that it correctly classifies each person in the dataset as "obese" or "not-obese." Note: We are analyzing a person's body weight and height while also talking about the perceptron's weights (*w1* and *w2*); keep in mind these two different meanings of the word "weight" while reading further.

Once the perceptron has learned the correct values for *w1* and *w2* and the bias term, it's ready to make predictions. Given another person's body weight and height—this person was not in the original dataset, so it's not a simple matter of consulting a table of entries—the perceptron can classify the person as obese or not-obese. Of course, a few assumptions underlie this model, many of them to do with probability distributions, which we'll come to in subsequent chapters. But the perceptron makes one basic assumption: It assumes that there exists a clear, linear divide between the categories of people classified as obese and those classified as not-obese.

In the context of this simple example, if you were to plot the body weights and heights of people on an xy graph, with weights on the x-axis and heights on the y-axis, such that each person was a point on the graph, then the "clear divide" assumption states that there would exist a straight line separating the points representing the obese from the points representing the not-obese. If so, the dataset is said to be linearly separable.

Here's a graphical look at what happens as the perceptron learns. We start with two sets of data points, one characterized by black circles ($y = +1$, obese) and another by black triangles ($y = -1$, not-obese). Each data point is characterized by a pair of values (*x1, x2*), where *x1* is the body weight of the person in kilograms, plotted along the x-axis, and *x2* is the height in centimeters, plotted along the y-axis.



The perceptron starts with its weights, *w1* and *w2,* and the bias initialized to zero. The weights and bias represent a line in the xy plane. The perceptron then tries to find a separating line, defined by some set of values for its weights and bias, that attempts to classify the points. In the beginning, it classifies some points correctly and others incorrectly. Two of the incorrect attempts are shown as the gray dashed lines. In this case, you can see that in one attempt, all the points lie to one side of the dashed line, so the triangles are classified correctly, but the circles are not; and in another attempt, it gets the circles correct but some of the triangles wrong. The perceptron learns from its mistakes and adjusts its weights and bias. After numerous passes through the data, the perceptron eventually discovers at least one set of correct values of its weights and its bias term. It finds a line that delineates the clusters: The circles and the triangles lie on opposite sides. This is shown as a solid black line separating the coordinate space into two regions (one of which is shaded gray). The weights learned by the perceptron dictate the slope of the line; the bias determines the distance, or offset, of the line from the origin.

Once the perceptron has learned the correlation between the physical characteristics of a person (body weight and height) and whether that person is obese ($y = +1$ or -1), you can give it the body weight and height of a person whose data weren't used during training, and the perceptron can tell you whether that person should be classified as obese. Of course, now the perceptron is making its best prediction, having learned its weights and bias, but the prediction can be wrong. Can you figure out why? See if you can spot the problem just by looking at the graph. (Hint: How many different lines can you draw that succeed in separating the circles from the triangles?) As we'll see, much of machine learning comes down to minimizing prediction error.

What's described above is a single perceptron unit, or one artificial neuron. It seems simple, and you may wonder what all the fuss is about. Well, imagine if the number of inputs to the perceptron went beyond two: (*x1, x2, x3, x4,* and so on), with each input (*xi*) getting its own axis. You can no longer do simple mental arithmetic and solve the problem. A line is no longer sufficient to separate the two clusters, which now exist in much higher dimensions than just two. For example, when you have three points (*x1, x2, x3*), the data is three-dimensional: you need a 2D plane to separate the data points. In dimensions of four or more, you need a hyperplane (which we cannot visualize with our 3D minds). In general, this higher-dimensional equivalent of a 1D straight line or a 2D plane is called a hyperplane.

Now think back to 1958. Rosenblatt built his Mark I Perceptron with numerous such units. It could process a 20x20-pixel image—for a total of 400 pixels, with each pixel corresponding to an *x* input value. So, the Mark I took as input a long row of values: *x1, x2, x3, . . . , x400.* A complex arrangement of artificial neurons, both with fixed, random weights and weights that could be learned, turned this vector of 400 values into an output signal that could be used to discern the pattern in the image. (This is an oversimplified description. Some of the computing was complex enough that it needed an IBM 704. We'll get a glimpse of the architectural details in chapter 10.) The Mark I could learn to categorize the letters of the alphabet encoded in those pixel values. All the logic just described, scaled up to handle 400 inputs, was built-in hardware. The machine, once it had learned (we'll see how in the next chapter), contained knowledge in the strengths (weights) of its connections. It's little wonder that everyone let their imagination run wild.

