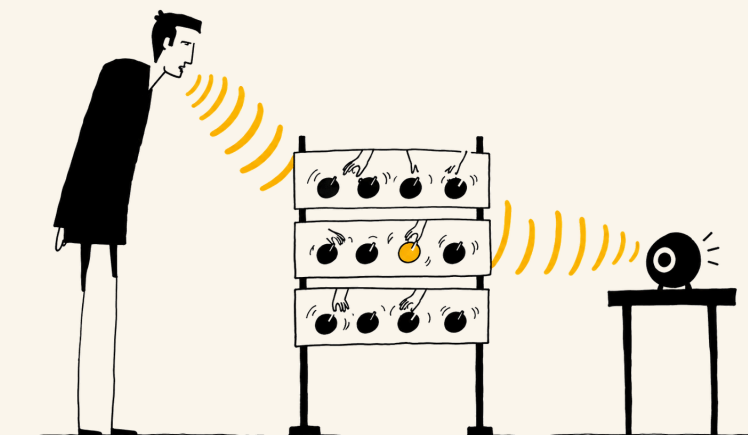


Dive into Deep Learning



ton Zhang, Zachary C. Lipton, Mu Li, Alexander J. Smola · 20

Illustrated audio lectures

Lecturized

Dive into Deep Learning

Illustrated audio lectures

Aston Zhang, Zachary C. Lipton, Mu Li, Alexander J. Smola · 2023

Retold as illustrated lectures by Lecturized
lecturized.com

From the original: Dive into Deep Learning (Cambridge University Press, 2023), d2l.ai

Contents

Rules that never learn

Knobs that learn a word

Programming with data

The right data and a fair score

Walking downhill, then learning from answers

How much? Regression

Which one? Classification

Many labels at once, and the best first

The feed that learns you

When order matters: sequences

Learning without a teacher

When the world answers back

Learning by acting: reinforcement learning

Old roots: from church doors to neurons

The road to deep learning

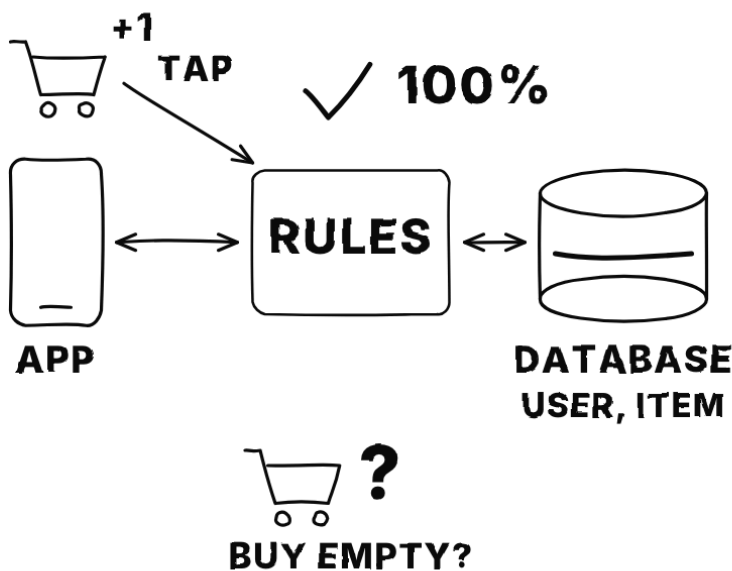
Hidden successes, visible ones

What AI is not, and what makes learning deep

Try it, share it

Rules that never learn

Open any shopping app on your phone, put one pair of socks in the basket, and watch what happens. A small number appears on the basket icon, the socks show up in the list, and the price is added to the total. Nobody at the shop looked at your tap; a program did all of it, and it did it the same way for you as for the millions before you. That program was written as a set of rules: when this happens, do that.



An online shop as rules

The book begins with exactly such a program, an online shop, and asks how its makers would build it. They would stand at a whiteboard

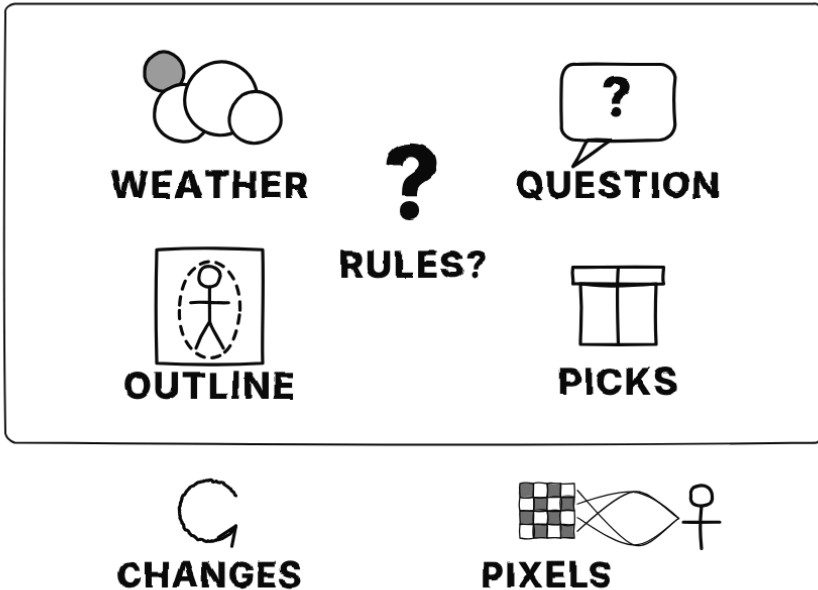
for a few hours and split the work into three pieces. First, there is the face of the shop, the pages you see in a web browser or in the app. Second, there is a database, a large, carefully kept table that remembers who you are, what is in your basket, and everything anyone ever bought. Third, in the middle, sits what programmers call the business logic, and what the authors, half joking, call the brains of the application. The brains are a long list of rules that say, for every situation anyone can think of, what the program must do.

Here is one such rule, and it is just as dull as it sounds. When a customer taps "add to basket", the program writes a new line into the basket table, pairing the customer's number with the product's number. Then the team walks through the strange corner cases one by one. What should happen if someone presses "buy" with an empty basket? They write a rule for that too, test it, fix what breaks, and test again. The same goes for the simple settings in your phone: "if the battery is below twenty percent, dim the screen" is a rule someone typed in once, and it will never change by itself.

The remarkable thing, the book says, is that this works. People can design such systems in their heads, often for situations nobody has met before, and launch them before a single real customer arrives. And the authors add a piece of plain advice: if you can write rules that are right 100% of the time, you do not need machine learning at all. A rule that is always right is cheaper, clearer and safer than anything learned.

Now picture the same smart team at the same whiteboard, but with a different task. The book offers four of them. Predict tomorrow's weather from maps, satellite pictures and the weather of the last few days. Answer any short factual question that someone types in their

own words. Look at a photo and draw an outline around every person in it. Show each shopper products they will probably love but would never have found by browsing. You meet the last two every day: your phone's photo gallery finds the faces of your friends, and a video app fills your feed with things you did not search for.



Four tasks with no rules

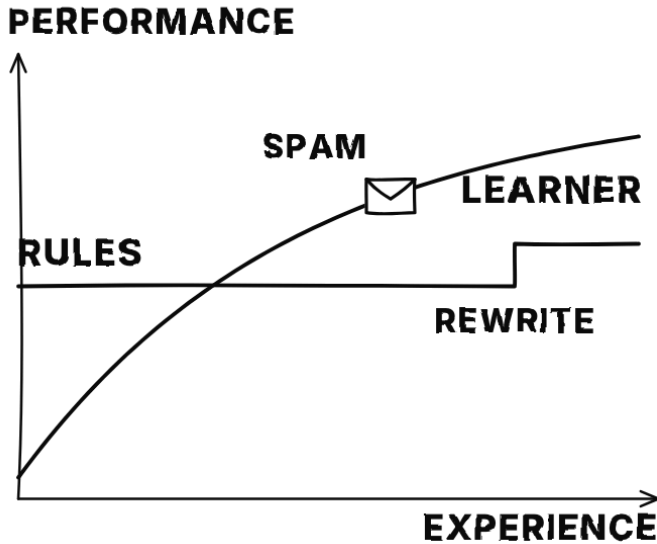
Try to write rules for any of these and you get stuck almost at once. The book names two reasons. Sometimes the right answer keeps moving: what shoppers like this spring is not what they liked last spring, so a fixed list of rules would go stale. Any good solution has to follow a world that changes. At other times the link between what goes in and what comes out is far too tangled. A photo is only a grid of coloured dots, called pixels, and the step from those dots to the idea

"this is a person" may take thousands or millions of small calculations that nobody knows how to write down. You recognise a friend's face in a blink, yet you could not explain, step by step, how you did it.

Try it yourself

Ask someone to write down, as rules, how to tell a cat from a dog in a photo. They will begin with ears, size and noses, and within a minute they will hit a fluffy dog with pointed ears or a cat photographed from behind. Every rule they add breaks on some new picture, even though they themselves never mistake a cat for a dog. That gap, between what we can do and what we can write down, is exactly where machine learning comes in.

So the book gives its first definition. Machine learning is the study of programs that learn from experience: the more experience such a program gathers, the better it does its job. The experience usually comes as data, such as millions of labelled photos, or from trying things out in some environment and seeing what happens. A spam filter is the everyday case: each time people mark a message as spam, the filter has a little more to learn from, and next week it catches more junk than it did this week. Compare that with the online shop. Its rules stay exactly the same however many customers pass through, and they change only when the programmers themselves learn something and sit down to rewrite the code. In the shop, the people learn; in machine learning, the program does.



Better with experience

The book then says where it is going. It will teach the foundations of machine learning, and above all one powerful family of its methods, called deep learning. Deep learning is what drives the recent leaps in computer vision, in understanding human language, in medicine and in reading the genome. When a chatbot answers your question in fluent sentences, or a voice assistant understands you over the noise of a street, you are hearing deep learning at work.

Remember

Most programs you use are lists of rules written by people, and they behave the same way forever until people rewrite them. That approach works beautifully when the rules can be written down and are always right. It fails when the answer keeps changing, or when the path from input to output is too tangled to describe, as in seeing a face in a photo. Machine learning is the study of programs that get better with experience, usually with more data. This book teaches machine learning, and mainly deep learning, the family of methods behind today's progress in vision, language and science.

Questions

1. A shop's rule says: when a customer taps "add to basket", write a new line into the basket table. What happens to this rule after a million customers have used it?
 - a. It slowly adjusts itself to the most common baskets
 - b. It stays exactly the same until programmers rewrite it
 - c. It is replaced by a learned rule automatically
 - d. It stops working once the table is full of lines
2. Why would a fixed list of rules for recommending products to shoppers go stale?
 - a. Because the database forgets old purchases
 - b. Because the rules run too slowly on phones

- c. Because what shoppers like keeps changing over time
 - d. Because shoppers never buy recommended products
3. You can tell a cat from a dog instantly. Why does that not help you write rules for a program that does it?
- a. You cannot spell out, step by step, how you do it
 - b. Cats and dogs look the same to a camera
 - c. Photos have too few pixels to show the animal
 - d. Programs are not allowed to look at photos
4. According to the authors' advice, when should you not bother with machine learning?
- a. When the program will be used by millions of people
 - b. When the task involves a database
 - c. When you have only a small amount of data
 - d. When you can write rules that are right every time

Answers

- 1. b)** The shop's rules stay exactly the same however many customers pass through; they change only when the programmers themselves learn something and rewrite the code.
- 2. c)** Sometimes the right answer keeps moving: what shoppers like this spring is not what they liked last spring, so a good solution has to follow a changing world.
- 3. a)** We recognise faces and animals in a blink, yet the steps are beyond what we can explain; every rule we write down breaks on some new picture.
- 4. d)** If you can write rules that are right 100% of the time, you do not need machine learning at all.

Practice

1. Your phone dims its screen when the battery drops below twenty percent, and your video app fills your feed with things you did not search for. Which of the two is a fixed rule and which calls for machine learning? Explain why, using the reasons from the lecture.

Hint: Ask whether the right answer could be written down once and stay right forever.

Answer: The battery setting is a fixed rule: it can be written down once and it is always right, so no learning is needed. The feed is a machine learning task: what each person will enjoy keeps changing, and the link between a person's history and the videos they will like is too tangled to write as rules, so the program has to improve from experience.

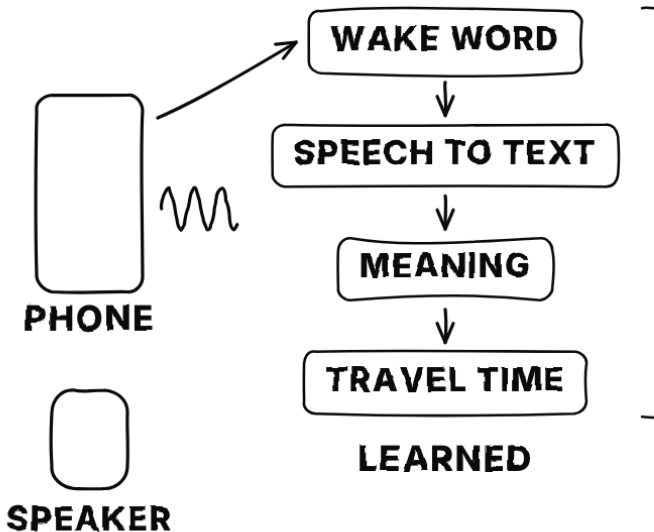
2. Explain in your own words the sentence: "In the shop, the people learn; in machine learning, the program does."

Hint: Think about what changes the shop's behaviour, and what changes a spam filter's behaviour.

Answer: The shop's program behaves the same however much it is used; it gets better only when its programmers learn something and rewrite the rules. A machine learning program, like a spam filter, gets better by itself as it gathers experience, usually more data, without anyone rewriting its rules.

Knobs that learn a word

Say "Hey Siri" or "OK Google" across a quiet room and watch your phone wake up. It lights up within a second, even though it was lying face down and, a moment ago, seemed to be doing nothing at all. Then ask it for the way to the nearest coffee shop. Your words appear on the screen as text, the map app opens by itself, and next to each route you see how many minutes it should take.

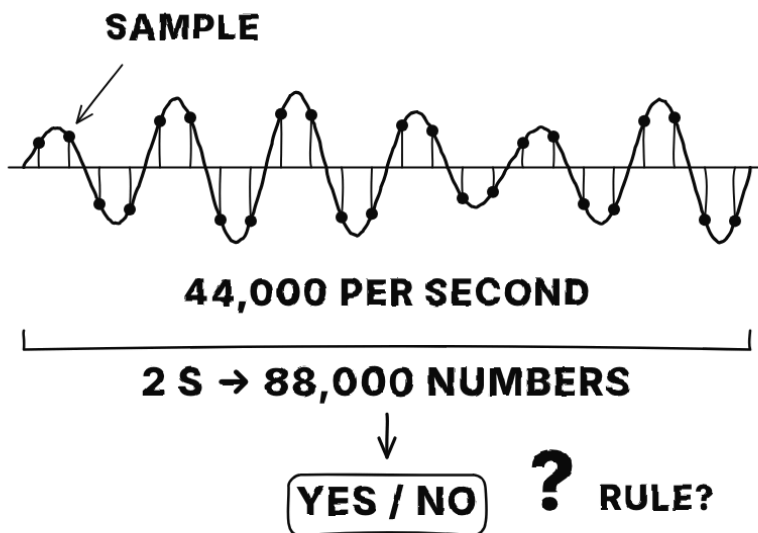


Four learned programs in a phone

The authors tell exactly this story about themselves, driving out for coffee before they started writing the book, and they admit it is made

up to make a point. The point is that in those few seconds the phone used several programs that were not written as rules. One heard the wake word, one turned speech into text, one understood that you wanted directions, and one guessed the travel time. Each of them learned its job from examples. The same holds for the voice assistant on a kitchen speaker: it answers to its name, but nobody ever typed into it a rule that says what its name sounds like.

To see why, try to write the first of these programs yourself, the one that wakes up on a word. Picture yourself alone in a room with a computer and nothing else; the book's figure shows just that, a person staring at a code editor and a sound wave. Start with what the program actually receives. The microphone does not hear words; it measures how far the air pushes its tiny membrane, again and again. Each such measurement is called a sample, and it is simply the height, or amplitude, of the sound wave at that instant. A phone takes roughly 44,000 samples every second. So a snippet of two seconds is $2 \times 44,000 = 88,000$ numbers. From those 88,000 numbers the program must give one answer: yes, the wake word is in there, or no, it is not.

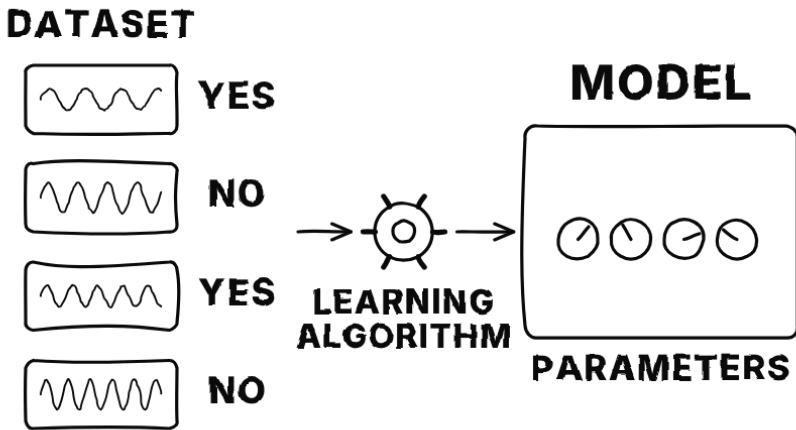


88,000 numbers, one answer

What rule would you write? Perhaps "if the sound gets loud and then quiet twice, say yes", but people speak loud and soft, fast and slow, with clogs and with accents, with music in the background. The book admits it plainly: the authors do not know how to write such a program from scratch either. That is exactly why we use machine learning.

Here is the trick the book offers. You cannot write down how you recognise the word "Alexa", but you recognise it at once. So you can do something else: collect thousands of short recordings and mark each one yourself, yes if it contains the word and no if it does not. Such a collection of examples is called a dataset, and the marks are called labels. Then, instead of writing the recogniser, you write a

flexible program whose behaviour depends on many adjustable numbers, called parameters. Finally you let the dataset decide which values of the parameters make the program score best on a measure you chose, for instance how often its yes or no matches your label.

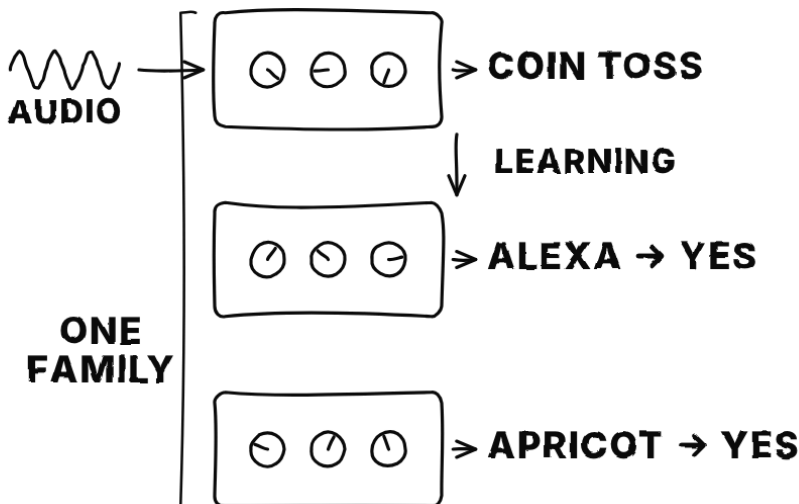


Labels set the knobs

The authors suggest picturing the parameters as knobs on a mixing desk. Turn the knobs and the program behaves differently; the same incoming sound gives a different answer. Once every knob is set, the book calls the program a model. All the different programs you could get by turning the same knobs make up a family of models. And the program that looks at the dataset and decides where to set the knobs is called the learning algorithm, a program that writes the settings of another program. A spam filter is built in just this way: its family is

fixed in advance, and the messages people mark as spam decide where its knobs end up.

Before any learning starts, someone has to state the task exactly. Here the input is a snippet of audio, those tens of thousands of numbers, and the output is one of two answers, yes or no. Then you pick a family of models suited to that kind of input and output. If the family is a good one, there is a setting of the knobs at which the model says yes every time it hears "Alexa". And since the wake word could be anything, a good family is rich enough that another setting makes it say yes only to "Apricot". Both tasks feel alike, so one family should serve for both. But if you wanted to go from photographs to captions, or from English sentences to Chinese ones, the inputs and outputs are of another kind entirely, and you might need a different family altogether.

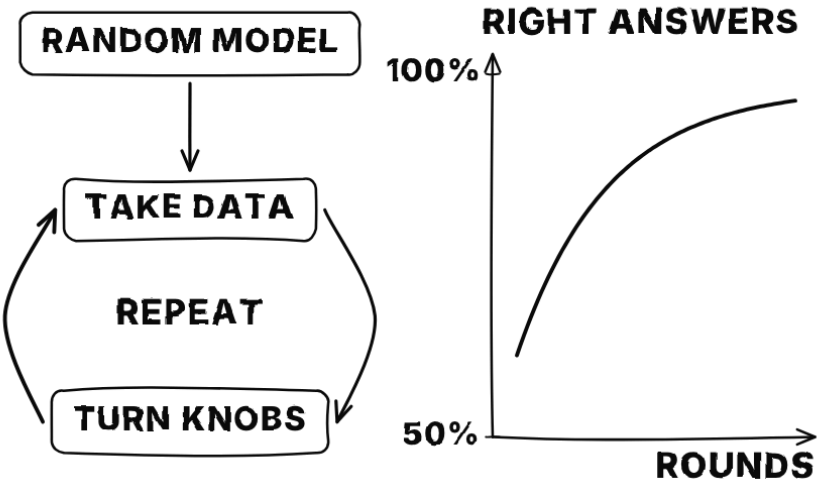


One family, many settings

Now, where do the knobs start? If you set them at random, the model will almost certainly not recognise "Alexa", nor "Apricot", nor any other word; it says yes and no more or less by chance. In machine learning, learning means exactly this: finding the setting of the knobs that makes the model do what we want. We also say that we train the model with data.