But if you closely examine what the perceptron learns, its limitations—in hindsight, of course—become obvious. The algorithm is helping the perceptron learn about correlations between values of (*x1, x2, . . . , x400*) and the corresponding value of *y,* if such correlations exist in the data. Sure, it learns the correlations without being explicitly told what they are, but these are correlations nonetheless. Is identifying correlations the same thing as thinking and reasoning? Surely, if the Mark I distinguished the letter "B" from the letter "G," it was simply going by the patterns and did not attach any meaning to those letters that would engender further reasoning. Such questions are at the heart of the modern debate over the limits of deep neural networks, the astonishing descendants of perceptrons. There is a path connecting these early perceptrons to the technology of large language models or the AI being developed for, say, self-driving cars. That path is not a straight one; rather, it's long and winding, with false turns and dead ends. But it's a fascinating, intriguing path nonetheless, and we are setting off on it now.

Building the perceptron device was a major accomplishment. An even bigger achievement was the mathematical proof that a single layer of perceptrons will always find a linearly separating hyperplane, *if* the data are linearly separable. Understanding this proof will require us to get our first taste of vectors and how they form the backbone of methods used to represent data in machine learning. It's our first mathematical pit stop.

CHAPTER 2

We Are All Just Numbers Here . . .

Less than a month before his death in September 1865, the Irish mathematician William Rowan Hamilton wrote a letter in four paragraphs to his son. In that letter, Hamilton recalled, among other things, a walk along the Royal Canal in Dublin, Ireland. It was October 16, 1843. Hamilton was on his way to attend a meeting of the Royal Irish Academy. His wife was with him. When the couple came underneath the Brougham Bridge, Hamilton, who had been struggling for more than a decade with some deep mathematical questions, had a flash of inspiration. "An electric circuit seemed to close; and a spark flashed forth . . . I [could not] resist the impulse—unphilosophical as it may have been—to cut with a knife on a stone of Brougham Bridge, as we passed it, the fundamental formula with the symbols, $i, j, k;$ namely, $i^2 = j^2 = k^2 = ijk = -1$."

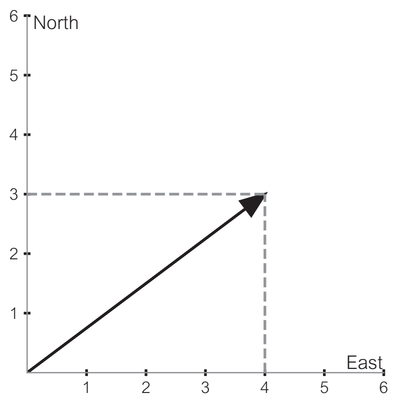
Hamilton signed off the letter to his son with these words: "With *this quaternion of paragraphs* [emphasis mine] I close this letter . . . Your affectionate father, William Rowan Hamilton." The use of the word "quaternion" was deliberate. A quaternion is a mathematical entity composed of four elements with very strange and special properties, which Hamilton discovered on that fateful day beneath Brougham Bridge. The equation he etched on the stone there, representing the general form of the quaternion, is one of the most famous examples of mathematical graffiti; the original, which has long since been defaced, was replaced by an official plaque reading:

Here as he walked by
on the 16th of October 1843
Sir William Rowan Hamilton
in a flash of genius discovered
the fundamental formula for
quaternion multiplication
$i^2 = j^2 = k^2 = ijk = -1$
& cut it on a stone of this bridge.

Quaternions are exotic entities, and they don't concern us. But to create the algebra for manipulating quaternions, Hamilton developed some other mathematical ideas that have become central to machine learning. In particular, he introduced the terms "scalar" and "vector." These days, most of us would likely not have heard of Hamilton, but we are intuitively familiar with the notion of scalars and vectors, even if not their formal definitions. Here's a quick primer.

Consider a man who walks five miles. Given that statement, the only thing we can say about what the man did is denoted by a single number: the distance walked. This is a scalar quantity, a stand-alone number. Now, if we were told that the man walked five miles in a northeasterly direction, we would have two pieces of information: the distance and the direction. This can be represented by a vector. A vector, then, has both a length (magnitude) and a direction. In the following graph, the vector is an arrow of magnitude 5.

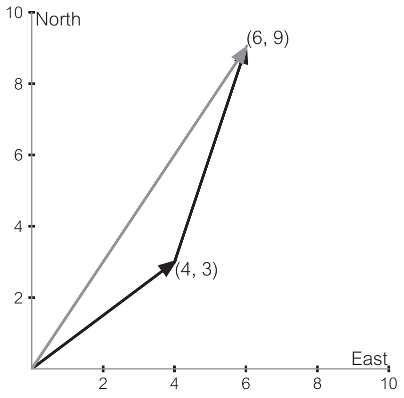
If you closely examine the vector, you'll see that it has two components: one along the x-axis and another along the y-axis. It's equivalent



to saying that the man went four miles in the direction due east and three miles in the direction due north. The vector representing the actual walk is an arrow going from (0, 0) to (4, 3), giving both the direction and the distance. The magnitude of the vector is simply the length of the hypotenuse of the right-angled triangle formed by the vector and its components along the x- and y-axes. So, the vector's magnitude, or length, is equal to $\sqrt{4^2 + 3^2} = 5$.

Thinking in terms of vectors, without using formal ways of representing and manipulating them, predates Hamilton. For example, by the late 1600s, Isaac Newton was already using geometric ways of thinking about vector-like entities such as acceleration and force. Newton's Second Law of Motion says that the acceleration experienced by an object is proportional to the force acting upon it and that the object's acceleration and the force have the same direction. The first corollary to Newton's Laws of Motion, in his *Principia,* states, "A body by two forces conjoined will describe the diagonal of a parallelogram, in the same time that it would describe the sides, by those forces apart." This is a statement about using geometry to add two vectors, even though Newton didn't call the quantities vectors.

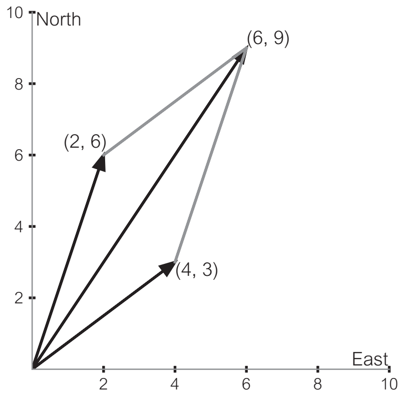
To understand vector addition, we can go back to our man who walked five miles, represented by a vector going from (0, 0) to (4, 3). After reaching the destination, the man turns more northward such that in the coordinate plane, he reaches (6, 9): He has effectively walked two more miles in the direction due east and six more miles in the direction due north. This is represented by a second vector, an arrow drawn from (4, 3) to (6, 9). This new vector has an x component of 2 and a y component of 6. What is the total distance the man walked? And what is the net distance in the xy coordinate space, from origin to the final destination? This graph shows you the answers to both:



The magnitude of the two individual vectors, or walks, is $\sqrt{4^2+3^2}=\sqrt{25}=5$ and $\sqrt{2^2+6^2}=\sqrt{4+36}=\sqrt{40}=6.32$. So, the total distance the man walks is 5 + 6.32 = 11.32 miles.

The resultant vector is an arrow drawn from the origin to the final destination, which is (6, 9), and its magnitude is $\sqrt{6^2+9^2}=\sqrt{36+81}=\sqrt{117}=10.82$. The net distance in the xy coordinate space is 10.82 miles.

This now helps us make sense of what Newton was saying. Let's say the acceleration caused by one force acting upon an object is given by the vector (2, 6) and that the acceleration caused by another force on the same object is given by the vector (4, 3). Both forces are acting on the object at the same time. What is the total acceleration of the object? According to Newton's corollary, the geometric interpretation involves drawing a parallelogram, as shown in the following figure; the net acceleration, then, is given by the diagonal vector (6, 9):



If the acceleration is in units of meters per second per second (m/s^2), then the net acceleration is given by the magnitude of the vector (6, 9), which equals 10.82 m/s^2, in the direction of the arrow.