The book's second figure draws training as a loop, and it has four steps. First, start with a model whose knobs are set at random and which is useless. Second, take some of your data, for example a handful of recordings with their yes or no labels. Third, turn the knobs a little so that the model gets more of those examples right. Fourth, go back to the second step, and keep going round until, in the

authors' words, "the model is awesome". Picture a few hundred recordings at a time going in, a small turn of every knob after each batch, and the share of right answers creeping up from about half, which is coin tossing, towards nearly all.



The training loop

Try it yourself

Ask a friend to hide an object in a room and guide you only with "warmer" and "colder". You start walking in a random direction, which is useless; after each step you hear one word and adjust a little; you repeat. You never learn a rule for where the object is, yet after enough rounds you stand right next to it. Training a model works the same way, except that it has millions of directions to walk in at once and the "warmer" comes from comparing its answers with the labels.

Remember

Your phone wakes up on a word because a model learned it from examples, not because someone wrote a rule for it. The input is huge, about 44,000 amplitude samples a second, and the output is a single yes or no. We cannot describe how we hear a word, but we can label recordings, and that labelled dataset is what teaches the program. A model is a flexible program with its knobs, the parameters, set to particular values; the learning algorithm is what finds those values. Training is a loop: start at random, look at some labelled examples, turn the knobs to do better on them, and repeat.

Questions

1. Nobody can write down how to recognise the word "Alexa".
What makes it possible to teach a program to do it anyway?
 - a. People can recognise the word and label recordings yes or no
 - b. The microphone measures the word directly
 - c. The word always has the same loudness pattern
 - d. The phone stores a recording of every possible voice
2. A wake-word program has all its knobs set at random. What should you expect?
 - a. It recognises Alexa but not other words
 - b. It refuses to answer at all
 - c. It answers yes and no more or less by chance
 - d. It recognises every word equally well
3. Why might the same family of models serve for "Alexa" and "Apricot" but not for turning photographs into captions?
 - a. Captions are longer than wake words
 - b. Photographs to captions is a different kind of input and output
 - c. Apricot is not a real wake word
 - d. The learning algorithm cannot handle photographs
4. Which of these is the learning algorithm in the wake-word example?
 - a. The microphone that collects the samples
 - b. The finished model with its knobs set
 - c. The labelled recordings of yes and no
 - d. The program that uses the dataset to set the knobs

Answers

1. **a)** You cannot write down how you recognise the word, but you recognise it at once, so you can label thousands of recordings and let that dataset teach the program.
2. **c)** With knobs set at random the model will almost certainly not recognise any word; learning is finding the setting that makes it do what we want.
3. **b)** The two wake words are alike, so one family serves both; photos to captions has inputs and outputs of another kind entirely and may need a different family.
4. **d)** The program that looks at the dataset and decides where to set the knobs is the learning algorithm, a program that writes the settings of another program.

Worked examples

How many numbers the wake-word model hears A phone's microphone takes about 44,000 samples every second. How many numbers does a wake-word model receive from a 3-second snippet, and how many answers must it give?

Each sample is one number, the height of the sound wave at that instant.

In one second there are about 44,000 samples.

In 3 seconds: $3 \times 44,000 = 132,000$ samples.

The task is stated as audio in, yes or no out, so all these numbers lead to a single answer.

Answer: About 132,000 numbers in, one yes-or-no answer out

Practice

1. Put the four steps of training in order and say what each one does: repeat; tweak the knobs; start with a random model; grab some labelled data.

Hint: The loop cannot begin until there is a model, however useless.

Answer: Start with a model whose knobs are random and which is useless; grab some data with labels; tweak the knobs so the model gets more of those examples right; repeat the last two steps until the model is good.

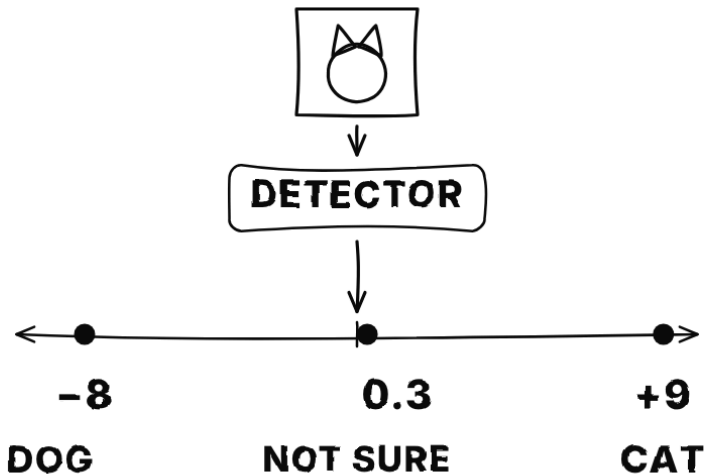
2. A snippet given to a wake-word model holds 88,000 samples. How long is the snippet?

Hint: Divide by the number of samples taken in one second.

Answer: $88,000 \div 44,000 = 2$, so the snippet lasts about 2 seconds.

Programming with data

Take a photo of your cat with your phone, then open the gallery and type "cat" into the search box. The photo turns up, together with every other cat you ever photographed, and no dogs. Nobody wrote a rule describing whiskers and ears for that search. Somewhere, a program was shown a great many photos of cats and dogs, each marked with the right answer, and it learned the difference from them.



A cat detector's score

This is what the training loop of the last part was for. Instead of writing a wake-word recogniser, we write a program that can learn to

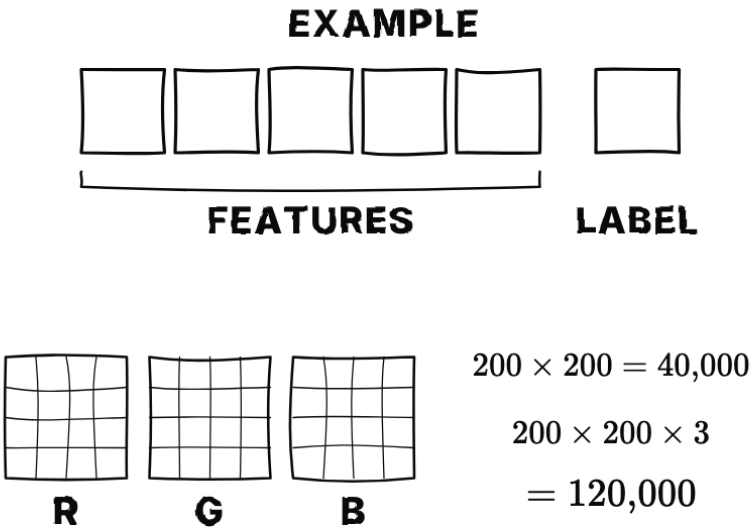
recognise wake words when it is given a large labelled dataset. The authors call this programming with data: the behaviour of the program is decided not by lines of code but by the examples we show it. In the same way we can "program" a cat detector simply by feeding it many pictures of cats and dogs.

The book says what such a detector ends up doing. For each picture it gives out one number. A very large positive number means "cat", a very large negative number means "dog", and something close to zero means "I am not sure". So if it gives +9 for one picture, -8 for another and 0.3 for a third, it is confident of a cat, confident of a dog, and undecided about the last one, perhaps a blurry fox. The book adds that this barely scratches the surface, and that deep learning is just one of many popular ways of solving such problems.

Now the book steps back and names the kind of problem we have been solving. We had recordings for which the answer was known, yes or no, and we wanted to predict the unknown answer for new recordings. Learning to predict a chosen unknown label from known inputs, using examples whose labels are known, is called supervised learning. It is only one kind of machine learning, and later parts meet others. But whatever the kind, the authors say, four ingredients always come along. There is the data to learn from. There is a model, which turns the data into an answer. There is an objective function, a score that says how well, or how badly, the model is doing. And there is an algorithm that turns the model's knobs so that the score improves.

This part looks at the first ingredient, the data. A dataset is a collection of examples, and the computer can only work with them once they are turned into numbers. Each example has a set of

attributes, called features, from which the model must make its prediction. In supervised learning there is one more special attribute, the label, which is what we want to predict; it is not given to the model as input. The book mentions that the same things go by other names, too: an example is also called a data point or a sample, and features are also called inputs or covariates.

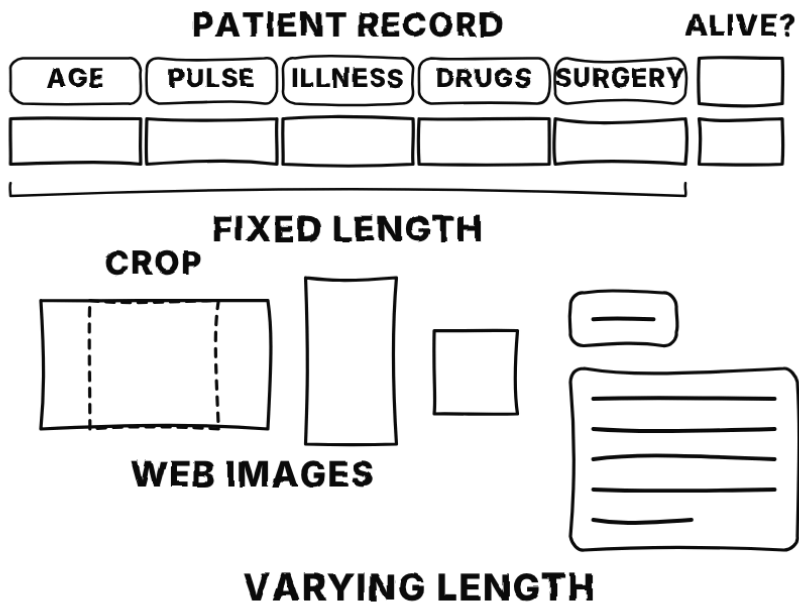


A photo as 120,000 numbers

For pictures, the features are the photograph itself, written as numbers, and the label is a number naming the category, say 1 for cat and 2 for dog. A colour photograph becomes three grids of numbers, one for the brightness of red light at every pixel, one for green, and one for blue. The book does the count for a photo of 200×200 pixels. One grid has $200 \times 200 = 40,000$ numbers, and there are three grids,

so the whole photo is $200 \times 200 \times 3 = 120,000$ numbers. A single small picture is already a list of a hundred and twenty thousand values. A photo straight from a modern phone has millions of pixels, so it is turned into many millions of numbers before any program can learn from it.

The data need not be pictures. The book's second example is hospital records, where the task is to predict whether a patient will survive the next 30 days. The features are things that are recorded anyway: age, vital signs such as pulse and blood pressure, other illnesses, the medicines taken, recent operations. The label, for each patient in the past records, is simply yes or no: did that patient survive the 30 days?

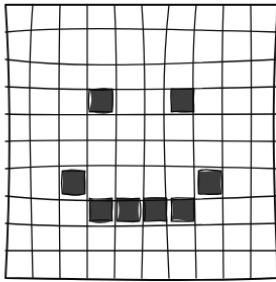


Fixed length versus varying length

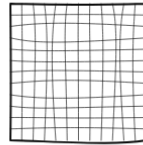
In both examples every case has the same number of features. Each photo of 200×200 pixels gives exactly 120,000 numbers, and each patient record gives the same list of measurements. Such inputs are called fixed-length vectors, a vector being an ordered list of numbers, and the length of that list is called the dimensionality of the data. So the dimensionality of our photo data is 120,000. Fixed length is convenient, the book says, because it is one less complication to worry about.

Real data often refuses to be so tidy. Pictures from a microscope come out of the same machine and share one size, but pictures collected from the Internet come in every size and shape. We could crop them all to a standard square, but whatever we cut off is lost. Text is even more stubborn. Think of reviews in an app store: one says only "it stinks!", the next runs on for pages about every feature. There is no natural fixed length for a review. The book counts it as one of the great advantages of deep learning that its models handle data of varying length with comparative grace.

Last, the book says something about quantity. In general, the more data we have, the easier the job. More data lets us train more powerful models and lean less on assumptions we made up in advance. The move from fairly small datasets to very large ones is one of the main reasons modern deep learning succeeded. Many of its most exciting models simply do not work without large datasets, and some that do work with little data are no better than older methods.



0 0 0 0 0 0 0 0 0 0
 ⋮
 0 0 0 1 0 0 1 0 0 0
 ⋮
 0 0 0 1 1 1 1 0 0 0
 ⋮
100 NUMBERS
DIMENSION 100



144
≠ 100

A smiley as 100 numbers

Try it yourself

Lay out a sheet of squared paper, ten squares by ten, and colour a simple smiley face in a few squares. Write 1 in every coloured square and 0 in every empty one, and read the numbers row by row. You now have a list of 100 numbers that is the picture, a fixed-length vector of dimensionality 100. Draw a second face on a sheet twelve by twelve and you get 144 numbers, which no longer fits the same list; that is the varying-length problem in miniature.

Remember

Programming with data means deciding a program's behaviour by the examples we show it, not by rules we write. Every machine learning problem has four ingredients: data, a model, an objective function that scores the model, and an algorithm that improves the score. In supervised learning each example has features, which the model sees, and a label, which it must predict. A 200×200 colour photo is 120,000 numbers; when every example has the same number of features, that number is the dimensionality. Deep learning copes well with data of varying length, and it needs large datasets to show its strength.

Questions

1. A cat detector gives out -7 for a photo. What does that tell you?
 - a. It is fairly confident the photo shows a dog
 - b. It is fairly confident the photo shows a cat
 - c. It is unsure what the photo shows
 - d. The photo has too few pixels to judge
2. In a hospital dataset for predicting 30-day survival, which of these is the label rather than a feature?
 - a. The patient's age
 - b. The medicines the patient takes
 - c. Whether the patient survived the 30 days
 - d. The patient's blood pressure

3. Why are app-store reviews hard to turn into fixed-length vectors?
 - a. They contain no numbers
 - b. They vary from two words to many pages
 - c. They are all written in different languages
 - d. They are too short to learn from
4. Which of these is NOT one of the four ingredients that come with every machine learning problem?
 - a. An objective function that scores the model
 - b. Data to learn from
 - c. An algorithm that improves the score
 - d. A rule written by hand for each case

Answers

1. **a)** A very large positive number means cat, a very large negative number means dog, and something near zero means not sure.
2. **c)** The features are things recorded anyway, like age, vital signs and medicines; the label is whether each past patient survived the 30 days.
3. **b)** One review says only "it stinks!" and another runs on for pages, so there is no natural fixed length for a review.
4. **d)** The four ingredients are data, a model, an objective function and an algorithm that turns the knobs; hand-written rules are what machine learning replaces.

Worked examples

The numbers in a 200×200 photo A colour photograph of 200×200 pixels is written as three grids of numbers, for red, green and

blue. How many numbers is it, and what is the dimensionality of a dataset of such photos?

One grid holds one brightness value per pixel: $200 \times 200 = 40,000$ numbers.

There are three grids, one for each colour of light: $40,000 \times 3 = 120,000$.

Every photo of this size gives the same number of values, so the inputs are fixed-length vectors.

The dimensionality is the length of those vectors.

Answer: 120,000 numbers per photo; the dimensionality is 120,000

Practice

1. A photo of 300×200 pixels is written as three grids for red, green and blue. How many numbers is it?

Hint: First count the pixels, then multiply by the number of colour grids.

Answer: $300 \times 200 = 60,000$ pixels, times 3 grids gives 180,000 numbers.

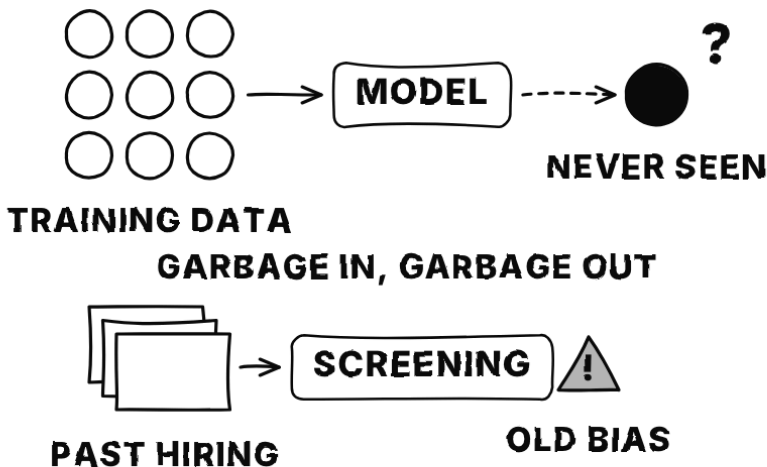
2. A smiley face is coloured on squared paper of 10 by 10 squares and written as 1s and 0s. A second one is drawn on 12 by 12. Why can these two not go into the same dataset of fixed-length vectors without changing something?

Hint: Count the numbers each sheet gives.

Answer: The first gives $10 \times 10 = 100$ numbers and the second $12 \times 12 = 144$, so the vectors have different lengths; one would have to crop or resize, which may lose information, or use a model that handles varying length.

The right data and a fair score

Imagine a skin-checking app that looks at a photo of a mole and says whether a doctor should see it. It was trained on many thousands of photos, and in tests it was very accurate. Now someone with dark skin photographs a mole and gets an answer that is plainly wrong. The book uses exactly this case, a skin cancer recognition system that had never seen black skin, and it gives the reason in three words: the data was wrong.



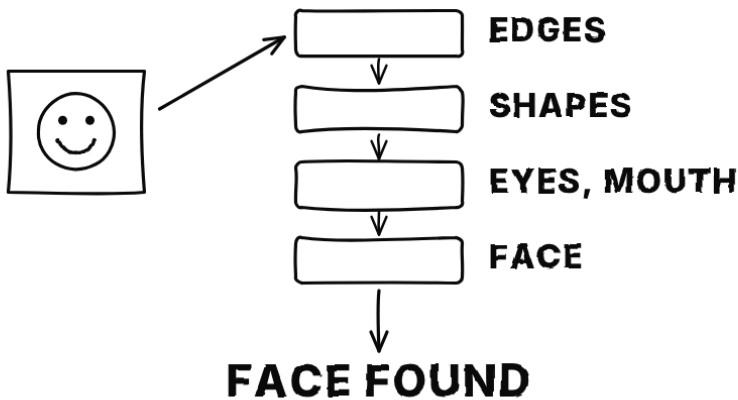
Missing groups, old prejudice

The previous part was about having enough data and turning it into numbers. Here the authors add that this is not enough: we need the right data. If the examples are full of mistakes, or if the features we chose simply do not tell us anything about what we want to predict, learning fails however clever the method. Programmers have a cliché for it, "garbage in, garbage out". And a poor score is not the worst that can happen. The book names areas where wrong data does real harm: predicting crime for the police, screening job applications, and deciding who gets a loan.

Two ways of going wrong are common. The first is that some group of people is simply missing from the training data, like the dark skin in our app. The model has never seen such cases, so it has nothing to go on. The second is subtler: the data includes everyone, but it records old prejudices. Train a program on years of past hiring decisions, and if those decisions were unfair, the program learns the unfairness and repeats it automatically, at scale. A chatbot trained on text from the Internet can pick up the same slants that the text contains, which is why its makers test it on such cases. The book stresses that none of this needs a villain: it can all happen without the data scientist intending it, or even noticing.

The second ingredient is the model. Almost all machine learning transforms data of one kind into a prediction of another kind. A system might take in photos and say how much each person is smiling, or take in readings from sensors on a machine and say how normal or how unusual they look. The model is that machinery, from input to prediction, and in this book it is a statistical model, one whose settings are estimated from data. Simple models are perfectly good for simple problems. But the problems this book cares about go past what classical methods can do, and deep learning is set apart

mainly by the powerful models it uses. Those models pass the data through many transformations in a row, each one feeding the next, stacked from top to bottom; that depth is where the name deep learning comes from. A photo app that finds faces does this: early stages pick out edges, later ones shapes, later still eyes and mouths, and the last stage says "face".

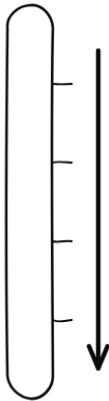


Many stages, one after another

The third ingredient answers a question that sounds philosophical. We said that learning means getting better at a task. But who decides what "better" is? If one person proposes a change to the model, another may well argue that it made things worse. To settle such arguments, machine learning needs a number that measures how

good or how bad a model is. That number is called an objective function.

LOSS



BETTER

score ↑

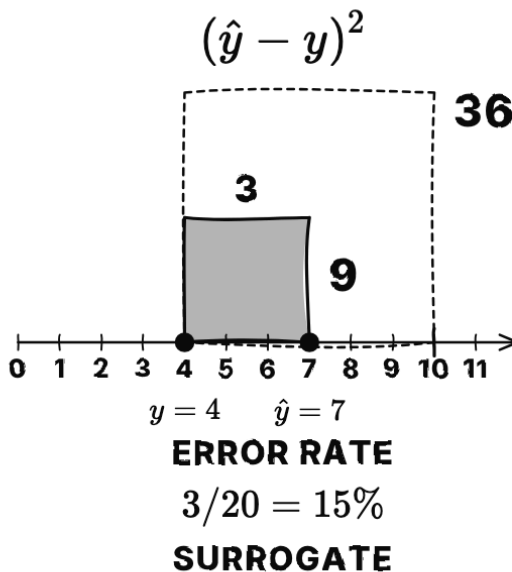
loss = - score ↓

Lower is better

By convention, objective functions are set up so that lower is better. That is only a habit, the book says. If you have a score where higher is better, such as the share of right answers, you just put a minus sign in front of it, and now lower is better while nothing else has changed. Because lower is better, these functions are also called loss functions: the smaller the loss, the better the model.

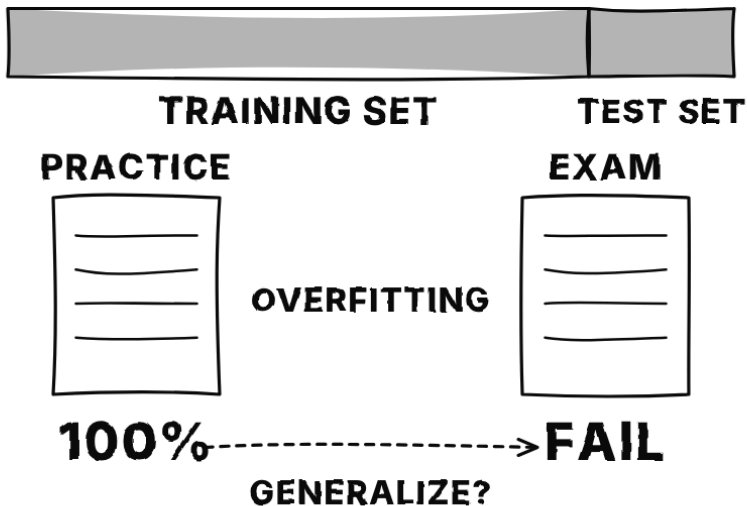
The book names the two most common losses. When we predict a number, the usual loss is the squared error: take the prediction, subtract the true value, and square the difference. Written down, if

the prediction is $\hat{y}$ and the true value is y , the squared error is $(\hat{y} - y)^2$. Say a model predicts a delivery will take 7 days and it takes 4. The difference is $7 - 4 = 3$, and the squared error is $3^2 = 9$. Squaring makes the loss positive whichever way we miss, and it punishes a big miss much more than a small one: missing by 6 costs 36, four times the cost of missing by 3. When we sort things into classes, the natural score is the error rate, the share of examples we get wrong; 3 wrong out of 20 is $3/20 = 0.15$, or 15 percent. The book warns that some losses, like squared error, are easy to make smaller step by step, while others, like the error rate, are awkward, because nudging the knobs a little usually changes no answer at all and the rate jumps only now and then. In such cases people optimize a stand-in that is easier to work with, called a surrogate objective.



Squared error and error rate

While we are training, we look at the loss as depending on the knobs of the model, the parameters, and we hold the data fixed. We collect a set of examples for training and look for the knob settings that make the loss on them as small as possible. But a small loss on those examples does not promise good answers on new ones. So the data is split in two: a training set, used to set the knobs, and a test set, kept aside and used only to check the result. In the end both scores are reported.



Training set, test set, overfitting

Try it yourself

Think of a student preparing for a final exam with a booklet of practice questions. After a few weeks they score full marks on the booklet every time, because they have learned the answers by heart. Then the exam comes with new questions, and they fail. The booklet score was the training score; the exam was the test.

That story is the book's own picture of what goes wrong. When a model does well on the training set but cannot carry that over to data it has not seen, we say it is overfitting the training data. What we really want is the opposite, a model that generalizes, doing about as well on new examples as on the ones it learned from. A spam filter that only blocks the exact messages it was shown would be overfitting; a useful one also blocks next week's new scam.

Remember

Lots of data is not enough; it has to be the right data, because garbage in gives garbage out. Data that leaves people out, or that carries old prejudices, produces models that fail or treat people unfairly, even when nobody meant it. A model turns input into prediction, and deep learning's models chain many transformations one after another. An objective function, or loss, scores the model, lower being better; the squared error $(\hat{y} - y)^2$ is the usual one for numbers, the error rate for classes. We train on a training set and check on a separate test set, because doing well only on the training data is overfitting.

Questions

1. A hiring model is trained on years of past decisions that included everyone but were unfair. What does the lecture say may happen?
 - a. The model learns the unfairness and repeats it automatically
 - b. The model fails because some group is missing
 - c. The model corrects the unfairness by itself
 - d. Nothing, unless the data scientist meant it to
2. A model predicts 10 where the true value is 13. What is its squared error?
 - a. 3
 - b. 6
 - c. 9
 - d. -9
3. Why do people often optimize a surrogate objective instead of the error rate?
 - a. The error rate is always too large
 - b. The error rate cannot be computed on a test set
 - c. The error rate rewards overfitting
 - d. Small turns of the knobs usually leave the error rate unchanged, so it is awkward to improve step by step
4. A spam filter blocks every message it was trained on, but lets most new scams through. What is going on?
 - a. It uses the wrong loss function

- b. It is overfitting the training data
- c. Its training set was too small to contain spam
- d. It was scored with lower-is-better

Answers

1. **a)** Data can include everyone and still record old prejudices; a program trained on unfair past decisions learns and repeats them, even without anyone intending it.
2. **c)** Squared error is the prediction minus the true value, squared: $10 - 13 = -3$, and $(-3)^2 = 9$; squaring makes it positive whichever way we miss.
3. **d)** Nudging the knobs a little usually changes no answer, so the error rate jumps only now and then; a stand-in that is easier to work with is optimized instead.
4. **b)** Doing well on the training set but failing to carry that over to unseen data is overfitting, like the student who memorized the practice booklet.

Worked examples

Two misses, two squared errors A delivery-time model predicts 7 days for one parcel that takes 4 days, and 7 days for another that takes 1 day. What are the two squared errors, and how do they compare?

Squared error is $(\text{prediction} - \text{true value})^2$.

First parcel: $7 - 4 = 3$, so the squared error is $3^2 = 9$.

Second parcel: $7 - 1 = 6$, so the squared error is $6^2 = 36$.

The second miss is twice as big but costs $36 \div 9 = 4$ times as much: squaring punishes big misses much more.

Answer: 9 and 36; the bigger miss costs four times as much

Practice

1. A classifier is tested on 50 photos and gets 4 of them wrong. What is its error rate?

Hint: The error rate is the share of examples predicted wrongly.

Answer: $4 \div 50 = 0.08$, or 8 percent.

2. Your score is "share of right answers", where higher is better. How do you turn it into a loss where lower is better, without changing what it measures?

Hint: Think of the sign.

Answer: Put a minus sign in front of it: a share of 0.9 becomes -0.9 and 0.8 becomes -0.8 , so the better model now has the lower number, and nothing else has changed.

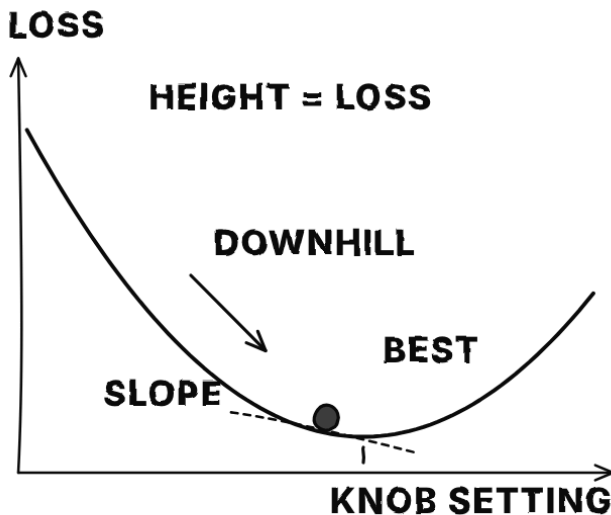
3. Explain in your own words why a skin-checking app that was very accurate in tests can still fail badly for some people.

Hint: Ask who was in the training and test photos.

Answer: If people with dark skin were missing from the photos it learned from, and from the tests too, the model never saw such cases and had nothing to go on; the good test score only covered the kind of people in the data. Garbage in, garbage out.

Walking downhill, then learning from answers

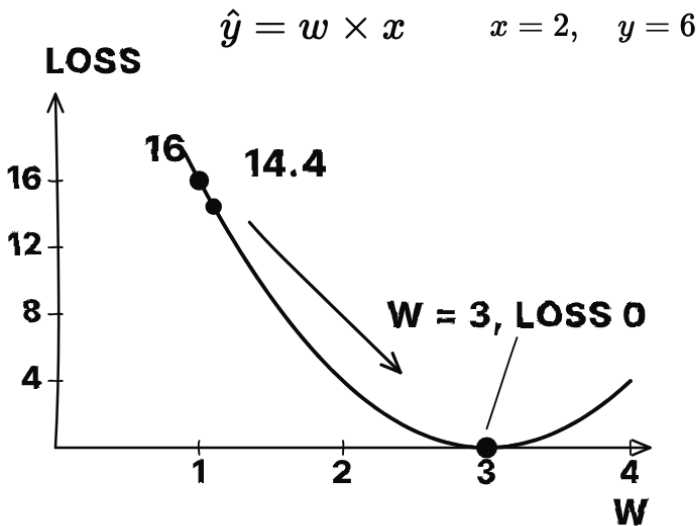
Stand blindfolded on a hilly field and try to reach the lowest point. You cannot see the valley, but you can feel the ground under your feet. Tap one foot a little forward, a little back, a little to each side, and feel which way the ground drops. Take a small step that way, and tap again. Step by step you end up at the bottom without ever having seen it.



Feeling the slope downhill

That is the fourth ingredient of machine learning, the algorithm that finds good knob settings. By now we have the data, a model with its knobs, and a loss that says how badly the model does. What is still missing is a way to search for the parameters that make the loss as small as possible. The methods most used in deep learning are built on one idea, called gradient descent. At every step it asks, for each parameter in turn: if I nudged this one by a tiny amount, would the loss on the training set go up or down, and by how much? Then it moves every parameter a little in the direction that makes the loss smaller. The height of the field is the loss, your position is the setting of the knobs, and the slope you feel with your foot is what the method measures.

Let us do it once with numbers, using a model with a single knob w . The model predicts a delivery time by multiplying the distance by w , so its prediction is $\hat{y} = w \times x$. We have one training example: a distance of $x = 2$ that took $y = 6$ days. Start with $w = 1$: the prediction is 2, the miss is $2 - 6 = -4$, and the squared error is 16. Now nudge the knob to $w = 1.1$: the prediction becomes 2.2, the miss -3.8 , and the loss about 14.4. The loss went down, so increasing w is the downhill direction, and the method takes a step that way. Repeat, and w creeps up towards 3, where the prediction is exactly 6 and the loss is zero. A real model does the same thing with millions of knobs at once, each nudged in its own downhill direction at every step. The language model behind a chatbot was trained exactly like this, by billions of such small downhill steps over its knobs.

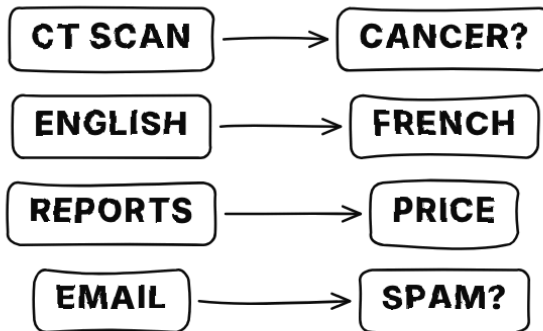


One knob, one example

With the four ingredients in hand, the book turns to a map of the kinds of problems machine learning can solve. The wake word was only one of them, and the authors want to give us a shared vocabulary for the rest of the book.



$$P(\text{label} \mid \text{features})$$



Supervised learning

The first and most common kind is supervised learning. We are given a dataset in which every example comes as a pair, features together with a label, and we must build a model that predicts the label from the features. It is called supervised because we, the supervisors, hand the model these labelled examples when its knobs are being set. In the language of probability, what we usually want is the chance of each possible label given the features we see. Supervised learning is not the only kind, but it accounts for most of the successful uses of machine learning in industry. The reason is that so many important tasks fit the pattern "predict the unknown thing from the data we have". The book gives three: cancer or not cancer from a computer tomography scan; the right French translation of an English sentence; next month's price of a share from this month's financial reports. Your

email's spam filter fits it too: the features are the message, and the label is "spam" or "not spam".

One short description covers all of supervised learning, but behind it lie many choices, which depend on what the inputs and outputs are like, how big they are, and how many there are. A sequence of any length, like a sentence, calls for a different kind of model from a fixed-length list of numbers.

The book then sketches the process from start to finish. Gather many examples whose features are known, choose a random subset, and get the true label for each. Sometimes the labels already exist in records, such as whether a patient died within the following year; sometimes people must be paid to supply them, for instance by sorting pictures into categories. Those inputs with their labels make up the training set. The training set goes into a supervised learning algorithm, which is a function of an unusual sort: you give it a whole dataset, and it gives you back another function, the trained model. Finally, new inputs the model has never seen go into it, and what comes out is taken as the prediction of their labels. The book's figure draws this as a line: labelled data, then the learning algorithm, then the model, and new data flowing through the model to predictions.

Remember

Gradient descent finds good parameters by feeling the slope: for each knob, it checks how the training loss would change after a tiny nudge, then moves every knob a little downhill. With one knob and one example, a few such steps take the loss from 16 towards zero. Supervised learning means predicting labels from features, learning from examples that come as feature and label pairs. Most practical successes of machine learning are of this kind. The algorithm takes a labelled dataset and returns a model, which then predicts labels for new inputs.

Questions

1. In gradient descent, nudging a knob up a little makes the training loss rise. What does the method do with that knob?
 - a. Leaves it exactly where it is
 - b. Moves it up a lot, to escape the rise
 - c. Moves it a little down, the direction that lowers the loss
 - d. Resets it to a random value
2. In the blindfolded walk on a hilly field, what stands for the loss?
 - a. The height of the ground where you stand
 - b. The direction you are facing
 - c. The size of your step
 - d. The number of steps you have taken

3. Why is the task "spam or not spam from the message" an example of supervised learning?
 - a. Because the filter discovers groups of messages without any labels
 - b. Because the filter learns from rewards it collects over time
 - c. Because messages have a fixed length
 - d. Because it learns from examples that pair a message with its label
4. What does a supervised learning algorithm give back when it is handed a labelled dataset?
 - a. A single prediction
 - b. A trained model, which is itself a function
 - c. A cleaned version of the dataset
 - d. A list of labels for the training examples

Answers

1. **c)** For each parameter the method checks how the loss would change after a tiny nudge, then moves it a little in the direction that makes the loss smaller.
2. **a)** The height of the field is the loss, your position is the setting of the knobs, and the slope you feel is what the method measures.
3. **d)** Supervised learning learns to predict labels from features using examples that come as feature and label pairs; here the message is the features and spam or not spam is the label.
4. **b)** You give it a whole dataset and it gives you back another function, the trained model, which then predicts labels for new inputs.

Worked examples

One step of gradient descent by hand A model predicts $\hat{y} = w \times x$. The one training example is $x = 2$, $y = 6$. Starting from $w = 1$, compare the squared error at $w = 1$ and $w = 0.9$ and decide which way gradient descent moves w .

At $w = 1$ the prediction is $1 \times 2 = 2$, the miss is $2 - 6 = -4$, and the squared error is 16.

At $w = 0.9$ the prediction is $0.9 \times 2 = 1.8$, the miss is -4.2 , and the squared error is $4.2^2 = 17.64$.

Nudging w down made the loss go up, from 16 to 17.64.

So the downhill direction is up: gradient descent increases w a little.

Answer: The loss rises when w decreases, so w is moved upwards, towards 3

Practice

1. With the same model $\hat{y} = w \times x$ and example $x = 2$, $y = 6$, what is the squared error at $w = 2$, and at $w = 4$? Where is the bottom of the valley?

Hint: Work out the prediction first, then the miss, then square it.

Answer: At $w = 2$ the prediction is 4, the miss -2 , the loss 4. At $w = 4$ the prediction is 8, the miss $+2$, the loss also 4. The loss is zero at $w = 3$, where the prediction is exactly 6, halfway between.

2. Describe in your own words the path from a pile of examples to a prediction in supervised learning.

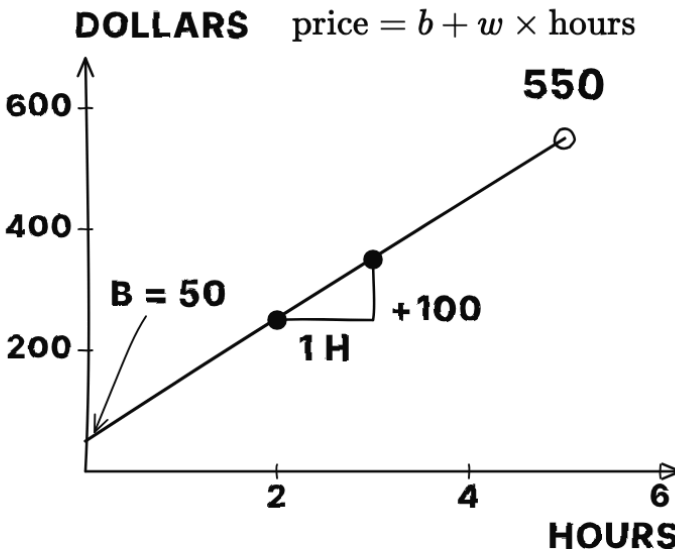
Hint: Start with gathering examples whose features are known.

Answer: Gather examples with known features, pick a random subset and get their true labels, from existing records or from people who label them. These pairs form the training set. The learning algorithm takes the

training set and returns a trained model, and new inputs fed to that model come out as predictions of their labels.

How much? Regression

Your drains are blocked, and a plumber spends 3 hours clearing them; the bill says 350 dollars. A friend calls the same plumber, who works 2 hours and charges 250 dollars. Now a neighbour asks you what a visit costs. You have never studied machine learning, yet you can already answer, and the book says that is exactly the point: you have just done regression.



Two bills fix the line

Here is how you did it, in three steps. You assumed that more hours cost more money, and that the plumber charges a fixed fee for

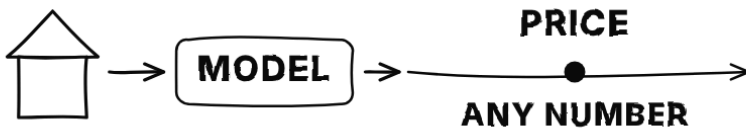
showing up plus a fixed amount per hour. Written as a formula, the bill is $\text{price} = b + w \times \text{hours}$, where b is the call-out fee and w the hourly rate. The extra hour, from 2 to 3, added $350 - 250 = 100$ dollars, so $w = 100$. Then 2 hours cost 200 of the 250 dollars, which leaves $b = 50$ for showing up. Check it on the first bill: $50 + 3 \times 100 = 350$, exactly right. So a 5-hour job should cost $50 + 5 \times 100 = 550$ dollars. The two numbers b and w are the knobs of the model, and finding them from examples is learning; because the formula draws a straight line, it is called linear regression.

The book calls regression perhaps the simplest supervised task to grasp, and introduces it with a table of house sales. Each row is one house; each column is one feature: the floor area in square feet, the number of bedrooms, the number of bathrooms, and the minutes it takes to walk to the town centre. One house is one example, and its row is its feature vector. In New York or San Francisco, the authors joke, unless you run one of the tech giants, your home might be $[600, 1, 1, 60]$: small, one bedroom, one bathroom, an hour's walk from the centre. In Pittsburgh it might be $[3000, 4, 3, 10]$. Every house gives four numbers in the same order, a fixed-length vector, and most classic methods need exactly that.

	[sq ft, beds, baths, min]
	ONE HOUSE
NEW YORK	[600, 1, 1, 60]
PITTSBURGH	[3000, 4, 3, 10]
	FIXED LENGTH: 4 NUMBERS

Houses as feature vectors

What makes a problem a regression, though, is not the features but the label. If you are buying a home and want its fair price, the examples are past sales and the labels are the prices actually paid. When the label can be any number, even if only within some range, the problem is a regression, and the aim is a model whose predictions come close to the true values. The book gives a handy rule: any question that begins with "how much" or "how many" is probably a regression. How many hours will this operation take? How much rain will fall on this town in the next six hours? How many days will this patient stay in hospital? The authors also recall the Netflix prize of 2009, a million dollars for whoever best predicted the star ratings users would give to films. A delivery app that tells you "arriving in 23 minutes" is answering a regression question each time you open it.



HOW MUCH? HOW MANY?

SURGERY HOURS

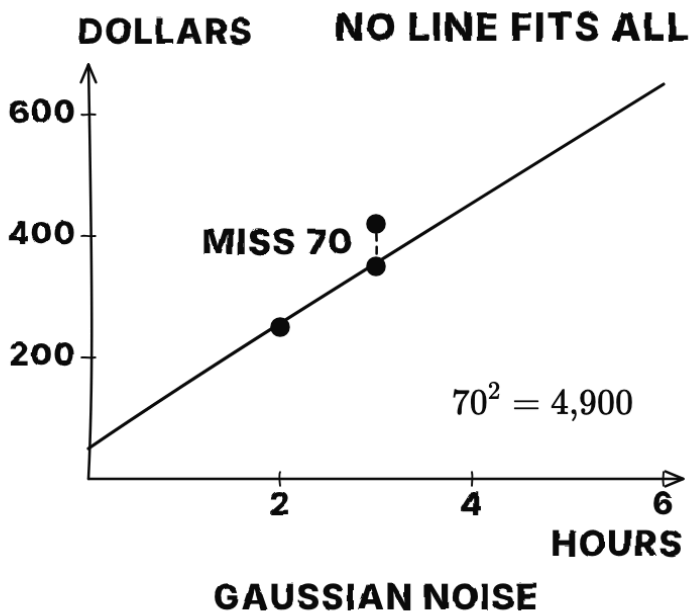
RAINFALL

HOSPITAL DAYS

MOVIE RATING

The label is a number

The plumber was an easy case, because two bills fixed the two knobs exactly. Real data is rarely so kind. A third bill might have been 420 dollars for 3 hours, because the pipes were older or the job was on a Sunday, things that hours alone do not capture. Then no straight line goes through every bill, and we settle for the line that misses them all as little as possible. The book says that for most of its chapters, "as little as possible" will be measured by the squared error, the miss squared and added up over all examples. If the line predicts 350 for that 420-dollar job, the miss is 70 and it adds $70^2 = 4,900$ to the loss. The authors add a promise for later: choosing squared error amounts to assuming that the data were disturbed by random noise of a particular bell-shaped kind, called Gaussian noise.