I have chosen to add the same vectors in this case as in the example of the man walking, but here the two vectors represent acceleration, not distance, and they both have their tails at (0, 0). What this tells you is that the vector (2, 6) is the same vector regardless of whether its tail is at (0, 0) or at (4, 3), as in the previous example. An important property of vectors is that we can move the arrow representing a vector in the coordinate space, and if we don't change the length of the arrow and its orientation, it's the same vector. Why? Well, because we haven't changed its length or its direction, the two properties that define the vector.

None of this was formally understood as the beginnings of vector analysis when Newton published his *Principia* in 1687. His contemporary Gottfried Wilhelm Leibniz (1646–1716), however, had more than an inkling about this new way of thinking. In 1679, in a letter to another luminous contemporary, Christiaan Huygens, Leibniz wrote, "I believe that I have found the way . . . that we can represent figures and even machines and movements by characters, as algebra represents numbers or magnitudes." Leibniz never quite formalized his intuition, but his prescience—as we'll see when we understand the importance of vectors for machine learning—was astounding. Following Leibniz, a host of other mathematicians, including Johann Carl Friedrich Gauss (1777–1855), developed methods for the geometric representation of certain types of numbers in two dimensions, setting the stage for Hamilton's discovery of quaternions and the formalization of vector analysis.

VECTORS BY THE NUMBERS

Vector analysis doesn't have to be geometric. It can come down to manipulating numbers written in a certain format. And in fact, for machine learning, that's how we need to think about vectors. For example, the accelerations caused by the two forces in the previous example are simply arrays of two numbers each, [4, 3] and [2, 6], respectively. Adding them is the same as adding the individual components of each vector (stacked vertically, as a column). You don't have to fuss with arrows:

$$\begin{bmatrix} 4 \\ 3 \end{bmatrix} + \begin{bmatrix} 2 \\ 6 \end{bmatrix} = \begin{bmatrix} 6 \\ 9 \end{bmatrix}$$

Subtracting vectors is similar.

$$\begin{bmatrix} 4 \\ 3 \end{bmatrix} - \begin{bmatrix} 2 \\ 6 \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \end{bmatrix}$$

What just happened? Why is the y component of the resultant vector negative? If these numbers still represent acceleration, then subtraction meant that the second force was acting against the first force; along the x-axis, the acceleration is just a little bit less than when we were adding the two vectors, but it is still positive; along the y-axis, however, the force is now acting against the initial direction of motion, resulting in a deceleration.

One can multiply a vector by a scalar—simply multiply each element of the vector by the scalar.

$$5 \times \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 20 \\ 15 \end{bmatrix}$$

Geometrically, that's the same as stretching the arrow (or vector) five times in the same direction. The magnitude of the original vector is 5. Scaling it 5 times gives us a new magnitude of 25. If you were to calculate the magnitude of the new vector using its scaled-up coordinates, you'd again get:

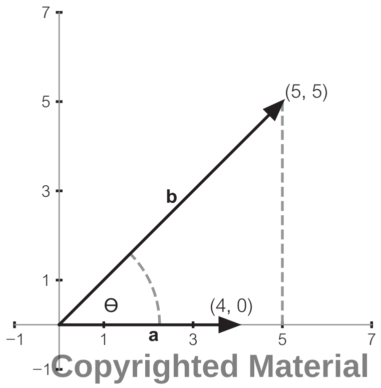
$$\sqrt{20^2 + 15^2} = \sqrt{400 + 225} = \sqrt{625} = 25$$