A bill off the line

Try it yourself

Find two receipts from the same taxi or ride app with different distances. Subtract the prices and divide by the difference in kilometres to get the price per kilometre, then subtract to find the fixed starting fee. Use your formula to predict a third ride, and compare it with the real receipt. If it is off, a surge or a traffic jam is the factor beyond your features, and the size of the miss is what squared error would measure.

Remember

Regression means predicting a number: any "how much" or "how many" question. Each example is a feature vector, such as a house's area, rooms and distance to the centre, and its label is a number, such as the sale price. From two plumber's bills we found the model price = $50 + 100 \times \text{hours}$: a fee of 50 dollars and 100 dollars an hour, which is linear regression. When no line fits every example, we choose the one with the smallest squared error.

Questions

1. What decides whether a problem is a regression?
 - a. That the features are numbers
 - b. That the label can be any number
 - c. That the data comes in a table
 - d. That the model is a straight line
2. Which of these questions is a regression, by the book's rule of thumb?
 - a. Is this mushroom poisonous?
 - b. Which digit is written here?
 - c. Is this message spam?
 - d. How many days will this patient stay in hospital?

3. A third plumber's bill does not lie on the line through the first two. What do we do?
- a. Choose the line that misses all the bills as little as possible, by squared error
 - b. Throw away the third bill
 - c. Use only the most recent bill
 - d. Draw a separate line for every bill

Answers

1. **b)** What makes a problem a regression is not the features but the label: when the label can be any number, the problem is a regression.
2. **d)** Any question that begins with "how much" or "how many" is probably a regression.
3. **a)** When no straight line goes through every bill, we settle for the one that misses them all as little as possible, measured by squared error.

Worked examples

The plumber's pricing A plumber charges 350 dollars for 3 hours and 250 dollars for 2 hours. Assuming $\text{price} = b + w \times \text{hours}$, find the call-out fee b and the hourly rate w , and predict the bill for 5 hours.

The extra hour from 2 to 3 adds $350 - 250 = 100$ dollars, so $w = 100$.

For 2 hours the hourly part is $2 \times 100 = 200$, so $b = 250 - 200 = 50$.

Check on the first bill: $50 + 3 \times 100 = 350$, which matches.

For 5 hours: $50 + 5 \times 100 = 550$.

Answer: $b = 50$ dollars, $w = 100$ dollars per hour; 5 hours cost 550 dollars

Practice

1. A taxi ride of 4 km costs 9 euros and one of 10 km costs 18 euros. Assuming a starting fee plus a price per kilometre, find both and predict a 6 km ride.

Hint: Six extra kilometres add how much?

Answer: 6 extra km add $18 - 9 = 9$ euros, so 1.5 euros per km; the starting fee is $9 - 4 \times 1.5 = 3$ euros; a 6 km ride costs $3 + 6 \times 1.5 = 12$ euros.

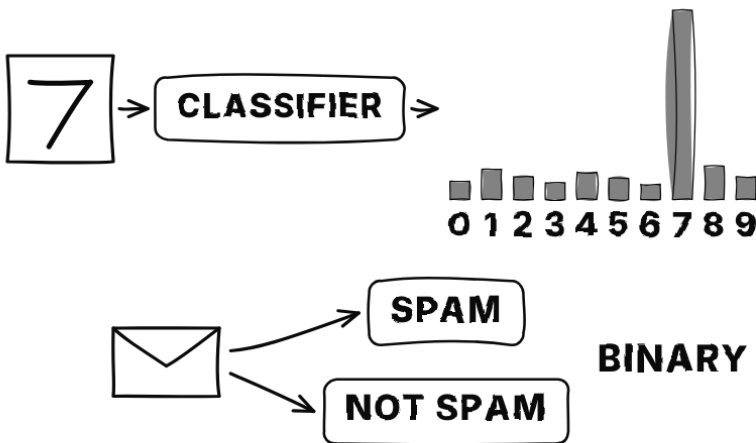
2. The plumber's model predicts 350 dollars for a 3-hour job that was actually billed 420 dollars. What does this job add to the squared error, and what does the miss suggest?

Hint: Subtract, then square.

Answer: The miss is $420 - 350 = 70$, so it adds $70^2 = 4,900$ to the loss. It suggests a factor beyond the hours, such as old pipes or a Sunday job, that the model's features do not capture.

Which one? Classification

Hold up a cheque to your banking app's camera, and a moment later the amount and the name are filled in by themselves. Somewhere in between, the app cut the handwriting into small patches, one character each, and for every patch it answered one question: which character is this? That question is not "how much". There is no sense in which a 7 is "a bit more" than a 1 written badly; the patch is one of a fixed set of characters, and the app has to pick the right one.

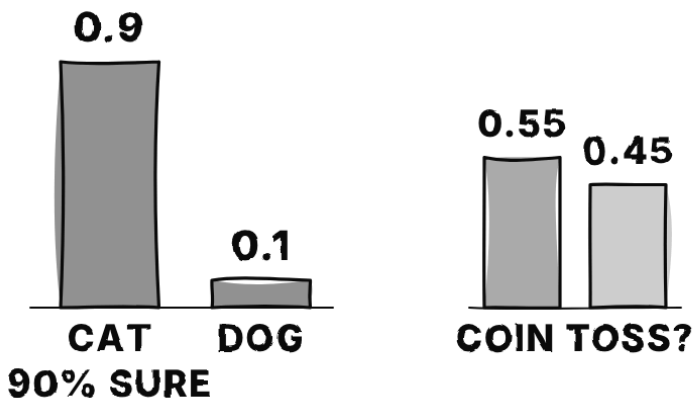


Which one? Ten classes

The book calls these "which one?" problems classification, and sets them beside the "how much?" problems of regression. Many of the tools carry over, but the task is different. The model still looks at features, for instance the brightness values of the pixels of the patch. But instead of a number, it must name a category, also called a class, from a list of options fixed in advance. For handwritten digits there are ten classes, the digits 0 to 9. A model that outputs a number was called a regressor; one that outputs a class is called a classifier.

The simplest case has only two classes, and it is called binary classification. Pictures of animals labelled cat or dog are the book's example. Your email is the everyday one: every message that arrives is put into one of two classes, spam or not spam. When there are more than two classes, the problem is multiclass classification, like reading handwriting, where the classes run from 0 to 9 and on through a, b, c and the rest of the alphabet.

Now a twist that matters later. It turns out to be hard to train a model that can only say a firm "cat" or a firm "dog"; the book promises to explain why when it gets more technical. It is much easier if the model speaks in probabilities. Given a picture, the model gives each class a probability, numbers between 0 and 1 that add up to 1. It might say $P(\text{cat}) = 0.9$, and so $P(\text{dog}) = 0.1$. We read that as: the classifier is 90 percent sure the picture shows a cat. How large the winning probability is tells us how sure the model is; 0.9 is confident, 0.55 is barely more than a coin toss. For training such a model, the book says, the usual loss is not the squared error but one called cross-entropy, whose name will make sense once the book reaches information theory.



LOSS: CROSS-ENTROPY

Probabilities, not verdicts

Then the book makes a point that is easy to miss: the most likely class is not always the one you should act on. Its figure shows a pretty mushroom growing in a back garden. Suppose you have trained a classifier on photos to tell poisonous mushrooms from safe ones, and for this one it says the probability of a death cap is 0.2. So it is 80 percent sure the mushroom is safe. Would you eat it? The authors' answer: you would have to be a fool.



DEATH CAP 0.2

SAFE 0.8

EAT

$$0.2 \times \infty + 0.8 \times 0 = \infty$$

DISCARD ✓

$$0.2 \times 0 + 0.8 \times 1 = 0.8$$

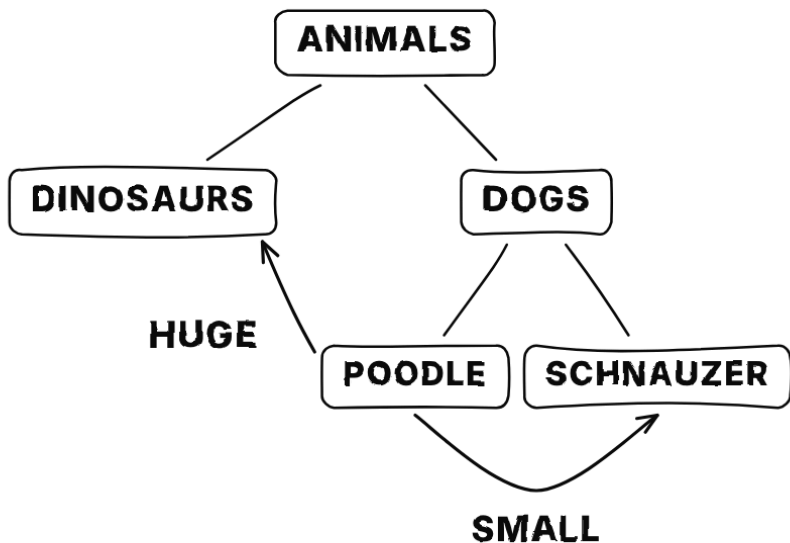
Expected cost of a mushroom

To see why, weigh each action by what it could cost, times how likely that is. Give each outcome a cost. Eating a death cap kills you, a cost so large the book writes it as infinity, ∞ . Eating a safe mushroom costs nothing, 0; throwing away a safe one costs a lost dinner, say 1; throwing away a death cap costs 0. The expected cost of eating is $0.2 \times \infty + 0.8 \times 0 = \infty$. The expected cost of throwing it away is $0.2 \times 0 + 0.8 \times 1 = 0.8$. Less than one lost dinner against certain disaster: you throw it away. The decision depends both on the probabilities and on how good or bad each outcome is. And the book adds that any mushroom expert would confirm it: the one in the figure really is a death cap.

Try it yourself

Take a weather app that says 30 percent chance of rain, and decide whether to carry an umbrella. Give a cost to each outcome: getting soaked in a good suit might cost 10, carrying an umbrella you did not need costs 1. Not carrying it costs on average $0.3 \times 10 = 3$; carrying it costs at most 1. So you take the umbrella, even though "no rain" is the more likely class, exactly as with the mushroom.

Classification can get more intricate still. Sometimes the classes form a family tree, and then not all mistakes are equally bad. If the model must be wrong, we would rather it picked a close relative than something far away. This is called hierarchical classification, and the book points to Linnaeus, who sorted the animals into such a tree of kinds and families. Mistaking a poodle for a schnauzer, two kinds of dog, is a small slip; mistaking a poodle for a dinosaur ought to cost the model dearly.



Near and far mistakes

But which tree counts depends on what the model is for. Rattlesnakes and garter snakes sit close together in the tree of life. Yet a model that calls a rattlesnake a harmless garter snake could get someone killed, so for a hiker's app those two belong far apart. A plant-identifying app on your phone faces the same choice: confusing two kinds of daisy hardly matters, but confusing an edible wild garlic with a poisonous lily of the valley does.

Remember

Classification answers "which one?": it puts each example into one of a fixed set of classes, while regression answers "how much?". With two classes it is binary classification, such as spam or not spam; with more it is multiclass, such as reading digits and letters. Classifiers usually give a probability for each class, and the size of the top probability shows how sure they are; their usual loss is cross-entropy. The most likely class is not always the one to act on: weigh each action by its probabilities and its costs, as with the mushroom that is 80 percent safe and still not worth eating. When classes form a hierarchy, some mistakes are worse than others, and which ones depends on how the model will be used.

Questions

1. Why is reading a handwritten digit a classification problem and not a regression?
 - a. Because the answer must be one of a fixed set of characters, not an amount
 - b. Because the image has too many pixels
 - c. Because digits are always written badly
 - d. Because regression cannot use pixel values as features
2. A classifier gives $P(\text{cat}) = 0.55$ and $P(\text{dog}) = 0.45$. What does this tell you?

- a. The picture is certainly a cat
 - b. The picture shows both a cat and a dog
 - c. It leans towards cat but is barely more sure than a coin toss
 - d. The classifier is broken, since the numbers should be 0 or 1
3. A mushroom classifier is 80 percent sure a mushroom is safe. Why throw it away anyway?
- a. Because the classifier was trained on too few photos
 - b. Because 80 percent is below the error rate
 - c. Because safe mushrooms are rare
 - d. Because the expected cost of eating, with death as one outcome, is far larger than a lost dinner
4. Why might a hiker's snake app treat a rattlesnake and a garter snake as far apart, though they are close relatives?
- a. Because they look nothing alike
 - b. Because confusing them could get someone killed, and the useful hierarchy depends on the purpose
 - c. Because binary classification cannot handle snakes
 - d. Because the tree of life is not a hierarchy

Answers

1. **a)** There is no sense in which a 7 is a bit more than a badly written 1; the patch is one of a fixed set of characters, which is a which-one question.
2. **c)** The size of the winning probability shows how sure the model is; 0.9 is confident, 0.55 is barely more than a coin toss.
3. **d)** Eating has expected cost $0.2 \times \infty + 0.8 \times 0 = \infty$, while throwing it away costs $0.2 \times 0 + 0.8 \times 1 = 0.8$; the decision depends on probabilities and

costs together.

4. **b)** Which tree counts depends on what the model is for; calling a rattlesnake a harmless garter snake could be deadly.

Worked examples

The mushroom decision A classifier gives probability 0.2 that a mushroom is a death cap. Costs: eating a death cap ∞ , eating a safe one 0, discarding a death cap 0, discarding a safe one 1. Find the expected cost of eating and of discarding.

The probability that the mushroom is safe is $1 - 0.2 = 0.8$.

Eating: $0.2 \times \infty + 0.8 \times 0 = \infty$, because any chance of an infinite cost makes the total infinite.

Discarding: $0.2 \times 0 + 0.8 \times 1 = 0.8$.

Choose the action with the smaller expected cost.

Answer: Eating ∞ , discarding 0.8: throw it away

Practice

1. A weather app gives a 30 percent chance of rain. Getting soaked costs 10, carrying an unneeded umbrella costs 1, and staying dry with an umbrella in the rain costs 1. What is the expected cost of each choice?

Hint: Without the umbrella you pay only if it rains; with it you pay 1 whatever happens.

Answer: No umbrella: $0.3 \times 10 + 0.7 \times 0 = 3$. Umbrella: $0.3 \times 1 + 0.7 \times 1 = 1$. Take the umbrella, though no rain is more likely.

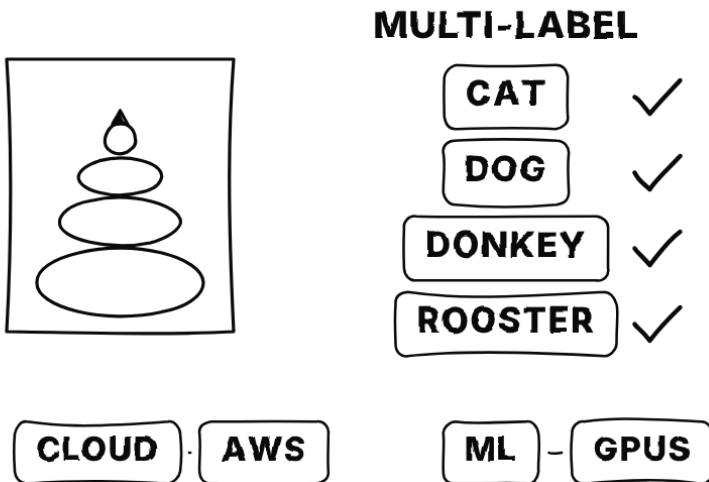
2. A spam filter outputs $P(\text{spam}) = 0.7$ for a message. What is $P(\text{not spam})$, and is this binary or multiclass classification?

Hint: The probabilities of all classes add up to 1.

Answer: $P(\text{not spam}) = 1 - 0.7 = 0.3$, and with two classes it is binary classification.

Many labels at once, and the best first

Show a cat-or-dog classifier a picture from the old fairy tale of the Town Musicians of Bremen. In the book's figure there is a donkey, with a dog on its back, a cat on the dog and a rooster on top, all in front of some trees. The classifier can only answer "cat" or "dog", so whatever it says, it is wrong; the true answer is "cat, and dog, and donkey, and rooster". Making the model more accurate does not help, the book points out: the problem was set up wrongly.



Many labels at once

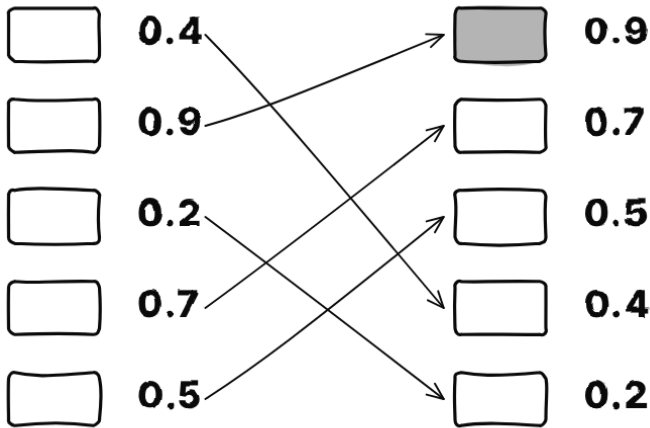
When one example can belong to several classes at the same time, the task is called multi-label classification. The classes are no longer exclusive; the model answers yes or no for each label separately. It is the natural form of tagging. A post on a technology blog might carry five to ten tags, such as "machine learning", "programming languages", "Linux" or "cloud computing". Tags also travel together: a post tagged "cloud computing" often mentions AWS, Amazon's cloud service, and one tagged "machine learning" often mentions GPUs. A good tagger can use those patterns. When a photo app labels one picture "beach", "sunset" and "dog" at once, it is doing multi-label classification.

Sometimes the list of possible tags is huge. The National Library of Medicine in the United States has experts give every article in its PubMed database a set of tags from a list called MeSH, Medical Subject Headings, with about 28,000 entries. Good tags let researchers find everything written on a question, but tagging by hand is slow, and articles usually wait about a year. Machine learning can propose provisional tags until a person checks them, and a competition called BioASQ has run for years on exactly this task.

The next kind of problem is about order, not about classes. When you search the web, thousands of pages may be relevant. The real question is which ones to put at the top for you. A natural way is to give every candidate a score and show the highest scores first.

PAGERANK

SHOWN FIRST



Score, then rank

The book recalls PageRank, the original secret ingredient of Google's search engine, as an early scoring system. Oddly, its score did not depend on what you typed. A simple filter first picked the pages that matched your query, and PageRank then put the more authoritative pages, roughly those that many other pages link to, ahead of the rest. Today, the book says, search engines learn their scores with machine learning and models of how people behave, so the score depends on the query, and whole academic conferences are devoted to the subject.

Try it yourself

Search your email for a word like "invoice" and look at the order of the results. Then search for "invoice March" and see how the order changes. The set of matching messages shrinks, but the ranking also changes, because the score now depends on more of your query. That is the difference between filtering what is relevant and ranking what matters most.

Remember

When an example can belong to several classes at once, like a photo of four animals, the task is multi-label classification, and tagging is its usual form. Tags tend to appear together, and the list of tags can be enormous, like the 28,000 medical headings used to index PubMed. Search is about ranking: give every relevant candidate a score and show the highest first. PageRank scored pages without looking at the query; modern search learns query-dependent scores.

Questions

1. A cat-or-dog classifier is shown the Town Musicians of Bremen. Why does making it more accurate not help?
 - a. The picture has too many trees

- b. The problem is set up wrongly: the picture holds several classes at once
 - c. The classifier was never trained on fairy tales
 - d. Accuracy only matters for regression
2. In multi-label classification, what does the model do for each possible label?
- a. Answers yes or no for that label separately
 - b. Picks the single most likely label
 - c. Ranks the labels and keeps only the top one
 - d. Predicts a number for how many labels there are
3. What is the main question in web search, according to the lecture?
- a. Whether any page at all matches the query
 - b. How many pages the web contains
 - c. Which of the many relevant pages to put at the top
 - d. Which tags to attach to each page
4. How did PageRank differ from the scores search engines learn today?
- a. It used machine learning, while today's scores do not
 - b. It ranked pages by length
 - c. It worked only on medical articles
 - d. Its score did not depend on the query, while today's scores do

Answers

1. **b)** The true answer is cat, dog, donkey and rooster; a model that must pick one class is wrong whatever it says, so the task should be multi-label

classification.

2. a) The classes are no longer exclusive; the model answers yes or no for each label separately.

3. c) Thousands of pages may be relevant; the real question is which to put at the top, which is done by scoring candidates and showing the highest first.

4. d) PageRank's score did not depend on what you typed; a filter picked matching pages, and PageRank put authoritative ones first. Today's learned scores depend on the query.

Practice

1. For each task, say whether it is multiclass or multi-label classification, and why: reading which digit is written in a box; tagging a news article with topics; listing every animal in a farm photo.

Hint: Ask whether one example can have more than one right answer.

Answer: The digit is multiclass: each box holds exactly one digit.

Tagging an article is multi-label: it can be about several topics. The farm photo is multi-label: several animals can be present at once.

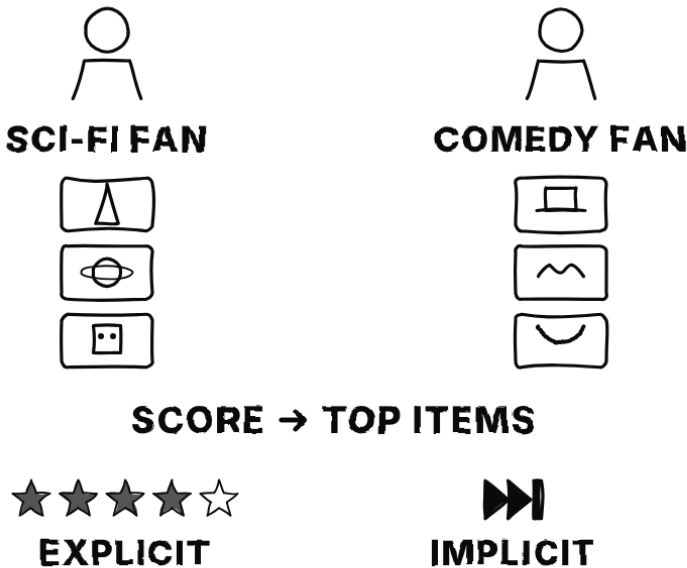
2. Explain in your own words the difference between filtering and ranking in search, using an example of your own.