There's yet another way to represent vectors. Restricting ourselves to two dimensions, think of a vector of length one, **i**, along the x-axis and a vector of length one, **j**, along the y-axis. Note that **i** and **j** are in lowercase and boldface; this signifies that they are vectors. So, **i** can be thought of as an arrow that points from (0, 0) to (1, 0) and **j** as an arrow that points from (0, 0) to (0, 1). Each has a magnitude of 1 and is also called a unit vector. Given this, the vectors (4, 3) and (2, 6), in Cartesian coordinates, can be written as 4**i** + 3**j** and 2**i** + 6**j**, respectively. That's the same as saying that the vector (4, 3) is 4 units along the x-axis and 3 units along the y-axis and that the vector (2, 6) is 2 units along the x-axis and 6 units along the y-axis. The use of **i** and **j** is shorthand for representing vectors. It's also important to point out that a unit vector is simply a vector with a magnitude of 1; it doesn't have to lie along the perpendicular axes of some coordinate space.

These ideas apply to higher dimensions, too, and we'll come to that. For now, getting a handle on the mathematical manipulation of 2D vectors and their corresponding geometric meanings will go a long way toward helping us understand the role of their higher-dimensional counterparts in machine learning.

THE DOT PRODUCT

Another important operation with vectors is something called the dot product. Consider vector (4, 0), call it **a**, and vector (5, 5), call it **b**. (Again, the boldface and lowercase for letters **a** and **b** signify that they are vectors.) Conceptually, the dot product **a.b**—read that as "a dot b"— is defined as the magnitude of **a** multiplied by the projection of **b** onto **a**, where the projection can be thought of as the "shadow cast" by one vector onto another.



The magnitude of **a** is denoted by $\|\mathbf{a}\|$ or $|\mathbf{a}|$. The projection of **b** onto **a** is given by the magnitude of **b**, or $\|\mathbf{b}\|$, multiplied by the cosine of the angle between the two vectors. For the vectors we have chosen, the angle between them is 45 degrees (or $\frac{\pi}{4}$ radians), as shown in the preceding graph. So:

$$\mathbf{a.b} = \|\mathbf{a}\| \times \|\mathbf{b}\| \times \cos(\pi / 4)$$

$$\cos(\pi / 4) = {}^{1}\!/\!{}_{\sqrt{2}}$$

$$\|\mathbf{a}\| = \sqrt{4^2 + 0^2} = 4$$

$$\|\mathbf{b}\| = \sqrt{5^2 + 5^2} = \sqrt{50} = 5\sqrt{2}$$

$$\Rightarrow \ \mathbf{a.b} = 4 \times 5\sqrt{2} \times \frac{1}{\sqrt{2}} = 20$$

Note: the symbol ⇒ means "which implies that."

Now let's make a couple of small tweaks. Let the vector **a** be given by (1, 0), vector **b** by (3, 3). Vector **a** has a magnitude of 1, so it's a "unit vector." Now, if you were to take the dot product **a.b**, you'd get:

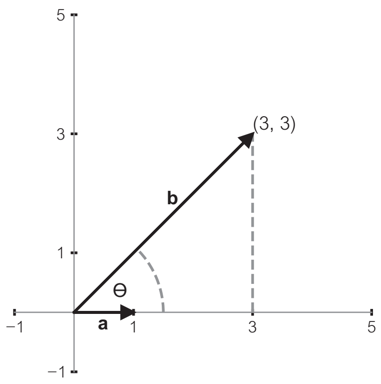
$$\mathbf{a.b} = \|\mathbf{a}\| \times \|\mathbf{b}\| \times \cos(\pi / 4)$$

$$= 1 \times 3\sqrt{2} \times \cos\left(\frac{\pi}{4}\right)$$

$$= 1 \times 3\sqrt{2} \times {}^{1}\!/\!{}_{\sqrt{2}}$$

$$= 3$$

The dot product turns out to be equal to the x component of vector **b**, or the shadow cast by **b** onto the x-axis, the direction of the

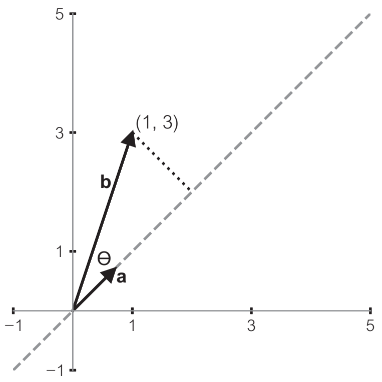


unit vector. This gives us a crucial geometric intuition: If one of the vectors involved in a dot product is of length 1, then the dot product equals the projection of the other vector onto the vector of unit length. In our special case, the unit vector lies along the x-axis, so the projection of vector **b** onto the x-axis is simply its x component, 3.