Hint: One decides what is in the list, the other decides the order.

Answer: Filtering picks which items are relevant at all, for instance all emails containing the word invoice. Ranking gives each relevant item a score and puts the best first, for instance the most recent or best-matching invoice at the top.

The feed that learns you

Open a video app on your phone and on a friend's phone, side by side. The search boxes are the same, but the home pages look nothing alike. Yours may be full of cooking and football, theirs of cats and old films, though nobody typed anything.



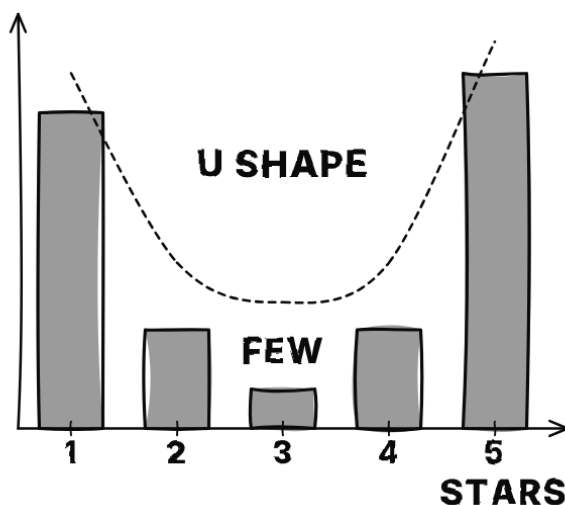
Same app, different feeds

These are recommender systems, the book's next kind of problem. They are close relatives of search: in both, the goal is to show a list of items that matter to the user. The difference is personalization. For films, the authors say, the page for a science fiction fan and the page

for a lover of Peter Sellers comedies should look very different, and the same goes for shops, music and news.

The system learns from what people tell it, in two ways. Explicit feedback is said out loud: star ratings and reviews on Amazon, IMDb or Goodreads. Implicit feedback is read from behaviour, such as skipping a song in a playlist. That is harder to read: a skip may mean you dislike the song, or only that it did not fit the moment. In the simplest version, the system learns to predict a score for each pair of person and item, say the star rating you would give, or the probability that you would buy it. Then, for you, it picks the items with the highest predicted scores and shows those. Real systems use much more, such as detailed records of what you do and what each item is like. The book's figure shows the deep learning books Amazon recommended to one of the authors, Aston.

Recommenders are worth a great deal of money, but the book warns that building them naively on top of a predictive model has deep flaws. The first is that the feedback is censored: we only see some of the opinions. People mostly rate what they feel strongly about. On a scale of one to five stars, there are many ones and many fives, and strangely few threes, not because few films are average but because people rarely bother to rate an average film.



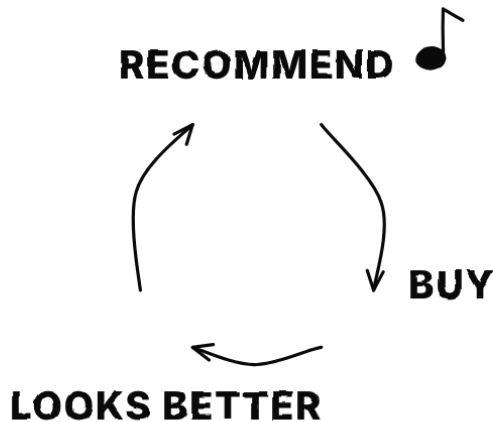
Few people rate the middle

Try it yourself

Open the reviews of any popular product in an online shop and look at the bar chart of star ratings. Count the share of one-star and five-star reviews, and the share of three-star ones. You will usually see the chart dip in the middle, a U shape. That dip is the censored feedback the book is talking about: the quiet middle never speaks.

The second flaw is a loop. What people buy today is partly what the current recommender showed them. Suppose the system pushes one item a little more than others. More people see it, so more people buy

it, so the next round of training takes it to be better, so it is recommended even more. The item rises because it was pushed, not because it was best. A song that a music app keeps putting in your mix becomes "popular" partly because the app put it there. The book says that dealing with censoring, incentives and such feedback loops is still an open research question.



A feedback loop

Remember

Recommender systems are like search, but personalized: different people see different lists. They learn from explicit feedback, such as ratings, and implicit feedback, such as skips, and predict a score for each person and item. Feedback is censored, because people mostly rate what they love or hate, which is why three-star ratings are scarce. Recommendations shape what people buy, so a system can create a feedback loop that keeps pushing the same items.

Questions

1. What mainly separates a recommender system from a search engine?
 - a. A recommender never uses scores
 - b. A recommender personalizes the list for each user
 - c. A recommender only shows items nobody has rated
 - d. A recommender works only for films
2. Why is skipping a song harder to learn from than a one-star rating?
 - a. Skips are never recorded
 - b. Skips only happen on music apps
 - c. A skip may mean dislike, or only that the song did not fit the moment
 - d. A skip always means the user loved the song

3. Why are three-star ratings strangely scarce?
 - a. People rarely bother to rate things they feel only lukewarm about
 - b. Few products are average
 - c. Recommenders delete three-star ratings
 - d. The rating scale has no middle value
4. A shop's recommender pushes one item slightly more than others. What can happen over several rounds of training?
 - a. The item disappears from the shop
 - b. The ratings of the item become censored
 - c. Nothing, since training only uses ratings
 - d. The item is bought more, taken to be better, and pushed even more

Answers

1. **b)** Both show a list of items that matter; the difference is personalization, so a science fiction fan and a Peter Sellers fan see different pages.
2. **c)** Implicit feedback is read from behaviour, and a skip may mean you dislike the song or only that it did not fit the moment.
3. **a)** Feedback is censored: people mostly rate what they feel strongly about, so the quiet middle rarely speaks.
4. **d)** More people see it, so more buy it, so the next round takes it to be better and recommends it even more: a feedback loop.

Practice

1. Sort these into explicit and implicit feedback: a five-star review of a book; stopping a film after ten minutes; adding a song to a favourites

list by pressing a heart; buying a product.

Hint: Explicit feedback states an opinion; implicit feedback has to be guessed from behaviour.

Answer: Explicit: the five-star review and pressing the heart, since both state an opinion directly. Implicit: stopping the film and buying the product, since the opinion has to be guessed from what was done.

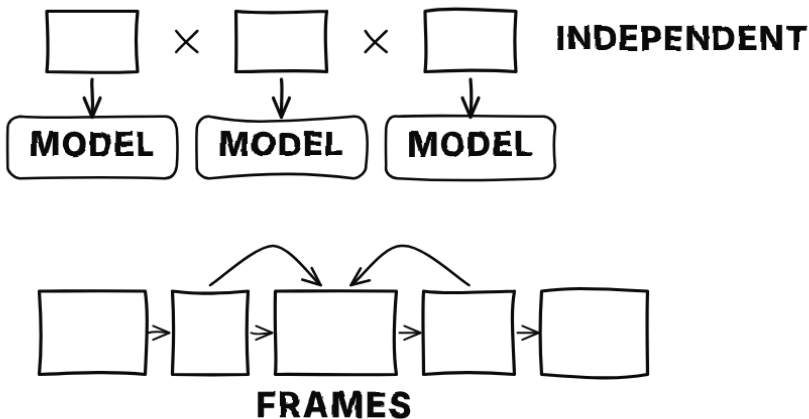
2. Explain in your own words why a recommender trained on what people bought last month may simply learn its own earlier choices.

Hint: Ask where last month's purchases came from.

Answer: Many of last month's purchases happened because the recommender showed those items. Training on them makes the items look better than they are, so the system recommends them even more: it learns from data it shaped itself, a feedback loop.

When order matters: sequences

Pause a football match on a single frame: a player in mid-air, the ball near his foot. Is he shooting, passing, or falling after a foul? From one frame you cannot tell; from the second before and the second after, you know at once. The meaning of each moment depends on the moments around it.



SEQUENCE IN, SEQUENCE OUT

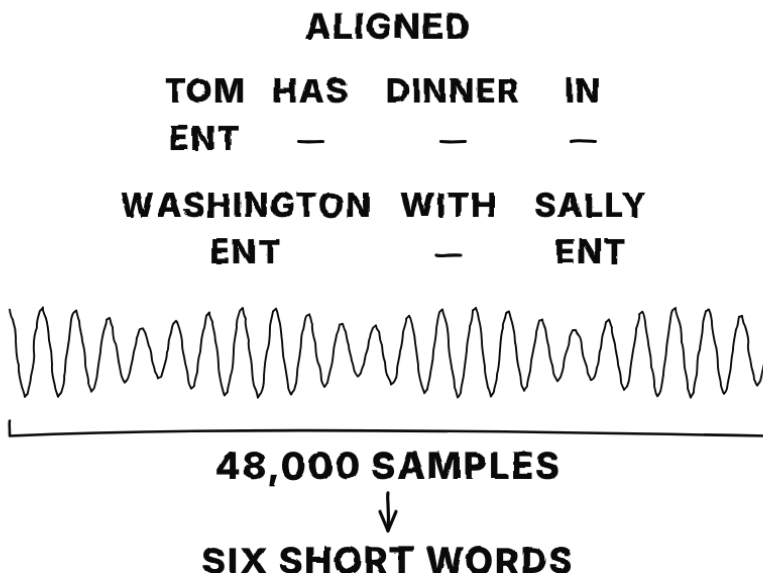
Independent inputs versus sequences

Everything in the course so far ignored that. A house came with a fixed set of features, four numbers, and got one price. A picture of fixed size got one probability per class from a fixed list. A star rating

was predicted from just a user number and a product number. And once the model had answered for one example, it forgot it entirely before the next one arrived. The book says this rests on an assumption: that one example tells us nothing about the next, so there is no reason to remember.

Video breaks that assumption twice over. Each clip may have a different number of frames, so there is no fixed length. And our guess about one frame is much better if we look at the frames before and after it. Language is the same: a sentence can be any length, and a word is understood from the words around it. Medicine too. Picture a model in an intensive care unit that should raise an alarm whenever a patient's risk of dying in the next 24 hours climbs above some level. It would be absurd for it to forget the patient's history every hour and judge only by the latest blood pressure and pulse.

Problems like these are called sequence learning, and the book counts them among the most exciting uses of machine learning. The model has to take in a sequence of inputs, or produce a sequence of outputs, or both. When both the input and the output are sequences whose length can vary, it is called sequence-to-sequence learning; translating sentences and turning speech into text are the standard examples. The book picks out a few special cases.



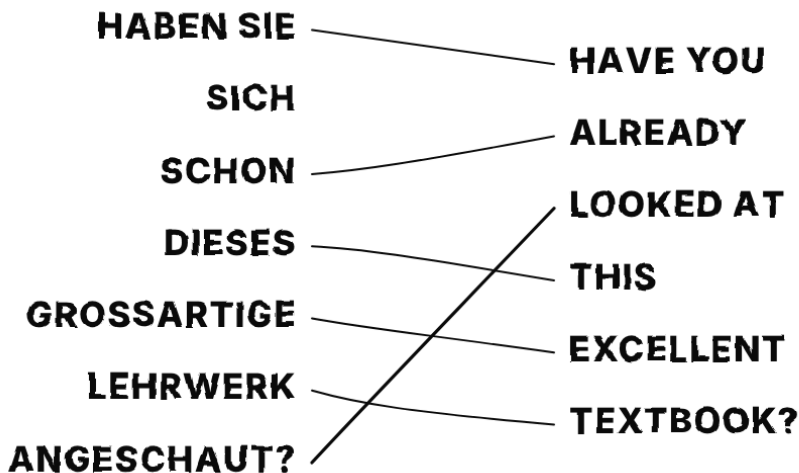
Aligned tags, unaligned audio

The first is tagging and parsing, where every word of a text gets a mark. Here the input and output are aligned: there is one mark per word, in the same order as the words. In part-of-speech tagging, each word gets its grammatical role, such as noun or verb. In a simpler version, each word is marked by whether it belongs to a name of a person, place or organization. The book's example is the sentence "Tom has dinner in Washington with Sally", where Tom, Washington and Sally are marked as names and the other four words are not. Seven words go in, seven marks come out.

The second is automatic speech recognition: a recording goes in, the written words come out. Here the lengths are wildly different. Sound is usually recorded at 8 kHz or 16 kHz, that is, eight or sixteen

thousand samples every second. A sentence of six words spoken in three seconds at 16 kHz is $3 \times 16,000 = 48,000$ samples. So 48,000 numbers must turn into six words, about 8,000 samples per word, and there is no neat one-to-one match between pieces of sound and letters. The book's figure shows the words "Deep Learning" stretched out along a sound wave, each sound smeared over a long stretch of it. People understand speech easily, even over a bad phone line; for computers, the book says, it is a formidable challenge. The voice assistant that understands you over the noise of a street is solving exactly this problem.

The third case is the same problem turned around: text to speech. A short text goes in, a long sound file comes out. When a navigation app reads out "in two hundred metres, turn left", a few words have become tens of thousands of samples.



Translation reorders words

The fourth is machine translation, and it brings a new difficulty. In speech, the sounds and the words at least come in the same order. In translation, the two sentences can have different lengths, and matching pieces can sit in a different order. The book shows it with German, which likes to put the verb at the end. "Haben Sie sich schon dieses grossartige Lehrwerk angeschaut?" means "Have you already looked at this excellent textbook?" Translate it word by word, keeping the German order, and you get "Have you yourself already this excellent textbook looked at?", which is the wrong alignment. The verb "looked at" has to travel from the end of the German sentence to the middle of the English one. A chatbot that translates a message for you has to learn that kind of reordering for every pair of languages.

Try it yourself

Take a short sentence in a language you know besides English, or ask a friend for one. Write it word by word in English, keeping the original order, then write a good English translation underneath. Draw lines between words that mean the same thing. Where the lines cross, the order changed; that crossing is what makes translation unaligned.

The book ends with a glimpse of harder cases. Working out the order in which a person reads a web page is a problem in two dimensions, not along one line. And in a dialogue, deciding what to say next needs knowledge of the world and memory of what was said long before. These, the authors say, are still active areas of research.

Remember

Many earlier tasks had fixed-size inputs and treated every example as independent, forgetting it after answering. Sequence learning handles inputs or outputs that are sequences, such as video, sentences or a patient's history, where context and length matter. In tagging, input and output are aligned: one mark per word, in order. Speech recognition turns tens of thousands of samples per second into a few words; text to speech does the reverse. Machine translation is unaligned: lengths differ, and matching pieces may come in a different order, as with the German verb at the end.

Questions

1. What assumption, made by the earlier tasks, does video break?
 - a. That every example is independent and can be forgotten after answering
 - b. That features are numbers
 - c. That labels are known for training examples
 - d. That models have parameters
2. Why is tagging the words of a sentence called an aligned problem?
 - a. The output is much shorter than the input
 - b. The words can appear in any order
 - c. There is one mark per word, in the same order as the words
 - d. Only names receive marks
3. Why is there no one-to-one match between sound and letters in speech recognition?
 - a. Because the transcript is longer than the recording
 - b. Because tens of thousands of samples become only a few words
 - c. Because speech is always noisy
 - d. Because each word is exactly one sample long
4. What makes machine translation unaligned?
 - a. The output is always shorter than the input
 - b. Every word must be tagged as a name

- c. Translations are made from sound
- d. Lengths differ and matching pieces may come in a different order

Answers

1. **a)** The earlier tasks assumed one example tells nothing about the next, so the model forgot each one; in video, frames vary in number and each is understood from its neighbours.
2. **c)** In tagging, input and output are aligned: one mark per word, in the same order; seven words go in and seven marks come out.
3. **b)** At 16 kHz, three seconds of speech is 48,000 samples turning into six words, about 8,000 samples per word.
4. **d)** In translation the two sentences can have different lengths and matching pieces can sit in a different order, as when the German verb at the end moves to the middle in English.

Worked examples

Samples per word Sound is recorded at 16 kHz, that is, 16,000 samples per second. A sentence of six words is spoken in three seconds. How many samples is that, and about how many per word?

16 kHz means 16,000 samples every second.

Three seconds give $3 \times 16,000 = 48,000$ samples.

Six words share them, so each word gets about $48,000 \div 6 = 8,000$ samples.

So thousands of input numbers correspond to one output word: the lengths are wildly different.

Answer: 48,000 samples, about 8,000 per word

Practice

1. A phone records at 8 kHz. How many samples does a 5-second voice message contain?

Hint: 8 kHz is 8,000 samples every second.

Answer: $5 \times 8,000 = 40,000$ samples.

2. Mark the sentence "Anna flew from Paris to Rome" in the simple name-tagging scheme, and say how many words go in and how many marks come out.

Hint: People and places are names.

Answer: Anna, Paris and Rome are marked as names; flew, from and to are not. Six words go in and six marks come out, because tagging is aligned.

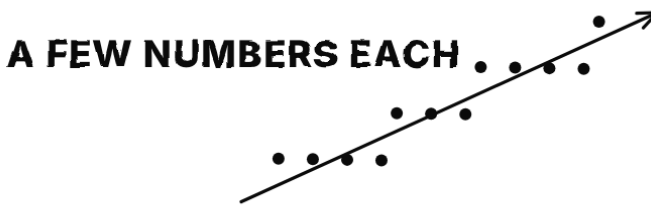
Learning without a teacher

Look at the photo gallery on your phone and search for "beach". Now look at the "people" album, where your phone has gathered every photo of the same friend into one pile. Nobody ever told it your friend's name or which photos belong together; it found the groups by itself. That is a different kind of learning from everything so far.

The book explains the difference with two bosses. Supervised learning is like working for a very bossy boss who stands behind you and says exactly what to do in every situation, until you have learned to copy them. Dull, the authors admit, but easy: spot the pattern and imitate it. The opposite boss has no idea what they want. They drop a huge pile of data on your desk and say: "Do some data science with this!" That sounds vague because it is vague, and a data scientist had better get used to it. Problems without labels like this are called unsupervised learning, and the only limit on the questions you can ask is your imagination.

The book gives a few such questions. Can a few typical examples sum up all the data, for instance sorting a heap of photos into landscapes, dogs, babies, cats and mountain peaks, or sorting users by how they browse? That is clustering. Your phone's "people" album and a music app that groups listeners with similar taste are both doing clustering. Can a few numbers capture what matters about each example? The flight of a ball is well described by its speed, size and mass, and tailors describe a human body for fitting clothes with a handful of measurements. That is subspace estimation, and when the numbers

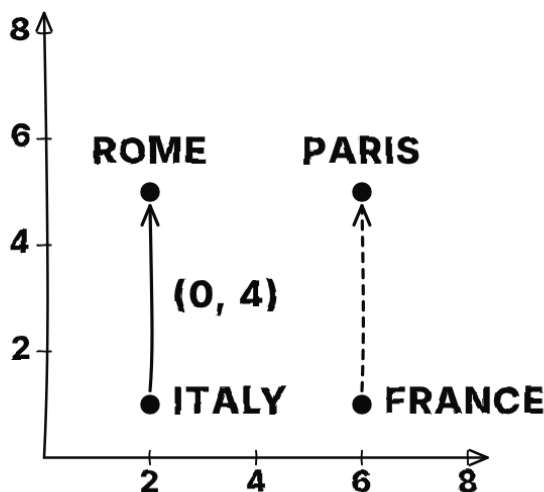
depend on the data in a straight-line way, it is called principal component analysis.



Groups found without labels

A third question is whether words and things can be placed as points in space so that their relations become arithmetic. The book's example is $\text{Rome} - \text{Italy} + \text{France} = \text{Paris}$. Here is a made-up picture of how that can work, with each word as a point with two numbers. Say Rome sits at (2, 5), Italy at (2, 1), and France at (6, 1). Rome minus Italy is $(2 - 2, 5 - 1) = (0, 4)$, a step that means "the capital of". Add that step to France: $(6 + 0, 1 + 4) = (6, 5)$, and if Paris sits at (6, 5), the sum lands exactly on it. Real systems use hundreds of numbers per word, but the idea is the same.

$$\text{ROME} - \text{ITALY} + \text{FRANCE} = \text{PARIS}$$

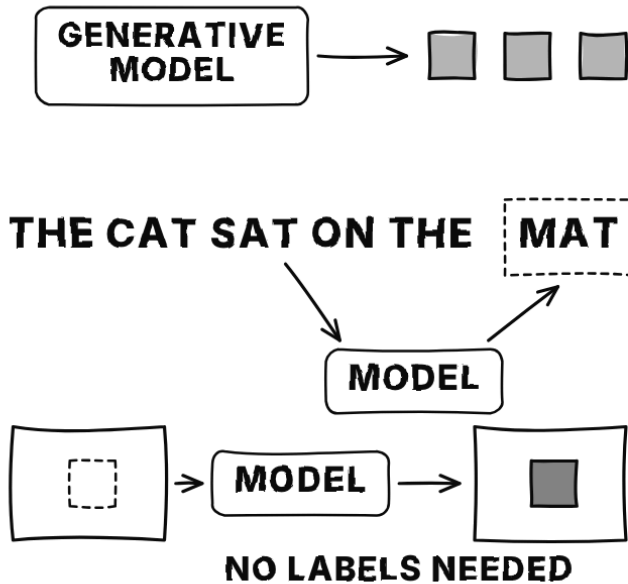


$$\text{Rome} - \text{Italy} + \text{France} = \text{Paris}$$

A fourth question is about causes. Given data on house prices, pollution, crime, location, education and salaries, can we work out from the numbers alone what drives what? The study of causality and of probabilistic graphical models tackles such questions.

The book singles out one recent and exciting line: deep generative models. They learn how the data is spread out, which kinds of examples are common and which are rare, sometimes explicitly and sometimes only implicitly. Once trained, such a model can do two things: say how likely a given example is, or produce brand-new examples that look as if they came from the same data. The early breakthroughs were variational autoencoders and generative adversarial networks; newer ones include normalizing flows and

diffusion models. The apps that draw a picture from a sentence you type are built on diffusion models.



Labels for free

Finally, the book describes self-supervised learning, a way of getting labels for free out of unlabelled data. With text, you hide some words at random and train the model to fill in the blanks from the words around them. Billions of sentences become training examples without anyone labelling anything, because the hidden word is the label. With pictures, a model can learn to tell where two cut-out pieces of the same photo sat relative to each other, to fill in a covered part from the rest, or to say whether two altered pictures came from the same original. The representations such a model learns along the way are then reused: the model is fine-tuned, adjusted a little, for the task we

actually care about. The language model inside a chatbot was first trained in just this way, by guessing missing and next words in enormous amounts of text.

Try it yourself

Take a newspaper paragraph and cover every seventh word with a strip of paper. Read it and try to guess each hidden word, then lift the strips. You will get most of them right, because the words around each gap say a lot about it. Doing exactly this, millions of times, is how a self-supervised model learns a language without a single label.

Remember

Supervised learning has a boss who gives every answer; unsupervised learning gets only a pile of data and must find structure by itself. Clustering finds groups, subspace estimation finds a few numbers that describe each example, and word representations can turn relations into arithmetic, as in Rome – Italy + France = Paris. Deep generative models learn how the data is spread out and can score examples or create new ones. Self-supervised learning makes its own labels from the data, such as hidden words, and the representations it learns are then fine-tuned for other tasks.

Questions

1. What makes a problem unsupervised, in the lecture's picture of the two bosses?
 - a. The boss checks every answer
 - b. You get only a pile of data, with no labels saying what to do
 - c. You copy what the boss does in each situation
 - d. The data comes with a fixed number of features
2. A music app groups listeners with similar taste without being told any groups. Which kind of problem is that?
 - a. Subspace estimation
 - b. Self-supervised learning
 - c. Clustering
 - d. A generative model
3. What can a trained deep generative model do?
 - a. Say how likely an example is, or produce new examples like the data
 - b. Only sort examples into labelled classes
 - c. Only find the causes of the data
 - d. Only fill in hidden words
4. In self-supervised learning on text, where does the label come from?
 - a. From people who read every sentence
 - b. From a boss who gives every answer
 - c. From a separate labelled dataset
 - d. From the hidden word itself, taken out of the text

Answers

1. **b)** The opposite boss drops a huge pile of data on your desk and says do some data science with it; problems without labels like this are unsupervised learning.
2. **c)** Finding a few typical groups that sum up the data, such as listeners with similar taste, is clustering.
3. **a)** Once trained, such a model can say how likely a given example is, or produce brand-new examples that look as if they came from the same data.
4. **d)** You hide some words and train the model to fill in the blanks; the hidden word is the label, so no one has to label anything.

Worked examples

Word arithmetic in a toy map In a made-up two-number map, Rome is at (2, 5), Italy at (2, 1), France at (6, 1) and Paris at (6, 5). Compute Rome – Italy + France and check whether it lands on Paris.

Subtract number by number: Rome – Italy = $(2 - 2, 5 - 1) = (0, 4)$, the step that means "capital of".

Add that step to France: $(6 + 0, 1 + 4) = (6, 5)$.

Paris sits at (6, 5), so the sum lands exactly on Paris.

Answer: (6, 5), which is Paris

Practice

1. In the same kind of toy map, Germany is at (4, 1), Berlin at (4, 5), Spain at (8, 1) and Madrid at (8, 5). Compute Berlin – Germany + Spain.

Hint: First find the step from Germany to Berlin.

Answer: Berlin – Germany = $(0, 4)$; adding it to Spain gives (8, 5), which is Madrid.

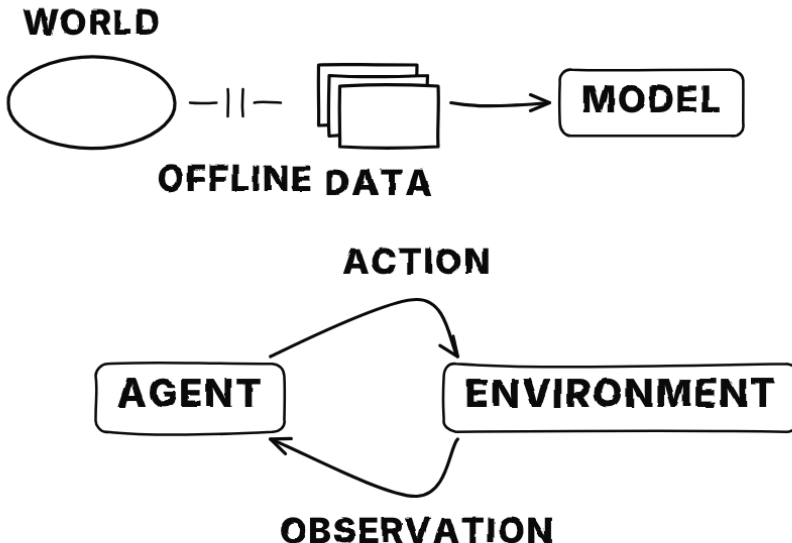
2. Explain in your own words why self-supervised learning can use billions of sentences while supervised learning usually cannot.

Hint: Who has to supply the labels in each case?

Answer: Supervised learning needs a label for every example, usually supplied by people, which is slow and costly. Self-supervised learning makes its labels from the data itself, for example by hiding words and asking the model to guess them, so any amount of text can be used without labelling.

When the world answers back

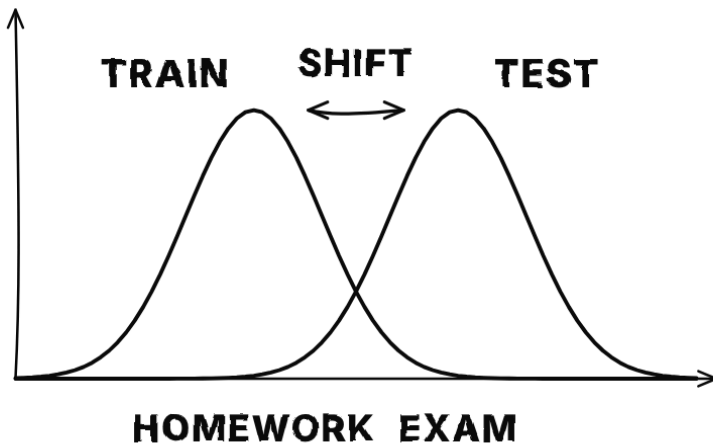
Think about where all our data came from so far. Somebody gathered a big pile of examples once, and then the learning machine was set to work on it, never touching the world again. The book points out that supervised and unsupervised learning hardly ask where the data comes from, or what happens after the model gives its answer. Because all the learning happens after the program has been cut off from its surroundings, this is called offline learning. The book's figure draws it simply: the environment supplies data, the data is collected, and the model learns from the pile.



Offline learning versus acting

This has real charm. We can study pattern recognition on its own, without worrying about a world that moves and reacts. But it is limiting. Anyone who grew up on Isaac Asimov's robot stories imagines intelligent machines that not only predict but act in the world. So we want to think of agents, not just models that predict, and of choosing actions, not just making predictions. The difference is that actions change the world. An agent's actions change what it will see next, so it cannot learn once from a fixed pile and be done. A robot vacuum is the everyday case: where it drives decides which part of the room it sees next, and so what it can learn about the room.

Once the world responds, new questions arise, and the book lists a few. Does the environment remember what we did before? Does it want to help us, like a person speaking clearly into a speech recogniser? Does it want to beat us, like spammers who keep rewriting their emails to slip past spam filters? Your spam filter really does play this game: every time it learns a trick, the senders change the trick. And does the world change over time, so that the future does not look like the past, either by itself or because of the very tools we deploy?



Trained on one, tested on another

All these questions lead to one problem, distribution shift: the data the model is tested on, or used on, differs from the data it was trained on. The authors give an example many students know. The homework was set by the teaching assistants, so you practised on their kind of questions. Then the exam was written by the lecturer, and the questions looked different. You trained on one distribution and were tested on another.

Try it yourself

Ask a friend to write ten arithmetic questions in their own style, and another friend ten more in a different style, for example word problems instead of bare sums. Practise only on the first set until you are fast, then time yourself on the second. You will usually slow down on the unfamiliar style, even though the arithmetic is the same. That slowdown is distribution shift in miniature.

Remember

Offline learning gathers a pile of data once and learns from it without ever acting in the world again. That is convenient, but an agent's actions change the world and so change what it sees next. The environment may remember, help, fight back, or drift over time. When the data at use time differs from the training data, we have distribution shift, like homework set by assistants and an exam set by the lecturer.

Questions

1. Why is offline learning not enough for an agent such as a robot vacuum?
 - a. Because the agent's actions change what it will see next
 - b. Because robots cannot store data

- c. Because offline learning only works on text
 - d. Because an agent never makes predictions
2. Spammers keep rewriting their emails to slip past filters. What kind of environment is this?
- a. One that wants to help us
 - b. One that wants to beat us
 - c. One that never changes
 - d. One that forgets everything
3. You practised on homework set by the teaching assistants, then faced an exam written by the lecturer. What is this an example of?
- a. Overfitting to a test set
 - b. Offline learning
 - c. Distribution shift
 - d. Clustering

Answers

1. **a)** Actions change the world: where the robot drives decides which part of the room it sees next, so it cannot learn once from a fixed pile.
2. **b)** The book's example of an environment that wants to beat us is spammers rewriting emails to get past spam filters.
3. **c)** You trained on one distribution, the assistants' questions, and were tested on another, the lecturer's: distribution shift.

Practice

1. A model that predicts what people buy online was trained on data from before a big holiday sale. Explain how distribution shift might affect it during the sale.

Hint: Does the data during the sale look like the data it was trained on?

Answer: During the sale people buy different things, in different amounts and at different prices, so the data at use time differs from the training data. The model's predictions may be poor because it learned patterns that no longer hold.

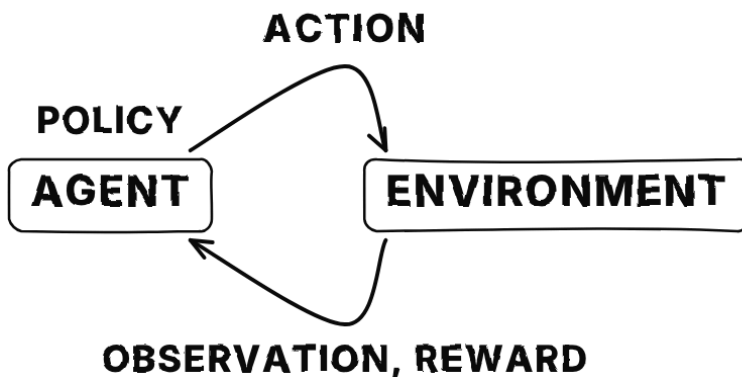
2. For each environment, say whether it helps, fights back, or simply changes over time: a person speaking clearly into a voice assistant; fraudsters changing their tricks to beat a bank's detector; fashion trends drifting from year to year.

Hint: Ask what the environment wants, if anything.

Answer: The clear speaker helps the system. The fraudsters fight back, adapting to beat the detector. Fashion simply changes over time, so future data does not look like the past.

Learning by acting: reinforcement learning

Teach a dog to sit, and you do not explain anything. You wait until it happens to sit, give it a treat, and repeat. Over many tries the dog works out which of its actions bring treats. It is never told the right answer; it only finds out, afterwards, whether things went well.



POLICY: OBSERVATION → ACTION

The reinforcement learning loop

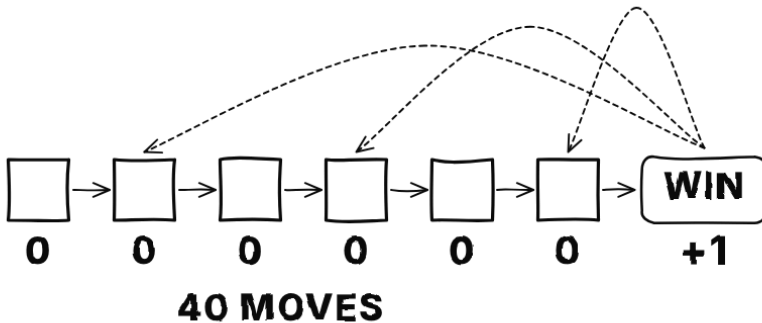
When machine learning is used to build an agent that acts in a world, the book says, you will most likely end up with exactly this kind of learning, called reinforcement learning. Its uses include robots,

dialogue systems and players for video games. Combined with deep learning it became deep reinforcement learning, and it made headlines twice. A program called a deep Q-network learned to beat people at Atari video games from nothing but the picture on the screen. And AlphaGo beat the world champion at the board game Go.

The setting is simple and very general. The agent and the environment take turns, step after step. At each step the agent receives an observation, what it can see of the world. It chooses an action, which is passed back to the world through some mechanism, sometimes called an actuator, such as a robot's motor. Then it receives a reward, a number saying how good that turn was, and the loop starts again with the next observation. The book's figure draws this as a circle: the agent sends actions to the environment, and the environment sends back observations and rewards. What the agent does is governed by its policy, simply a rule, a function, that maps what it observes to what it does. The aim of reinforcement learning is to find good policies. A robot vacuum that learns the layout of your flat is in this loop: it senses the room, chooses where to drive, and is rewarded for floor cleaned.

The framework is so general that it even swallows supervised learning. Take a classification task with ten classes. Give the agent ten actions, one per class, and let the environment pay a reward taken straight from the loss function of the original problem, so that the reward scores each choice just as the loss scored each prediction. Choosing actions to earn reward is then the same as classifying.

CREDIT?



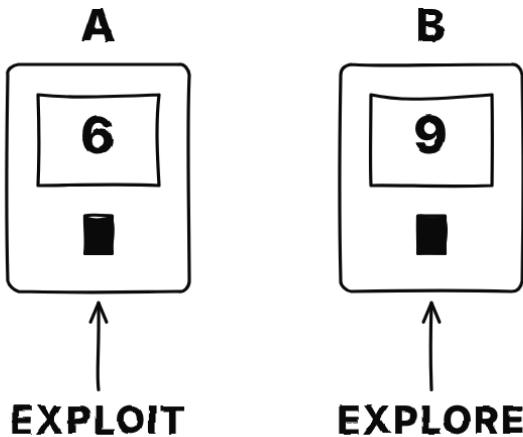
Reward only at the end

And it handles things supervised learning cannot. In supervised learning, every training input comes with its correct label. In reinforcement learning, nobody tells the agent the best action for each observation; it just gets some reward. Worse, the world may not say which actions earned that reward.

The book uses chess to show the trouble. The only real reward comes at the very end, say +1 for a win and -1 for a loss. A game might last 40 moves, and the +1 arrives only after the last of them. Which of those 40 moves won the game, and which nearly lost it? Working out which actions deserve the credit or the blame is called the credit assignment problem. The book's human example is an employee promoted on October 11: the promotion reflects many good choices

over the past year, and to be promoted again one has to work out which of them mattered.

A second difficulty is partial observability: what the agent sees now may not tell it where it is. Imagine your cleaning robot stuck in one of several identical cupboards in your house. From inside, they all look the same; to find which one it is in, it must remember what it saw before it went in.



Exploit or explore

A third is the choice between using and searching. At any moment the agent may know one good policy, while better ones exist that it has never tried. It can exploit, sticking with the best strategy it knows, or

explore, trying others and giving up some reward now in exchange for knowledge.

Try it yourself

Imagine two coffee machines at work that you have not tried. On Monday you try machine A and the coffee is fine, say 6 out of 10. If you only exploit, you drink from A for ever and never learn that machine B makes a 9. If you spend Tuesday exploring B, you risk a bad cup that day, but every day after can be a 9. That trade between a sure 6 now and a possible 9 later is exactly the explore-or-exploit choice.

All of this together, actions shaping later observations, rewards seen only for the actions taken, a world seen fully or only in part, is a great deal to handle at once. And not every practical problem has all of it. So researchers study special cases, and the book names three. When the agent sees the whole state of the world, the problem is called a Markov decision process. When the state does not depend on what the agent did before, it is a contextual bandit problem. And when there is no state at all, only a set of actions whose rewards are unknown at first, it is the classic multi-armed bandit problem, named after a row of slot machines, each with its own unknown payout; our two coffee machines are a tiny one. A news app that tries out two headlines on different readers to see which gets more clicks is solving a bandit problem.

Remember

In reinforcement learning an agent acts step by step: it observes, chooses an action, and receives a reward. Its policy maps observations to actions, and the goal is a good policy. Unlike supervised learning, it is never told the right action, and the reward may come late, so it must solve credit assignment, as in chess with +1 or -1 only at the end. It may see only part of the world, and it must balance exploiting what works against exploring what might work better. Special cases include Markov decision processes, contextual bandits and multi-armed bandits.

Questions

1. What does a reinforcement learning agent's policy do?
 - a. It maps what the agent observes to what it does
 - b. It decides how large the reward is
 - c. It labels each training example
 - d. It stores every observation forever
2. A chess program wins a 40-move game and receives +1 at the end. What problem must it solve to learn from this?
 - a. Partial observability
 - b. Distribution shift
 - c. Credit assignment: which of the moves deserve the credit
 - d. Clustering the moves into groups

3. Why does a cleaning robot stuck in one of several identical cupboards face partial observability?
 - a. Its reward is too small
 - b. What it sees now does not tell it which cupboard it is in
 - c. It has too many actions to choose from
 - d. It is told the correct action at every step
4. A problem has no state at all, only a set of actions whose rewards are unknown at first. What is it called?
 - a. A Markov decision process
 - b. A contextual bandit problem
 - c. Supervised learning
 - d. A multi-armed bandit problem

Answers

1. **a)** What the agent does is governed by its policy, a function that maps what it observes to what it does; the aim is to find good policies.
2. **c)** The reward arrives only after the last move, so the learner must work out which actions deserve the credit or the blame.
3. **b)** From inside, the cupboards all look the same; to know which one it is in, it must remember what it saw before going in.
4. **d)** With no state, only actions whose rewards are unknown at first, it is the classic multi-armed bandit problem.

Worked examples

Is exploring worth it? Over 10 working days you drink one coffee a day. Machine A, tried on day 1, scores 6. Machine B is untried. Compare total scores if you always use A, and if you try B on day 2

and then use the better machine, when B turns out to score 9, and when it turns out to score 2.

Always exploiting A: $10 \times 6 = 60$.

Exploring B on day 2 when B scores 9: $6 + 9 + 8 \times 9 = 87$.

Exploring B on day 2 when B scores 2: $6 + 2 + 8 \times 6 = 56$.

Exploring costs at most 4 points if B is bad, but gains 27 if B is good: giving up a little short-term reward can buy valuable knowledge.

Answer: Always A: 60; exploring: 87 if B is good, 56 if B is bad

Practice

1. Name the observation, the action and the reward for a robot vacuum cleaning a flat.

Hint: What does it sense, what does it do, and what is it rewarded for?

Answer: The observation is what its sensors tell it about the room around it; the action is where it drives next; the reward is the amount of floor it cleans.

2. Explain in your own words how a classification task with ten classes can be turned into reinforcement learning.

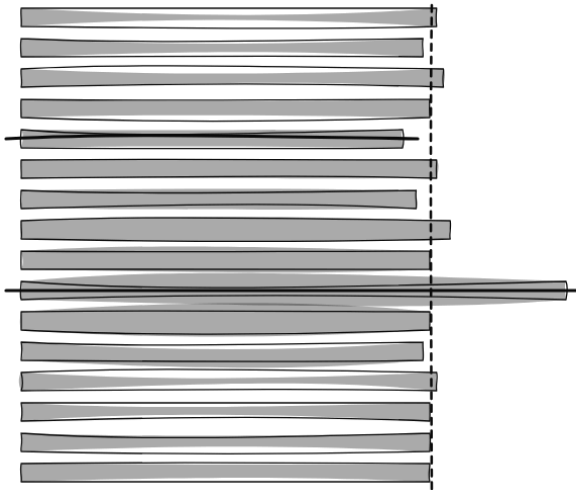
Hint: Make each class an action.

Answer: Give the agent ten actions, one per class, and let the environment pay a reward taken from the loss function of the original problem. The reward then scores each choice of class just as the loss scored each prediction, so learning to act well is learning to classify.

Old roots: from church doors to neurons

Picture a Sunday in a sixteenth-century German town. The service ends, and as people leave the church, sixteen grown men are asked to stand in a row, heel to toe. The total length of their sixteen feet is measured and divided by sixteen. The result is declared to be one "foot", the town's unit of length.

30.1, TRIMMED



Sixteen feet, one foot

The book takes this scene from a geometry book by Jacob Köbel, who lived from 1460 to 1533, and its figure shows the men lined up. Its

point is that learning from data is far older than computers. People have always wanted to read data and predict what comes next, and that wish lies at the root of much of science and mathematics. Even this simple recipe was improved later, to cope with odd feet: the men with the shortest and the longest foot were sent away, and only the other fourteen were averaged. That is one of the earliest known trimmed means, an average that ignores the extremes.

Let us see why that helps, with made-up numbers. Say fifteen men have feet close to 30 centimetres, adding up to 450, and the sixteenth, a giant, has a foot of 40. The plain average is $(450 + 40) / 16 \approx 30.6$ centimetres, pulled up by one man. Drop the longest foot, 40, and the shortest, say 28, and average the remaining fourteen: $(450 - 28) / 14 \approx 30.1$. One unusual foot no longer moves the town's unit. Ratings in an app store work on the same worry: a few furious or fake reviews can drag an average down, which is why some sites trim or weight them.

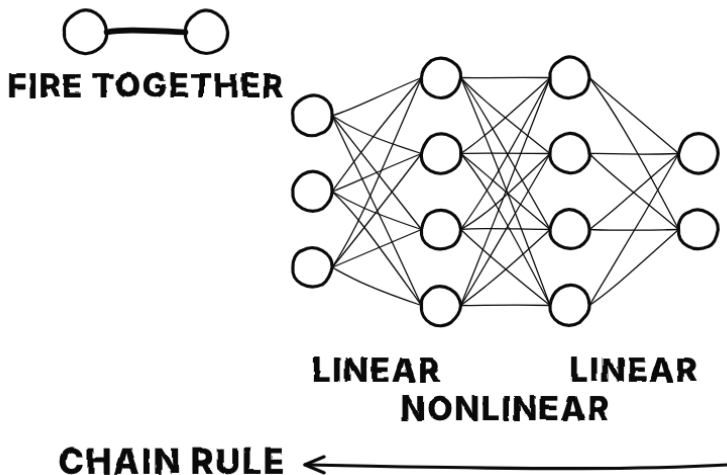
The book names more ancestors. Jacob Bernoulli, 1655 to 1705, and Carl Friedrich Gauss, 1777 to 1855, each gave their names to a probability distribution still used everywhere. Gauss also invented the method of least squares, choosing the line that makes the sum of squared misses smallest, still used for everything from insurance to medical diagnosis. Such tools strengthened experimental science. Ohm's law for a resistor, $V = R \times I$, voltage equals resistance times current, is a perfect linear model: double the current through a 10 Ω resistor from 0.5 A to 1 A, and the voltage doubles from 5 to 10 volts.

Statistics really took off once data could be collected in quantity. Ronald Fisher, 1890 to 1962, built much of its theory and applied it to genetics; methods and ideas named after him are still basic to the

field. His Iris dataset of 1936, measurements of three kinds of iris flower, is still used to demonstrate learning algorithms. The book adds a warning: Fisher also promoted eugenics, a reminder that data science has been put to morally dubious uses for as long as to good ones.

Other roots lie in Claude Shannon's theory of information and in Alan Turing's theory of computation. Turing asked "can machines think?" and proposed what we now call the Turing test: a machine counts as intelligent if a person exchanging text with it finds it hard to tell its replies from a human's. Many people chatting with a modern chatbot would find that test hard to judge, which is part of why it is still argued over.

Still other roots come from the study of the brain and the mind. Humans are clearly intelligent, so scholars asked whether that ability could be explained and copied. Donald Hebb, 1904 to 1985, proposed in his book *The Organization of Behavior* that neurons learn by positive reinforcement: connections that are used together grow stronger. This Hebbian rule inspired Rosenblatt's perceptron, an early learning machine, and it underlies the idea behind the training methods of today's deep learning: reinforce what works, weaken what does not, and so arrive at good settings of the network's parameters.



Layers and backpropagation

That is where the name neural network comes from. For more than a century, since models by Alexander Bain in 1873 and James Sherrington in 1890, as the book names them, researchers have tried to build circuits that resemble networks of neurons. Over time the link to biology has grown loose, but the name has stayed. Two principles are found in most networks today. First, layers that alternate between simple linear steps and nonlinear ones. Second, the chain rule of calculus, known in this field as backpropagation, which adjusts every parameter in the whole network at once.

Try it yourself

Hold a ruler and ask a friend to drop it, catching it as fast as you can, ten times. At first your catches are slow and uneven; after a few tries they get steadier. Nobody told your brain which nerve connections to strengthen; the ones that led to a good catch were simply used again. That is the spirit of Hebb's rule, which inspired the first learning machines.

After quick early progress, neural network research stalled from about 1995 to 2005, for two reasons. Training a network takes a great deal of computation, and at the end of the last century memory was plentiful but computing power was scarce. And datasets were small. Fisher's Iris data from 1936, with 150 flowers, was still a common test, and the MNIST collection of 60,000 handwritten digits was thought huge. With little data and little computing power, other methods, kernel methods, decision trees and graphical models, simply worked better in many uses. They did not need weeks of training, and they gave predictable results backed by strong theory.

Remember

Learning from data is centuries old: Köbel's sixteen feet, averaged and later trimmed of the extremes, is an early estimate. Bernoulli, Gauss, Fisher, Shannon and Turing all laid foundations, and Gauss's least squares and linear models like Ohm's law are still in use. Hebb's idea that neurons learn by reinforcement led to the perceptron and gave neural networks their name. Today's networks rest on alternating linear and nonlinear layers and on backpropagation. From about 1995 to 2005 neural networks stalled, because computing was scarce and datasets small, and simpler, faster methods did better.

Questions

1. Why did Köbel's recipe send away the men with the shortest and longest feet?
 - a. So that a single unusual foot would not move the town's unit much
 - b. Because only fourteen men fitted on the church steps
 - c. To make the division easier
 - d. Because the longest feet belonged to visitors
2. What did Turing propose as a test of whether a machine is intelligent?
 - a. That it can beat a human at chess
 - b. That it learns from reinforcement

- c. That a person exchanging text with it finds it hard to tell its replies from a human's
 - d. That it can measure feet accurately
- 3. Which two principles are found in most neural networks today?
 - a. Trimmed means and least squares
 - b. Alternating linear and nonlinear layers, and backpropagation
 - c. Hebb's rule and the Turing test
 - d. Kernel methods and decision trees
- 4. Why did kernel methods and decision trees do better than neural networks from about 1995 to 2005?
 - a. They were inspired by the brain
 - b. They needed more data
 - c. They were invented later
 - d. With little data and computing power they worked better, trained fast and had strong theory

Answers

- 1. **a)** Dropping the extremes before averaging, a trimmed mean, stops one odd foot from pulling the unit up or down.
- 2. **c)** The Turing test counts a machine as intelligent if a person exchanging text with it finds it hard to tell its replies from a human's.
- 3. **b)** Most networks have layers that alternate simple linear and nonlinear steps, and use the chain rule, backpropagation, to adjust all parameters at once.
- 4. **d)** With little data and little computing power, they simply worked better, needed no weeks of training, and gave predictable results backed by strong theory.

Worked examples

Plain and trimmed average Fifteen men's feet add up to 450 centimetres; a sixteenth foot is 40 centimetres. The shortest foot is 28 centimetres. Compute the plain average of all sixteen, and the trimmed mean after dropping the 40 and the 28.

Plain average: $(450 + 40) \div 16 = 490 \div 16 \approx 30.6$ centimetres.

Trimming removes the 40 and the 28; the 28 is among the fifteen, so the remaining fourteen add up to $450 - 28 = 422$.

Trimmed mean: $422 \div 14 \approx 30.1$ centimetres.

The single long foot no longer pulls the result up.

Answer: Plain average about 30.6 cm; trimmed mean about 30.1 cm

Practice

1. Using Ohm's law $V = R \times I$, what voltage is across a 20-ohm resistor carrying 0.3 amps? What happens to the voltage if the current is tripled?

Hint: Multiply resistance by current; the law is linear.

Answer: $V = 20 \times 0.3 = 6$ volts. Tripling the current to 0.9 amps triples the voltage to 18 volts, because the model is linear.

2. Give the two reasons neural network research stalled between about 1995 and 2005, and say how each limited the networks.

Hint: One is about computers, one about data.

Answer: Computing power was scarce, and training a network takes a great deal of computation, so networks could not be trained well. And datasets were small, the Iris data and the 60,000 digits of MNIST being typical, so large networks had too little to learn from.

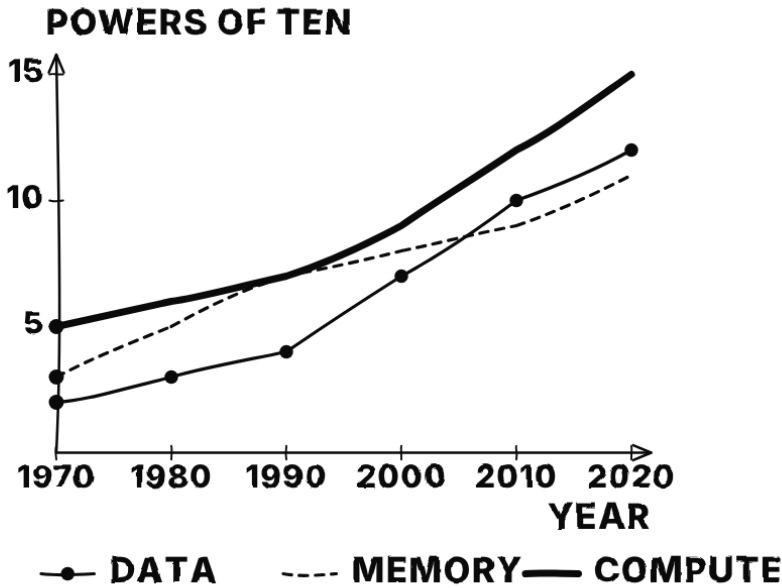
The road to deep learning

Think of the phone in your pocket next to a desktop computer from 1990. Your phone holds thousands of times more photos than that computer could store, and it does more calculations in a second than that machine did in hours. Now think of how many photos, messages and clicks the whole world produces every day. The book says that this double change, more data and far more computation, is what woke neural networks from their long sleep.

Several things came together. The World Wide Web, and companies serving hundreds of millions of people online, produced data in amounts nobody had seen. Cheap, good sensors spread everywhere, from cameras to microphones. Storage became cheaper and cheaper, a trend the book calls Kryder's law, and computation too, following Moore's law. And one piece of hardware changed the game: the graphics processor, or GPU, built for computer games. A GPU does thousands of simple calculations side by side, which happens to be exactly what training a neural network needs. Suddenly, models that had looked impossibly expensive to compute were within reach. The same kind of chip that draws the scenery in a video game now trains the models behind chatbots and voice assistants.

The book shows the change in a table, decade by decade, with three columns: how big a typical dataset was, how much memory a computer had, and how many calculations it could do per second. In 1970, a dataset meant about a hundred examples, Fisher's Iris flowers; memory was one kilobyte; a processor did about a hundred thousand calculations a second. By 1990, datasets held about ten

thousand examples, such as scanned handwriting; memory was ten megabytes; the processor did ten million calculations a second. By 2010, datasets held about ten billion examples, from online advertising; memory was one gigabyte; a GPU did a trillion calculations a second. And by 2020, a dataset from a social network could hold a trillion examples, with a hundred gigabytes of memory and a quadrillion calculations a second.



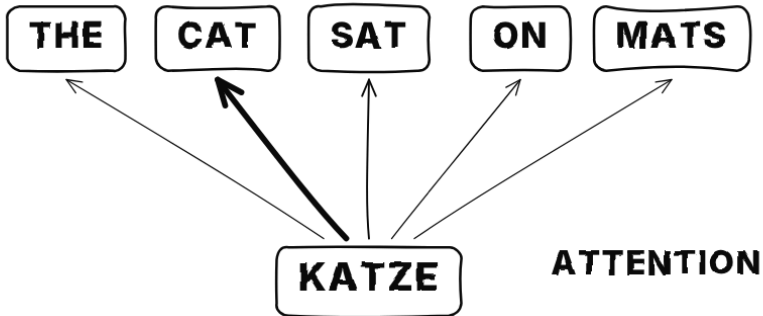
Data, memory and compute, 1970–2020

Let us read the table across the whole fifty years. Data grew from a hundred examples to a trillion, that is from 10^2 to 10^{12} , a factor of 10^{10} , ten billion. Memory grew from one kilobyte to a hundred gigabytes, a factor of 10^8 , a hundred million. Calculations per second grew from a hundred thousand to a quadrillion, again a factor of 10^{10} .

So memory fell a hundred times behind the growth of data, while computing power, the book says, grew even faster than the datasets. The book draws the conclusion: models must use memory sparingly, but they can afford to spend far more calculation on tuning their parameters. That shifted the sweet spot of machine learning away from linear models and kernel methods, which suited the old conditions, towards deep neural networks. It is also why many classic tools of deep learning, such as multilayer perceptrons, convolutional networks, long short-term memory and Q-learning, were in effect rediscovered in the 2010s after lying dormant for years.

But the authors stress that progress was not only old ideas on new hardware. They compare the last decade to the Cambrian explosion, the moment in the history of life when new kinds of animals appeared in a rush. New ideas multiplied, and the book lists some of them, admitting that the list barely scratches the surface.

DROPOUT: NOISE IN TRAINING



TRANSFORMER: ATTENTION ONLY

Attention as a pointer

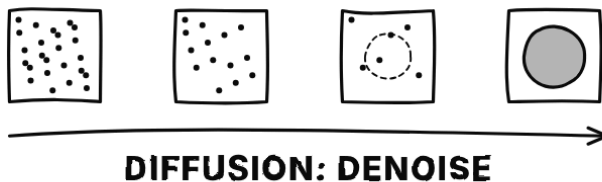
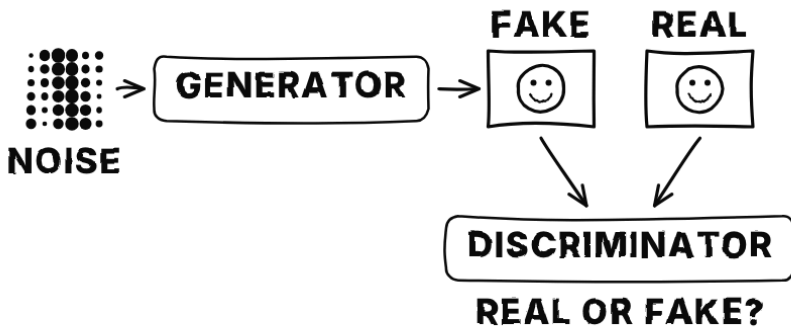
The first is better control of overfitting, with methods such as dropout. During training, random noise is injected throughout the network, for example by switching off parts of it at random, so it cannot simply memorise the training examples.

The second is attention. Statistics had long struggled with a puzzle: how to give a system more memory and more complexity without adding more learnable parameters. Older translation models had to squeeze a whole sentence into one fixed-size summary before they could start translating it, and long sentences suffered. Attention lets the model keep a kind of learnable pointer instead, looking back at the relevant part of the input as it goes. Accuracy on long sentences rose

sharply, because the model no longer had to remember everything before starting to produce the translation.

The third grew out of the second: the Transformer, a design built entirely on attention. Its striking property is how it scales: it keeps getting better as the data, the model and the computing budget grow. Transformers now succeed in language, vision, speech recognition, reinforcement learning and more. The book mentions a single Transformer trained on text, images, robot joint movements and button presses that can play Atari games, describe pictures, chat and control a robot arm.

The fourth is language models, which learn the probability of text and so can predict text from other text. Scaling up data, model and computation has unlocked more and more abilities. ChatGPT, which the book names, is a language model tuned to follow people's intentions; you can talk with it to fix a bug in your code or to draft a story.



Generator against discriminator

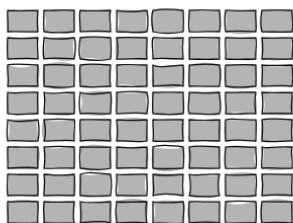
The fifth is designs that work in several stages, such as memory networks and the neural programmer-interpreter. They let a network change its own inner state again and again, step by step, much as a processor updates memory during a calculation, so it can follow a chain of reasoning.

The sixth is a new way to generate data, generative adversarial networks. Older methods had to write down a proper probability formula and then find a way to draw samples from it, which limited them. A generative adversarial network replaces the sampler with any adjustable program, the generator, and pits it against a second network, the discriminator, which tries to tell fake examples from real ones. The generator is tuned until the discriminator can no longer tell

the difference. The results include videos of galloping zebras and convincing faces of people who never existed; even amateurs can turn a rough sketch into a photo-like picture.

The seventh is diffusion models. Diffusion slowly adds random noise to a picture until only noise is left; a diffusion model learns to run that process backwards, removing noise step by step, so it can start from pure noise and end with a new picture. These models have started to replace adversarial networks, for instance in DALL-E 2 and Imagen, which make images from a written description. The apps that turn a sentence like "a cat in a spacesuit" into a picture are built on such models.

The eighth is training on many GPUs at once. Often one GPU is not enough for all the data. The workhorse of training, stochastic gradient descent, works on small batches of examples at a time, called minibatches, but small batches waste a GPU's power. So people spread the work: with 1,024 GPUs, each handling a minibatch of 32 images, one step processes $1,024 \times 32 = 32,768$ images, about thirty-two thousand. Later work pushed this to sixty-four thousand images per step and cut the training of a well-known image model, ResNet-50, on the ImageNet collection to under seven minutes, where it had once taken days.



1,024 GPUS

$$1,024 \times 32 = 32,768$$

IMAGES PER STEP

DAYS



MINUTES

THEANO → TENSORFLOW → PYTORCH, JAX

Many GPUs, one big batch

The ninth is reinforcement learning, which also gained from parallel computing. Computers reached superhuman play at Go, Atari games and Starcraft, and did well in physics simulations, wherever a simulator of the environment exists. The reason is simple: reinforcement learning works best with plenty of examples of state, action and reward, and a simulator can produce as many as you want.

The tenth is software. Deep learning frameworks spread the ideas. The first generation of open-source tools was Caffe, Torch and Theano, and many landmark papers were written with them. They were replaced by TensorFlow, often used through its simpler layer Keras, and by CNTK, Caffe 2 and Apache MXNet. The third generation is imperative: you describe a model in ordinary Python,

step by step, the way you would write any calculation with the NumPy library; Chainer started this, and PyTorch, the Gluon interface of MXNet, and JAX followed.

Try it yourself

Take a sheet of paper and write the fifty-year growth factors from the table side by side: data, ten billion times; memory, a hundred million times; calculations, ten billion times. Now imagine your home grew the same way: a room that became a hundred million times bigger, filled with ten billion times more boxes. You could not keep every box in view; you would have to be clever about what you keep, and use your vastly faster legs to fetch the rest. That is the situation the book describes, and deep neural networks fit it better than the older methods.

Remember

Neural networks came back because data exploded, through the web, sensors and cheap storage, and computation became cheap, above all through GPUs built for games. Over fifty years data and computation grew about ten billion times, memory only about a hundred million times, so models now trade memory for computation, which favours deep networks. New ideas mattered too: dropout against overfitting, attention and the Transformer, language models, adversarial networks and diffusion models for generating data. Training spread across many GPUs, with steps of tens of thousands of images, cut training from days to minutes, and simulators fed reinforcement learning. Open-source frameworks, from Theano and Caffe to TensorFlow and PyTorch, carried the ideas to everyone.

Questions

1. Why did the graphics processor, built for games, suit neural networks so well?
 - a. It has more memory than any other chip
 - b. It does thousands of simple calculations side by side, which training needs
 - c. It was designed to store large datasets
 - d. It runs Python code faster

2. Memory grew much more slowly than data, while computation grew fast. What did this mean for models?
 - a. They had to store every example
 - b. They had to stop using GPUs
 - c. They had to use memory sparingly but could spend far more calculation tuning parameters
 - d. They had to become smaller and simpler, like linear models
3. What problem did attention solve for translation models?
 - a. They had to squeeze a whole sentence into one fixed summary before translating
 - b. They could not run on GPUs
 - c. They needed labelled data
 - d. They overfitted the training set
4. In a generative adversarial network, when is the generator good enough?
 - a. When it copies the training examples exactly
 - b. When the discriminator spots every fake
 - c. When it writes down a probability formula
 - d. When the discriminator can no longer tell its fakes from real examples

Answers

1. **b)** A GPU does thousands of simple calculations side by side, which happens to be exactly what training a neural network needs.
2. **c)** Models must use memory sparingly, but can afford far more calculation on tuning their parameters, which shifted the sweet spot to deep

networks.

3. a) Older models squeezed the whole sentence into one fixed-size summary; attention keeps a learnable pointer and looks back at the relevant part as it goes.

4. d) The generator is tuned against a discriminator that tries to tell fake from real, until the discriminator can no longer tell the difference.

Worked examples

The aggregate minibatch Training is spread over 1,024 GPUs, each handling a minibatch of 32 images. How many images does one training step process?

Each GPU processes 32 images per step.

All 1,024 GPUs work at the same time.

$1,024 \times 32 = 32,768$ images per step, about thirty-two thousand.

Answer: 32,768 images, about 32,000

Practice

1. Using the table, by what factor did memory grow from one kilobyte in 1970 to a hundred gigabytes in 2020, and by what factor did data grow from a hundred examples to a trillion?

Hint: A gigabyte is a million kilobytes; count powers of ten.

Answer: Memory: 100 GB is 100,000,000 KB, a factor of 10^8 . Data: from 10^2 to 10^{12} , a factor of 10^{10} . Data grew a hundred times more than memory.

2. If 256 GPUs each process a minibatch of 64 images, how many images does one step handle?

Hint: Multiply the number of GPUs by the minibatch size.

Answer: $256 \times 64 = 16,384$ images per step.

Hidden successes, visible ones

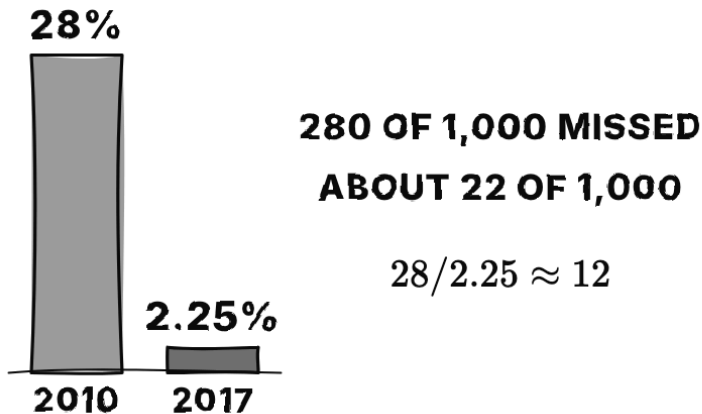
Write a letter, put a handwritten address on the envelope, and post it. Somewhere in a sorting centre, a machine reads that address and sends the letter on its way, and it has been doing so since the 1990s. You never see it, but it is machine learning, and the book says this is typical: the technology has been working quietly for decades.

First, though, the authors note how much easier the work itself has become. System researchers now build the tools, and statistical modellers build the networks on top of them. In 2014, training a simple model called logistic regression was a serious homework problem for new doctoral students at Carnegie Mellon University. Today it takes fewer than ten lines of code, within reach of any programmer.

Now the hidden successes. Reading handwritten addresses on mail is where the famous MNIST collection of handwritten digits came from. The same kind of reading handles cheques paid into banks, and similar models score whether loan applicants are likely to repay. Card payments are checked for fraud automatically, and that check is the backbone of payment systems such as PayPal, Stripe, AliPay, WeChat, Apple, Visa and MasterCard. Chess programs have been strong for decades. And machine learning drives search, recommendations, personalization and ranking across the Internet. In short, the book says, machine learning is everywhere, though mostly out of sight. Each time your card is declined abroad until you confirm a text message, a fraud model has looked at that payment and found it unusual.

Only recently has artificial intelligence stepped into the spotlight, mostly by solving problems that had seemed out of reach and that people meet directly; many of these advances are credited to deep learning. The first and most visible is the voice assistant. Siri, Alexa and Google's assistant answer spoken requests quite reliably, from switching on a light to booking a haircut or handling a support call. At their heart is speech recognition, which has improved until, for some uses, it matches human accuracy. When you dictate a message in a noisy street and it comes out right, you are using that improvement.

IMAGENET TOP-FIVE ERROR



From 28% to 2.25%

Recognising objects in pictures has come just as far. The book gives numbers from the ImageNet benchmark, where a model gets five guesses for each picture and counts as wrong only if none of the five is

right. In 2010, the best team, from NEC Labs and the University of Illinois, had a top-five error of 28%. By 2017 it was down to 2.25%. Out of a thousand pictures, that is about $0.28 \times 1,000 = 280$ missed in 2010, and only about 22 in 2017. The error shrank by a factor of $28 / 2.25 \approx 12$ in seven years. Similar leaps came in identifying birds by their song and in diagnosing skin cancer.

Games have long served as a yardstick of human ability, and the book traces a ladder of them. It begins with TD-Gammon, a backgammon program that learned by a form of reinforcement learning. Chess is far more complex than backgammon, and IBM's Deep Blue beat the world champion Garry Kasparov with massive parallel computing, special hardware and a clever search through possible moves. Go is harder still, because the number of possible positions is enormous; AlphaGo reached human level in 2015 by combining deep learning with a method of sampling likely continuations, called Monte Carlo tree search. Poker added another difficulty: the game is large and only partly visible, since you cannot see the other players' cards. A program called Libratus nonetheless beat human players, using carefully structured strategies.

HARDER



POKER



HIDDEN

GO

CHES

BACKGAMMON

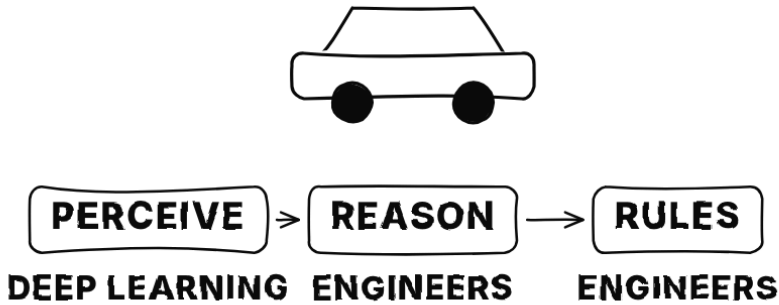
A ladder of games

Try it yourself

Think about how you play noughts and crosses against a child. You can see the whole board, so you can plan every move: that is like chess or Go. Now think of a card game where the other player's hand is hidden; you must guess from how they bet or play. That missing information is what made poker its own challenge for computers.

Self-driving cars are another sign of progress. Full autonomy is not here yet, but companies such as Tesla, NVIDIA and Waymo sell cars that can drive themselves in part. The hard part is that good driving

needs perception, reasoning, and following rules, all in one system. For now, the book says, deep learning mostly handles the seeing; the rest is carefully tuned by engineers.



AI ALREADY DECIDES: CREDIT, BAIL

Driving needs more than vision

And this is only a small sample. Robotics, logistics, the study of living cells, particle physics and astronomy all owe some of their recent advances to machine learning, which is becoming an everyday tool of engineers and scientists.

With success come worries. Popular articles often ask whether an "AI apocalypse" is coming, or a "singularity", a point where machines would become conscious and start making decisions about our lives without their programmers. The book's reply begins with a sober

observation: AI already affects people's lives directly. Whether you get credit is often decided automatically; autopilots do most of the flying; and decisions about granting bail to an accused person use statistical data. And, on a lighter note, we can ask Alexa to switch on the coffee machine.

Remember

Tools have made machine learning far easier: a model that was doctoral homework in 2014 now takes under ten lines of code. For decades it has worked out of sight, reading addresses and cheques, scoring credit, catching fraud and ranking search results. Deep learning brought visible successes: voice assistants, speech recognition at near-human level for some uses, and image recognition whose ImageNet error fell from 28% to 2.25%. Games from backgammon to chess, Go and poker fell one by one, and self-driving cars use deep learning mainly for seeing. AI already shapes real decisions about people, such as credit and bail.

Questions

1. Why does the lecture say machine learning is everywhere though mostly out of sight?
 - a. Because it only runs in laboratories
 - b. Because for decades it has quietly read mail, checked payments and ranked search results

- c. Because it became possible only in 2017
 - d. Because people refuse to use it
2. In ImageNet's top-five error, when does a model's answer count as wrong?
- a. When its first guess is wrong
 - b. When any of its five guesses is wrong
 - c. When none of its five guesses is right
 - d. When it takes more than five seconds
3. What made poker a different challenge from chess or Go?
- a. The game is only partly visible, since the other players' cards are hidden
 - b. It has fewer possible positions than chess
 - c. It cannot be played by computers
 - d. It has no rules
4. What part of self-driving does deep learning mainly handle today, according to the lecture?
- a. Following traffic rules
 - b. Planning the route
 - c. Reasoning about other drivers
 - d. Seeing, that is, perception

Answers

1. **b)** Mail sorting, cheque reading, credit scoring, fraud checks, search and recommendations all use machine learning that nobody sees.
2. **c)** A model gets five guesses for each picture and counts as wrong only if none of the five is right.

- 3. a)** Poker is large and only partly visible, because you cannot see the other players' cards.
- 4. d)** Good driving needs perception, reasoning and rules; deep learning mostly handles the seeing, and the rest is tuned by engineers.

Worked examples

How far image recognition came The ImageNet top-five error was 28% in 2010 and 2.25% in 2017. How many of 1,000 pictures were missed in each year, and by what factor did the error shrink?

2010: $0.28 \times 1,000 = 280$ pictures missed.

2017: $0.0225 \times 1,000 = 22.5$, about 22 pictures missed.

Factor: $28 \div 2.25 \approx 12.4$, so the error shrank about twelvefold in seven years.

Answer: About 280 missed in 2010, about 22 in 2017; roughly twelve times smaller

Practice

1. A model's top-five error on 2,000 pictures is 5%. How many pictures does it miss?

Hint: Take 5 percent of 2,000.

Answer: $0.05 \times 2,000 = 100$ pictures, those for which none of its five guesses is right.

2. Name three ways in which, according to the lecture, AI already affects people's lives directly.

Hint: Think of money, flying and the courts.

Answer: Credit is often decided automatically, autopilots do most of the flying, and decisions about granting bail use statistical data.

What AI is not, and what makes learning deep

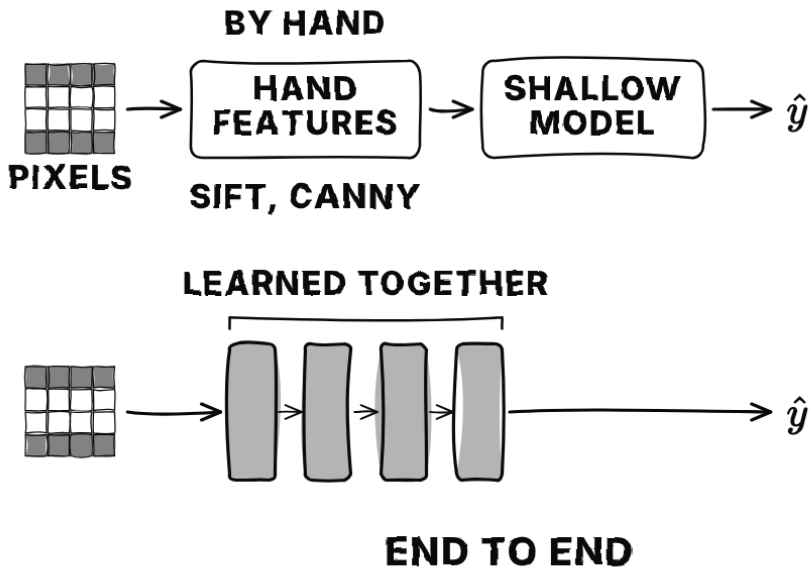
Ask a chatbot whether it is planning to take over the world, and it may give you a thoughtful, even eerie answer. It sounds like someone with a mind of their own. The book's verdict on that impression is calm: we are far from any machine that could knowingly manipulate the people who built it.

The authors give two reasons. First, AI systems are engineered, trained and deployed for particular goals. Their behaviour may give the illusion of general intelligence, but underneath it is a mix of rules, rules of thumb and statistical models. Second, there are at present simply no tools for artificial general intelligence, systems that could improve themselves, reason about themselves, and redesign their own structure while trying to solve any task at all. The chatbot's eerie answer is text predicted from other text, as we heard earlier, not a plan.

The real worry, the book says, is how AI is used in everyday life. Many routine jobs now done by people can and will be automated. Farm robots will probably make organic farming cheaper, but they will also take over the harvest. This stage of the industrial revolution may hit large parts of society hard, because in many countries simple manual work provides a great many jobs. And statistical models applied carelessly can treat people differently by race, sex or age, which raises fair questions about justice when such models make important decisions automatically. These algorithms must be used with care,

and the authors judge that to be far more urgent than any evil superintelligence destroying humanity.

Now the book turns to what deep learning actually is. Deep learning is the part of machine learning built on neural networks with many layers. It is deep in exactly that sense: its models learn many layers of transformations, one after another. One could argue that all machine learning has layers, starting with steps that prepare the features. What sets deep learning apart is that every layer is learned, and all of them are learned together, from the data.



Hand-made features versus end to end

The problems we have met, sound waves, raw pixels, sentences of any length to be translated, are just the ones where deep learning shines

and older methods struggled. Many-layered models can work directly on raw, low-level data from the senses in a way earlier tools could not. The common thread, the book says, is end-to-end training. Instead of building a system from parts, each tuned on its own, you build the whole system and tune all of it together.

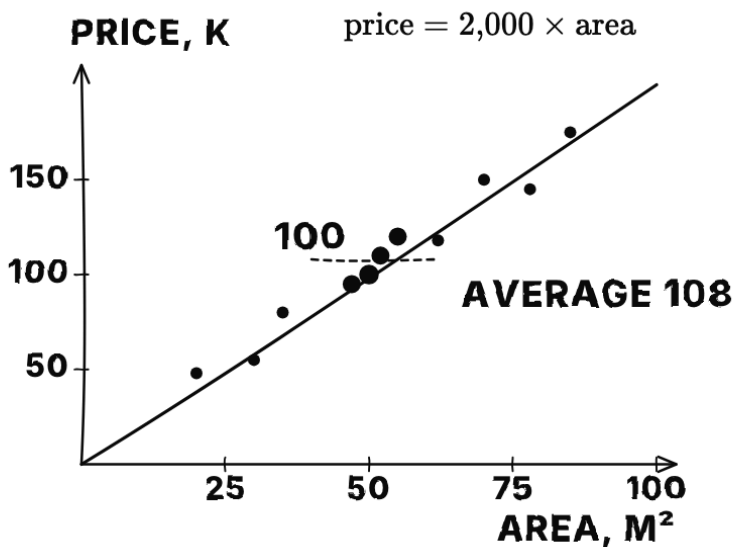
Computer vision is the clearest case. For years, scientists kept two jobs apart: feature engineering, deciding by hand how to turn a picture into numbers, and building the model that learned from those numbers. Hand-made tools such as the Canny edge detector and Lowe's SIFT method ruled for more than a decade. The clever part of the work was inventing such transformations so that a simple, shallow model could cope. But human ingenuity can only try so much, compared with an algorithm that tests millions of choices consistently and automatically. When deep learning arrived, the hand-made feature extractors gave way to filters tuned automatically, and accuracy went up. The face filters in a video-call app find your eyes and mouth with features nobody drew by hand; the network learned them from examples.

So deep learning replaces two things at once: the shallow model at the end of the old pipeline, and the laborious feature engineering at its start. Because it does away with so much preparation specific to each field, it has also broken down walls. Vision, speech recognition, language processing and medical informatics used to have their own separate toolkits; deep learning gives them one shared set of tools.

Try it yourself

Try describing, in written rules, how to find the edge of a coffee mug in a photo: "where dark meets light", perhaps. Then think of a white mug on a white table, or a mug in shadow, and watch your rule fail. An engineer of the 2000s spent years refining such rules; a deep network learns its own from thousands of mug photos. That is the trade the book describes: hand-made features out, learned layers in.

The book names a second shift: from parametric to nonparametric models. A parametric model fixes the shape of the answer in advance, with a few numbers to fill in, like a straight line with its slope and its starting value. When data is scarce, such simplifying assumptions are the only way to get a useful model. When data is plentiful, the assumptions can be dropped and the model can follow the data itself. Here is a made-up example. To price a flat, a straight-line rule might say $\text{price} = 2,000 \times \text{area}$, so a flat of 50 square metres costs 100,000. A nonparametric approach would instead find the three sold flats most like this one and average their prices, say 95, 110 and 120 thousand, giving $(95 + 110 + 120) / 3 \approx 108$ thousand. With little data, the line is safer; with a city's worth of sales, the neighbours capture things the line never could, such as a quiet street or a view.



A line versus the neighbours

The authors compare this to physics in the middle of the last century. Once computers arrived, physicists no longer had to solve simplified formulas for how electrons behave by hand; they could simulate the underlying equations numerically. The models became far more accurate, but often harder to interpret. Deep learning pays the same price: the flexible models fit better, and are harder to explain.

Remember

We are far from sentient AI: today's systems are built for specific goals, and no tools exist yet for machines that improve and redesign themselves. The pressing concerns are automation of routine work and careless models that treat people unfairly. Deep learning means many layers of transformations, all learned together from data, and trained end to end. It replaces both hand-made feature engineering and the shallow model, and gives vision, speech and language one shared toolkit. With abundant data, flexible nonparametric models replace simplifying assumptions, gaining accuracy at some cost to interpretation.

Questions

1. Why, according to the book, are we far from an AI that could knowingly manipulate its makers?
 - a. Because AI systems are switched off at night
 - b. Because AI is built for particular goals and no tools exist yet for self-improving general intelligence
 - c. Because chatbots cannot write text
 - d. Because laws forbid it
2. What does the book consider more urgent than an evil superintelligence?
 - a. Computers running out of memory

- b. Chatbots giving eerie answers
 - c. Automation of routine work and careless models treating people unfairly
 - d. Games becoming too easy for computers
3. What sets deep learning apart, if all machine learning arguably has layers?
- a. Every layer is learned, and all are learned together from the data
 - b. It has exactly two layers
 - c. Its first layer is always designed by hand
 - d. It uses no data
4. What price do flexible nonparametric models pay, like physics simulations?
- a. They need less data
 - b. They are less accurate
 - c. They cannot be computed
 - d. They are often harder to interpret

Answers

- 1. b)** AI systems are engineered for particular goals, a mix of rules and statistical models, and there are no tools yet for systems that improve and redesign themselves.
- 2. c)** The real worry is how AI is used: routine jobs being automated, and statistical models applied carelessly treating people differently by race, sex or age.
- 3. a)** What sets deep learning apart is that every layer is learned, and all of them are learned together, from the data.

4. **d)** Simulating the equations made physics models far more accurate but often harder to interpret; deep learning's flexible models pay the same price.

Worked examples

A line against the neighbours A straight-line rule prices a flat at $\text{price} = 2,000 \times \text{area}$. A 50-square-metre flat's three most similar sold neighbours went for 95, 110 and 120 thousand. Compare the line's price with the neighbours' average.

Line: $2,000 \times 50 = 100,000$.

Neighbours: $(95 + 110 + 120) \div 3 = 325 \div 3 \approx 108.3$ thousand.

The line fixes the shape in advance; the neighbours' average follows the data itself.

With plenty of sales data the neighbours can capture things the line misses, such as a quiet street.

Answer: Line: 100,000; neighbours: about 108,000

Practice

1. With the same line rule, $\text{price} = 2,000 \times \text{area}$, what is the price of a 65-square-metre flat? If its three closest neighbours sold for 120, 126 and 138 thousand, what does the nonparametric approach give?

Hint: Multiply for the line; average for the neighbours.

Answer: Line: $2,000 \times 65 = 130,000$. Neighbours: $(120 + 126 + 138) \div 3 = 384 \div 3 = 128$ thousand.

2. Explain in your own words what end-to-end training replaced in computer vision.

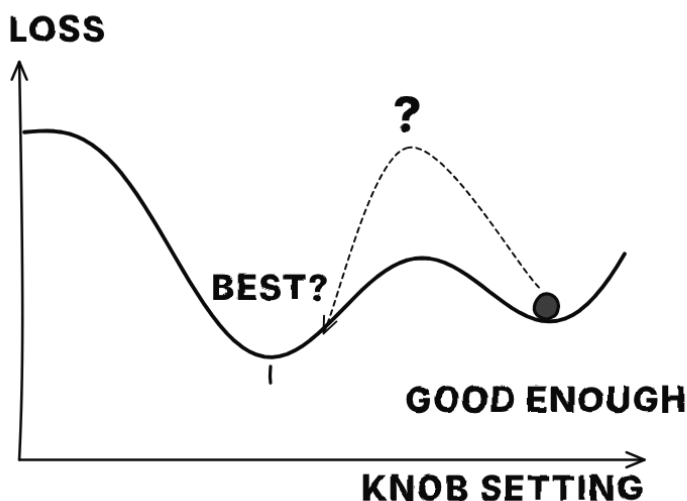
Hint: Think of the two separate jobs scientists used to do.

Answer: Scientists used to design features by hand, with tools such as edge detectors and SIFT, and then train a shallow model on them

separately. End-to-end training builds the whole system and tunes all of it together, so the network learns its own features and its final model at once.

Try it, share it

Picture the walk down the hillside from earlier in the course, but now the field is full of hollows and ridges. You may end up in a small hollow rather than the deepest valley, and you cannot prove you found the best spot. The book says deep learning has made peace with that.



Hollows and ridges

Earlier work wanted problems where the best answer can be found and proved. Deep learning accepts answers that are good rather than perfect, works with tangled, non-convex landscapes of the loss full of

hollows, and is willing to try things before anyone can prove they work. This new empiricism, together with a flood of talented newcomers, has produced practical methods quickly, though often by reworking or reinventing tools that existed for decades.

The community also prides itself on sharing across universities and companies: libraries, models and fully trained networks are released openly. In the same spirit, the authors make the notebooks of their book free to use and pass on, to lower the barriers for anyone who wants to learn. Many of the models behind the apps on your phone started from such openly shared networks.

The chapter's summary pulls the threads together. Machine learning studies how computers can use experience, usually data, to get better at a task; it draws on statistics, data mining and optimization, and is often the way AI is built. Representation learning, a branch of it, looks for good ways to represent data automatically. Deep learning learns representations in many layers, and so replaces both the shallow model and the hand-made feature engineering. Its recent progress came from abundant data, from cheap sensors and the Internet, from far more computation, mostly through GPUs, and from efficient frameworks that make it easy to build and tune whole systems.

That closes the introduction: from rules that never learn, through the many kinds of learning problems, to the ingredients that made deep learning work.

Remember

Deep learning accepts good, imperfect solutions, works with tangled loss landscapes, and tries ideas before proving them. Its community shares tools, models and trained networks openly, and this book is free in the same spirit. Machine learning uses experience to improve; deep learning learns layered representations and replaces hand-made features. Data, GPUs and good frameworks are the three forces behind its recent progress.

Questions

1. How does deep learning's attitude to optimization differ from earlier work?
 - a. It only accepts solutions proved to be the best
 - b. It accepts good but imperfect solutions and tries things before proving them
 - c. It avoids non-convex problems
 - d. It never uses gradient descent
2. On a loss landscape full of hollows and ridges, what can happen when walking downhill?
 - a. You always reach the deepest valley
 - b. You can never move at all
 - c. The loss stops depending on the parameters

- d. You may end up in a small hollow rather than the deepest valley
- 3. Which three forces does the summary name behind deep learning's recent progress?
 - a. Abundant data, more computation through GPUs, and efficient frameworks
 - b. Hand-made features, small datasets and proofs
 - c. Rules, databases and web browsers
 - d. Chess, Go and poker

Answers

- 1. **b)** Earlier work wanted provably best answers; deep learning accepts good rather than perfect answers, works with tangled landscapes and tries ideas before proving them.
- 2. **d)** In a field full of hollows you may stop in a small hollow and cannot prove you found the best spot.
- 3. **a)** Recent progress came from abundant data, from far more computation mostly through GPUs, and from efficient frameworks that make whole systems easy to build and tune.

Practice

- 1. Explain in your own words what representation learning is and how deep learning carries it further.

Hint: Think of what feature engineering did by hand.

Answer: Representation learning looks for good ways to represent data automatically, instead of people designing the features by hand. Deep learning learns such representations in many layers, each building on the

previous one, which lets it replace both hand-made feature engineering and the shallow model at the end.

2. Why might openly sharing trained networks speed up progress, according to the spirit the lecture describes?

Hint: Think of who can build on a shared model.

Answer: When libraries, models and trained networks are released openly, anyone at a university or company can start from them instead of from scratch, which lowers the barriers to entry and lets ideas spread and be improved quickly.