But here's something amazing about dot products. Even if the unit vector is not along one of the axes, this geometric truth still holds. Let's say **a** is $\left(1/\sqrt{2}, 1/\sqrt{2}\right)$. Its magnitude is 1, so it's a unit vector, but it's at a 45-degree angle to the x-axis. Let's say **b** is the vector (1, 3). The dot product **a.b** is $\|\mathbf{a}\| \times \|\mathbf{b}\| \times \cos(\theta)$, which equals $1 \times \|\mathbf{b}\| \times \cos(\theta)$, which in turn is the projection of the vector **b** onto the straight line that extends along vector **a** (see figure on next page).

Another important thing the dot product tells us about two vectors is whether they are at right angles, or orthogonal, to each other. If they are at right angles, then cosine of (90°) equals zero. So, regardless of the length of the vectors, their dot product, or the projection of vector **b** onto vector **a**, is always zero. Conversely, if the dot product of two vectors is zero, they are orthogonal to each other.

How would we calculate the dot product if we were to use the



other method for representing vectors, using their components, and we didn't know the angle between the two vectors?

Say, $\mathbf{a} = a1\mathbf{i} + a2\mathbf{j}$ and $\mathbf{b} = b1\mathbf{i} + b2\mathbf{j}$. Then:

$$\mathbf{a}.\mathbf{b} = (a1\mathbf{i} + a2\mathbf{j}).(b1\mathbf{i} + b2\mathbf{j}) = a1b1 \times \mathbf{i}.\mathbf{i} + a1b2 \times \mathbf{i}.\mathbf{j} + a2b1 \times \mathbf{j}.\mathbf{i} + a2b2 \times \mathbf{j}.\mathbf{j}$$

Note that the second and third terms in the equation turn out to be zero. The vectors **i** and **j** are orthogonal, so **i.j** and **j.i** are zero. Also, both **i.i** and **j.j** equal 1. All we are left with is a scalar quantity:

$$\mathbf{a}.\mathbf{b} = a1b1 + a2b2$$

MACHINES AND VECTORS

If all this feels far removed from machine learning, perceptrons, and deep neural networks, rest assured it's not. It's central to the plot. And we are getting there, by leaps and bounds, yet by stepping carefully only on the stones necessary for sure footing.

It's time to revisit the perceptron and think of it in terms of vectors. The intent is to gain geometric insights into how data points and the weights of a perceptron can be represented as vectors and how to visualize what happens when a perceptron tries to find a linearly separating hyperplane that divides the data points into two clusters. Much of it has to do with using dot products of vectors to find the relative distances of the data points from the hyperplane, as we'll see.

Recall the generic equation for a perceptron, which says that the perceptron outputs 1 if the weighted sum of its inputs plus some bias term, *b,* is greater than 0; otherwise, it outputs −1.

$$g(\mathbf{x}) = w1x1 + w2x2 + \cdots + wnxn + b = \sum_{i=i}^{n} wixi + b$$

$$f(z) = \begin{cases} -1, & z \leq 0 \\ 1, & z > 0 \end{cases}$$

$$y = f(g(\mathbf{x})) = \begin{cases} -1, & g(\mathbf{x}) \leq 0 \\ 1, & g(\mathbf{x}) > 0 \end{cases}$$

We have made a subtle change to the notation we used previously: The argument to the function g is now a vector; in the previous chapter, because we hadn't yet introduced the notion of vectors, we simply had $g(x)$ instead of $g(\mathbf{x})$. Let's stick to the two-dimensional case, with data points given by different values for (*x1, x2*) and the weights of the perceptron given by (*w1, w2*). The perceptron first computes the weighted sum of the inputs:

$$w1x1 + w2x2$$

If this weighted sum is greater than some threshold, call it $-b$, then the perceptron's output, *y,* is 1. Else it is −1. So: