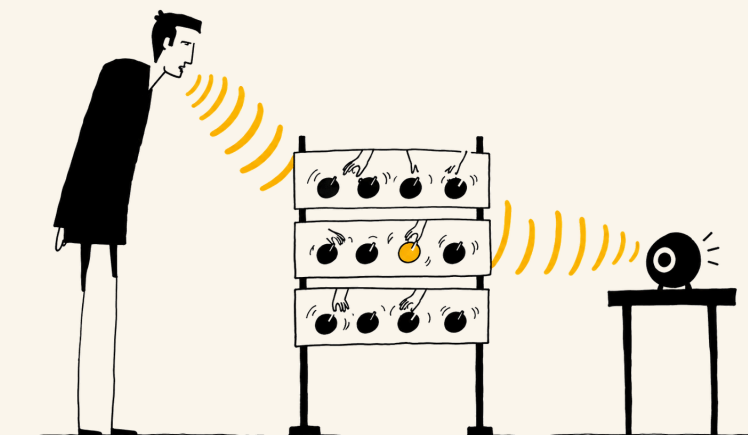


# Dive into Deep Learning



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 instead of rules

What the machine learns from

Scoring the machine

Feeling for the way down

How much, how many

Which one, and how sure

Many labels, and the best first

Your feed, and things that unfold in time

Turning things round, and learning without answers

When the machine acts

Blind spots, gambles and old roots

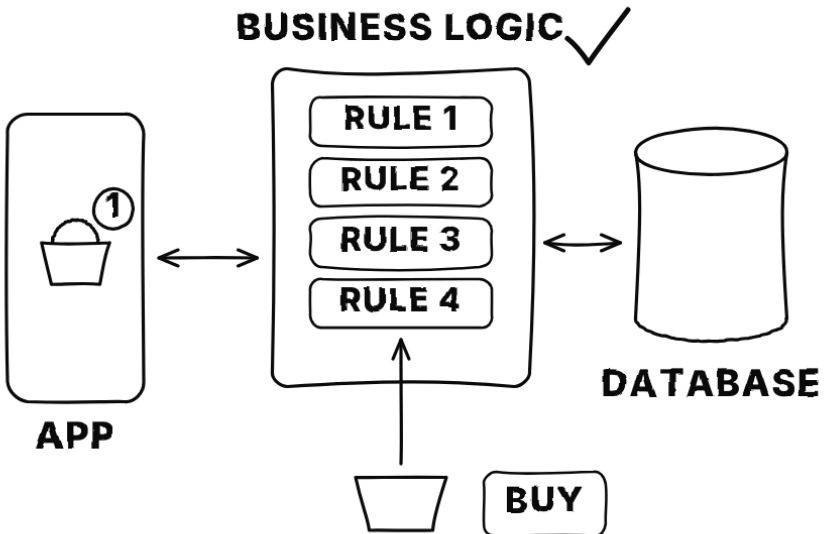
From a dead end to an explosion

Quiet successes and real worries

What makes it deep

# Rules that never learn

Open any shopping app on your phone, put a pair of socks in the basket, and watch the little number on the basket icon jump from zero to one. You can do it a hundred times, and it will jump every time, in exactly the same way. Somewhere a programmer wrote a rule for that moment: when this person taps "add", write a new line in a table that pairs this person with these socks. Nothing about that rule changes because you used it. The app does not get better at adding socks; it simply does what it was told.

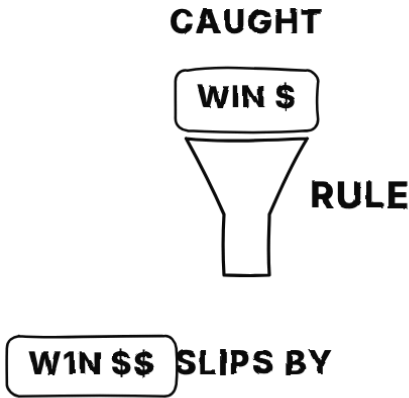


A shop built from rules

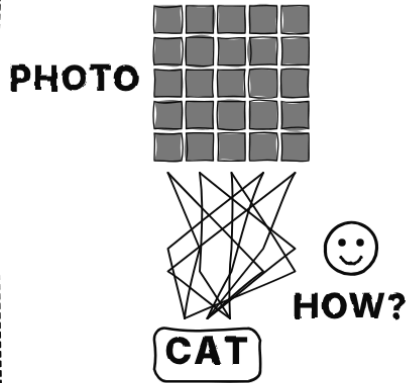
For most of the history of computing, nearly every program you touched in a day was built like this, as a long list of rules written by people. The book pictures a team in front of a whiteboard, planning a shop on the internet. They decide that customers will see the shop in a browser or a phone app, that a database will remember every customer and every past purchase, and that in the middle there will be the part the authors call the business logic, the brains of the shop. Those brains are a set of rules that tell the program what to do in every situation the team can think of. Then the team goes hunting for awkward cases, such as a customer who presses "buy" with an empty basket, and adds a rule for each one. It rarely works perfectly the first time, but after some testing it does, and the shop can open before a single real customer has walked in. The authors call this a remarkable feat of human thinking, and they add a sharp piece of advice: if you can write rules that are right one hundred percent of the time, you do not need machine learning at all.

Now imagine the same clever team at the same whiteboard, but with different tasks. Predict tomorrow's weather from maps, satellite pictures and the last few weeks of measurements. Read a question typed in ordinary words and answer it correctly. Look at a photograph, find every person in it and draw an outline around each one. Show a shopper things they will probably love but would never have stumbled on by browsing. The last task is the one your recommendation feed tries to do every time you open it. For tasks like these, even the best programmers in the world would struggle to write the rules from scratch.

## MOVING TARGET



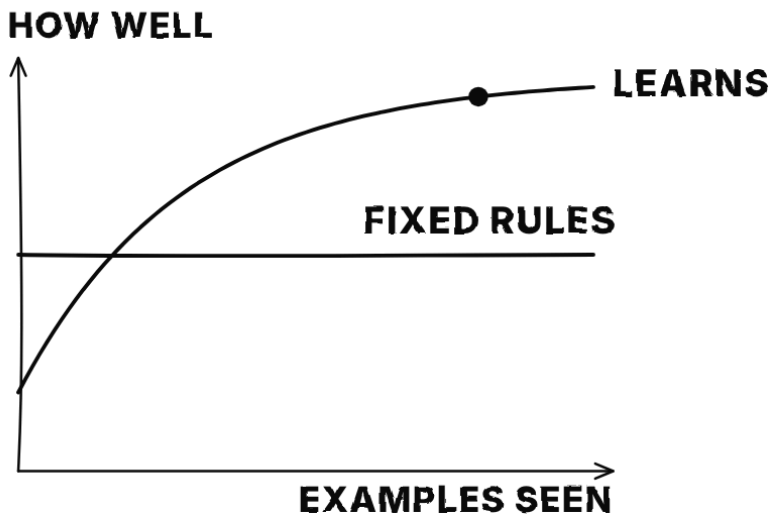
## TOO TANGLED



Two reasons rules fail

The book gives two reasons. The first is that the right answer may keep moving. What shoppers want this spring is not what they wanted last spring, so any fixed set of rules goes stale, and a good solution has to adjust as the world changes. A spam filter knows this well: the moment it learns to catch one kind of junk mail, the senders change their wording, and yesterday's rule stops working. The second reason is that the link between what goes in and what should come out can be far too tangled to write down. A photograph is just a grid of coloured dots, and the word "cat" is an idea; between the two there may be millions of small calculations following principles nobody has ever spelled out. You yourself recognise a friend's face in a crowd without effort, yet you could not write down the steps your brain took to do it. The skill is there, but it is hidden from your own conscious mind.

This is where a different idea comes in. Machine learning is the study of methods that learn from experience. Experience here usually means data, meaning many recorded examples, or the results of acting in some setting and seeing what happens. The more experience such a method gathers, the better it performs. Compare that with the sock button. The online shop follows the same business logic no matter how many millions of customers pass through it, and it changes only when its programmers learn something and decide to rewrite it. A learning system, by contrast, improves from the very examples it sees. When a voice assistant gets better at understanding your accent, no programmer wrote a special rule for your accent; it improved because it was trained on large amounts of recorded speech.



Fixed rules versus learning

The book sets out to teach the foundations of machine learning, and above all one powerful family of its methods, called deep learning. These are the methods behind much of today's progress in teaching computers to see, to work with human language, to help in medicine and to read genetic information. The chatbot you type questions into is one product of deep learning: nobody wrote its answers as rules; it learned the patterns of language from an enormous amount of text. So the dividing line is simple to state. On one side are programs whose behaviour is fixed by people in advance; on the other are programs whose behaviour is shaped by examples.

### **Remember**

Most programs are lists of rules written by people, and they behave the same way however much they are used. Rules work well when people can foresee every case and get them right every time, and then machine learning is not needed. Some tasks resist rules, either because the right answer keeps changing or because the path from input to answer is too tangled to write down, even when our own brains manage it without effort. Machine learning is the study of methods that improve with experience, usually from data. Deep learning is the family of such methods this book is about.

---

## **Questions**

1. Why does the shop's add-to-basket button not get any better, however many customers use it?

- a. Because it follows rules written in advance, which change only when programmers rewrite them
  - b. Because it has not yet collected enough customers to learn from
  - c. Because the database forgets every purchase after it is made
  - d. Because shopping is a task that changes too quickly to learn
2. A team can write rules for a task that are right every single time. What does the book advise?
- a. Train a learning system anyway, because it will do even better
  - b. Mix the rules with a learning system to be safe
  - c. Use the rules; machine learning is not needed for that task
  - d. Collect data first, then decide which rules to keep
3. Recognising a friend's face is easy for you. Why is it still hard to write as rules?
- a. Because faces change too often for any program
  - b. Because the steps your brain takes are hidden from your conscious mind
  - c. Because computers cannot store photographs
  - d. Because there are too few faces to learn from
4. Which of these is the other reason the lecture gives for rules failing, besides tangled links between input and answer?
- a. Rules take too long for a computer to run
  - b. Programmers rarely test their rules
  - c. Databases cannot hold enough rules
  - d. The right answer keeps changing, so fixed rules go stale

## Answers

1. **a)** The shop follows the same business logic no matter how many customers pass through; it changes only when its programmers decide to rewrite it.
2. **c)** If you can write rules that are right one hundred percent of the time, you do not need machine learning at all.
3. **b)** You recognise the face without effort, yet you could not write down the steps your brain took; the skill is hidden from your conscious mind.
4. **d)** Sometimes the right answer keeps moving, so any fixed set of rules goes stale and a good solution has to adjust as the world changes.

## Practice

1. Take the four whiteboard tasks: tomorrow's weather, answering a typed question, outlining every person in a photo, and suggesting products a shopper would not find by browsing. For each, say which of the two reasons makes hand-written rules hard: the answer keeps changing, or the link from input to answer is too tangled to write down.

**Hint:** Ask of each task: does the right answer drift over time, or would you be unable to list the steps even though the answer is fixed?

**Answer:** Weather and shopping tastes change over time, so fixed rules go stale; weather also depends on far too many tangled measurements. Answering free questions and outlining people in a photo are mostly the tangled kind: we do them easily ourselves but cannot spell out the steps. Several tasks have both difficulties at once.

2. Explain in your own words why the empty-basket case belongs to ordinary programming and not to machine learning.

**Hint:** Can the team foresee this case and write the correct response once and for all?

**Answer:** Pressing buy with an empty basket is a situation the team can foresee before any customer arrives, and the right response, such as showing a message, can be written down and is right every time. Tasks like that are handled with rules; machine learning is for tasks where no one can write such rules.

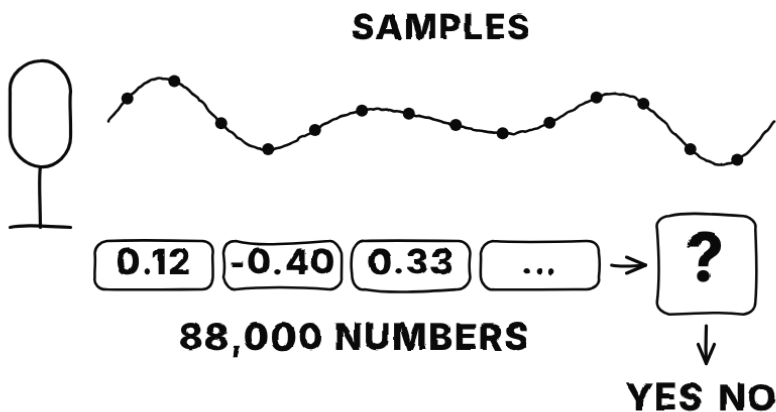
**3.** A spam filter is built in two ways: one with a fixed list of banned phrases written by programmers, the other trained on many examples of mail marked spam or not spam. What happens to each as spammers change their wording over the months?

**Hint:** Which one changes only when a person rewrites it, and which one can improve from new examples?

**Answer:** The fixed list stays the same until someone rewrites it, so new wording slips through and it grows stale. The trained filter can improve as it is given new marked examples, because a learning method gets better with experience.

## Knobs instead of rules

Say "Hey Siri" or "OK Google" across the room and watch your phone light up. It woke because it picked one particular phrase out of all the sound around it: the television, the kettle, other people talking. Now try to imagine writing that program yourself, alone in a room with a computer and nothing else. What would the first rule be?



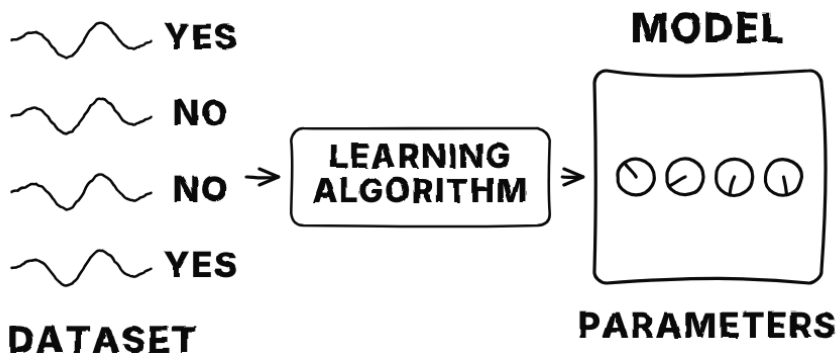
Sound in, yes or no out

The authors open the section with a small scene, which they admit they made up to make the point. On their way to buy coffee, one of them says "Hey Siri" to a phone, and the phone wakes. Another asks

for directions to a coffee shop. The phone writes out what he said, understands that he wants directions, opens the maps, finds several routes and shows a predicted travel time next to each. In those few seconds, several separate learned programs did their work: one woke the phone, one turned speech into text, one worked out what was wanted, one estimated the travel times. The same thing happens when you ask a voice assistant to set a timer: listening for the wake phrase, writing down your words and working out what you mean are each done by a learned program.

Take only the first of these jobs, the wake word. A phone's microphone measures the sound about forty-four thousand times every second, and each measurement is a single number, how far the sound wave is pushed up or down at that instant. So a two-second snippet of sound is forty-four thousand times two, about eighty-eight thousand numbers in a row. The program must look at those eighty-eight thousand numbers and answer just yes or no: was the wake word in there? No one knows a rule that does this reliably. The book admits it plainly: the authors do not know how to write such a program from scratch either, and that is exactly why machine learning is used.

Here is the trick. We cannot tell the computer how to recognise "Alexa", but we can recognise it ourselves without effort. So people listen to a huge number of short recordings and mark each one yes or no. That pile of recordings with their marks is called a dataset, and each mark is a label. Then, instead of writing a recogniser, we write a flexible program whose behaviour depends on many adjustable numbers, called parameters. The dataset is used to find the values of the parameters that make the program do best by some chosen measure of performance.



Labels, knobs and a learner

The book asks you to think of the parameters as knobs. Turn the knobs one way and the program behaves one way; turn them another way and it behaves differently. Once the knobs are set and left alone, the program is called a model. All the programs you could get by turning the knobs of one design form a family of models. And the program that uses the data to choose the knob settings is called the learning algorithm, a program whose output is another program.

Before any of this can run, the job has to be pinned down: what comes in, what goes out, and which family of models to use. Here the input is a snippet of audio and the output is yes or no. If the family is well chosen, one setting of its knobs makes it say yes whenever it hears "Alexa". And since the choice of wake word is arbitrary, a rich enough

family should also have another setting that says yes only for "Apricot". The two tasks are so alike that the same family should cope with both. But turning photographs into captions, or English sentences into Chinese ones, involves quite different inputs and outputs, and may need a different family altogether.

### **Try it yourself**

Ask someone to whisper a word in a noisy room, then mumble it, then shout it from the next room. You will recognise it every time, yet if you try to describe in words what makes the sound "Alexa" and not "Alexis", you will find there is nothing you can write down. That gap between doing something and being able to state the rule is the reason learning from examples exists.

With the knobs set at random, the model recognises nothing at all. Learning means searching for the setting of the knobs that makes the model behave the way we want, and doing this with data is called training. The book draws training as a loop. Start with a model whose knobs are random and which does nothing useful. Take a handful of examples, recordings with their yes or no labels. Nudge the knobs so the model does a little better on those examples. Then take more examples and nudge again, round and round, until the model is very good.

So instead of writing a wake word recogniser, we write a program that learns to become one when shown enough labelled examples. The authors call this programming with data. In the same way you could "program" a detector that tells cats from dogs simply by showing it many pictures of each. In the end it gives out a large positive number

when it sees a cat, a large negative number when it sees a dog, and a number near zero when it is unsure. Your spam filter works the same way: nobody listed every spam phrase; it was shown many messages marked spam or not spam, and its knobs settled on whatever separates them. The book adds that deep learning, which comes later, is only one of many popular ways of doing this.

### **Remember**

Even a few seconds with a phone can involve several learned programs, from waking on a phrase to estimating travel time. A wake word arrives as tens of thousands of numbers per second, and no one knows a rule that turns them reliably into yes or no. We can recognise the word ourselves, so we label many examples and let a flexible program with adjustable knobs, its parameters, learn from them. A model is the program with its knobs set; a learning algorithm chooses the settings; training repeats small adjustments on examples until the model does well. This is programming with data, and deep learning is one way of doing it.

---

## **Questions**

1. Why do people label recordings yes or no before training a wake word detector?
  - a. Because the computer cannot store unlabelled sound

- b. Because we can recognise the word ourselves even though we cannot write the rule
  - c. Because labels make each recording shorter
  - d. Because the microphone records only labelled sound
2. What is the difference between a model and a family of models?
- a. A family is a model trained on more data
  - b. A model is the program with its knobs set; a family is all the programs one design can become by turning its knobs
  - c. A model answers yes or no; a family answers with numbers
  - d. A family is a group of datasets used together
3. Why should one family of models cope with both "Alexa" and "Apricot" but perhaps not with captioning photos?
- a. Because captions need more knobs than any family can have
  - b. Because the two wake words are similar tasks with the same kind of input and output, while captioning has quite different inputs and outputs
  - c. Because "Apricot" is easier to hear than "Alexa"
  - d. Because photos cannot be turned into numbers
4. In the training loop, what happens to the knobs at each round?
- a. They are all set to new random values
  - b. They are nudged so the model does a little better on the examples just shown
  - c. They are fixed by a programmer who listens to the examples
  - d. They are left alone while more data is collected

## Answers

1. **b)** We cannot tell the computer how to recognise the word, but we can recognise it ourselves, so we mark many recordings and let the program learn from them.
2. **b)** Once the knobs are set the program is called a model; all the programs you could get by turning the knobs of one design form a family.
3. **b)** The two wake word jobs are alike, so another setting of the same knobs suffices, while photos to captions involves quite different inputs and outputs.
4. **b)** Take a handful of labelled examples and nudge the knobs so the model does a little better on them, then repeat.

## Worked examples

**How many numbers a snippet holds** A microphone measures the sound about 44,000 times every second, and each measurement is one number. How many numbers must a wake word detector look at for a three-second snippet?

Each second gives about 44,000 measurements, each one number.

Three seconds give three times as many.

44,000 times 3 is 132,000.

All of those numbers must be turned into a single yes or no, which is why no simple rule works.

**Answer:** about 132,000 numbers

## Practice

1. Put the four steps of the training loop in order and say why the first model is useless: nudge the knobs; start with random knobs; repeat until good; take some labelled examples.

**Hint:** What can a model do before it has seen any data?

**Answer:** Start with random knobs, take some labelled examples, nudge the knobs so the model does better on them, and repeat until it is good. The first model is useless because its knob settings are random and it has learned nothing yet.

**2.** A cat-and-dog detector gives out a number for each picture. It gives 9 for one picture, minus 8 for another and 0.3 for a third. What does it think each picture shows?

**Hint:** Large positive means cat, large negative means dog, near zero means unsure.

**Answer:** The first is a cat (large positive), the second a dog (large negative), and for the third it is unsure, since the number is close to zero.

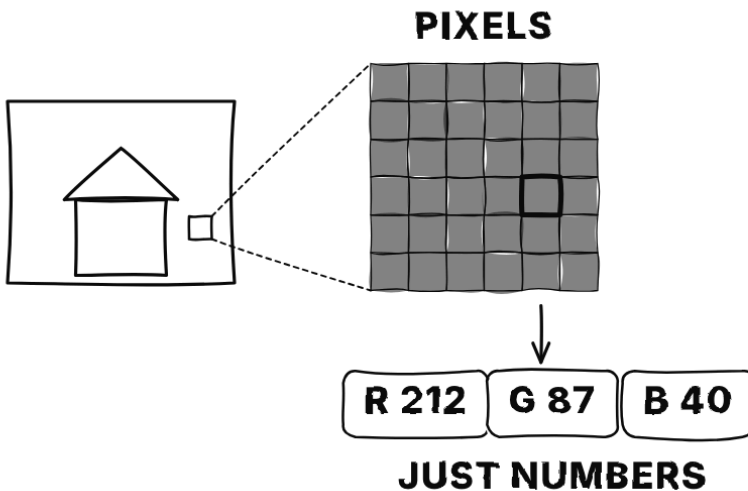
**3.** Explain in your own words why the authors call training "programming with data".

**Hint:** Who decides the program's behaviour, a person writing rules or something else?

**Answer:** The behaviour of the program is not written by a person rule by rule; it is set by showing the program labelled examples and letting the learning algorithm choose the knob settings. The data takes the place of the hand-written rules.

# What the machine learns from

Take a photo with your phone and zoom in as far as it will go. Somewhere along the way the picture falls apart into little coloured squares, the pixels. Each square is really three numbers: how bright its red light is, how bright its green, how bright its blue. To a learning program, that is all your photo ever is: a long list of numbers.

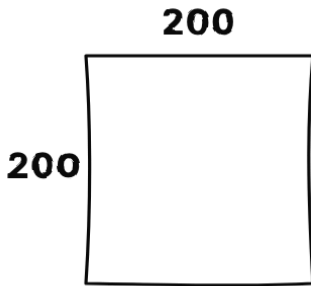


A photo is only numbers

The wake word detector from the last part learned from recordings marked yes or no. That kind of task has a name: supervised learning. We have many examples whose right answers are known, and we

want a program that predicts the right answer for new examples where it is not known. It is only one of many kinds of machine learning problem, but the book first names four parts that turn up in every kind. 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 adjusts the model's parameters, the knobs, to make that score as good as possible. This part is about the first of the four, the data.

Data is a collection of examples, and to be useful each example has to be turned into numbers. The things about an example that the model gets to see are called its features. In supervised learning there is also one special thing the model must predict and is not shown, called the label. In a photo sorter, the photo is the features and a number naming its category is the label. The book counts what a photo becomes. A colour picture two hundred pixels wide and two hundred pixels high has two hundred times two hundred pixels, which is forty thousand. Each pixel carries three numbers, for red, green and blue, so forty thousand times three gives one hundred and twenty thousand numbers for one small picture.

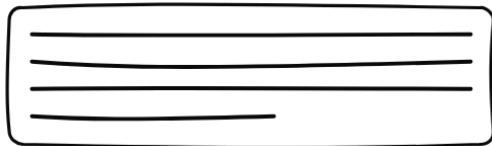


$$200 \times 200 = 40,000$$

$$40,000 \times 3 = 120,000$$

**IT STINKS!**

**SHORT**



**LONG**

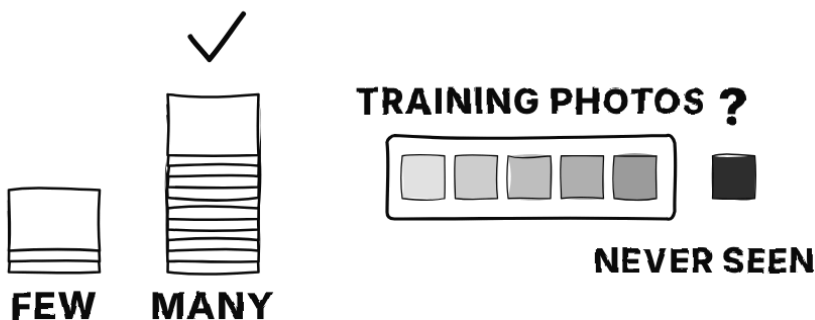
$$200 \times 200 \times 3 = 120,000$$

The book's second example comes from a hospital. The task is to predict whether a patient will survive the next thirty days. The features are things already in the patient's record: age, pulse and blood pressure, other illnesses, current medicines, recent operations. The label, for training, is simply whether each past patient did survive those thirty days, a yes or a no. The same pattern sits under your spam filter: the features describe each message, and the label is spam or not spam.

When every example is described by the same number of features, the examples are called fixed-length, and that number is the dimensionality of the data. For the small photos it is one hundred and twenty thousand; for the patient records it is however many items

each record holds. Fixed length makes life easier, but much data refuses to fit. Pictures from a microscope may all come out the same size, but pictures gathered from the internet come in every shape and size. One can crop them to a standard size, but whatever is cut off is lost. Text is even more stubborn. A review on a shopping site may be two words long, "it stinks!", or run on for pages. The book counts it as one great advantage of deep learning that its models cope comparatively well with examples of varying length. The chatbot you type to is a case in point: it takes your question whether it is three words or three paragraphs.

More data usually makes the job easier. With more examples one can train more powerful models and lean less on assumptions made in advance. The move from fairly small datasets to very large ones is, the book says, one of the main reasons for deep learning's success. Many of its most exciting models do not work at all without huge datasets, and some that do work on small ones are no better than older methods.



## **GARBAGE IN, GARBAGE OUT**

More data, the right data

But quantity is not enough; the data has to be the right data. If it is full of mistakes, or the features have nothing to do with what we want to predict, learning will fail. The book quotes the old saying "garbage in, garbage out". And in some uses the harm goes beyond a wrong answer: in predicting crime, screening job applications or deciding who gets a loan, bad data can hurt people. One common failure is that some group of people is simply missing from the examples. Picture a system for spotting skin cancer that was never shown dark skin during training; it has learned nothing about the people it will fail. Another failure is subtler: the data does contain everyone, but it carries the prejudices of the past. If a model is trained on a company's past hiring decisions and then used to sort new applications, it can

quietly copy and automate old injustice. The authors stress that this can happen without anyone meaning it, or even noticing.

### **Remember**

Supervised learning means predicting an unknown label from known features, after learning from examples whose labels are known. Every machine learning problem has four parts: data, a model, an objective function that scores the model, and an algorithm that adjusts its parameters. Data has to become numbers: a small colour photo of two hundred by two hundred pixels is already one hundred and twenty thousand of them. Fixed-length examples are easy, but text and internet images vary in size, and deep learning handles such varying-length data comparatively well. More data helps, but only the right data: missing groups or past prejudice in the examples become failures and injustice in the model, even without anyone intending it.

---

## **Questions**

1. In the hospital example, which is the label?
  - a. The patient's age and pulse
  - b. The list of current medicines
  - c. Whether the patient survived the next thirty days
  - d. The number of recent operations

2. Why is text harder to handle than photos from one microscope?
  - a. Because text cannot be turned into numbers
  - b. Because texts come in very different lengths, so they do not fit a fixed number of features
  - c. Because text has no labels
  - d. Because reviews are always too short to learn from
3. A skin cancer detector works well in tests but fails on patients with dark skin. What went wrong, according to the lecture?
  - a. The model had too many parameters
  - b. The objective function was the wrong score
  - c. Dark skin was missing from the training examples
  - d. The pictures were not cropped to a standard size
4. Why can a model trained on a company's past hiring decisions be unfair even if no one intended it?
  - a. Because the data carries past prejudice, and the model learns and repeats it
  - b. Because hiring data is always too small
  - c. Because job applications are varying-length text
  - d. Because the model was never given labels

## Answers

1. **c)** The features are what is already in the record; the label is whether each past patient survived the thirty days.
2. **b)** A review may be two words long or run on for pages, so text resists a fixed length even more than internet images.

3. c) One common failure is that some group of people is simply missing from the examples, as with a system never shown dark skin.
4. a) If the data carries the prejudices of the past, a model trained on it can quietly copy and automate old injustice.

## Worked examples

**Counting the numbers in a photo** A colour photo is 200 pixels wide and 200 pixels high, and each pixel is stored as three numbers, for red, green and blue. How many numbers describe the photo?

Count the pixels: 200 across times 200 down is 40,000 pixels.

Each pixel has three brightness numbers, one per colour.

So multiply the pixels by three: 40,000 times 3 is 120,000.

Every photo of this size has the same count, so its dimensionality is 120,000.

**Answer:** 120,000 numbers

## Practice

1. A smaller colour photo is 100 pixels wide and 100 pixels high. How many numbers describe it, and how does that compare with the 200 by 200 photo?

**Hint:** First count the pixels, then remember each pixel has three numbers.

**Answer:** 100 times 100 is 10,000 pixels, times 3 is 30,000 numbers.

Halving both the width and the height gives a quarter of the pixels, so a quarter of the 120,000 numbers.

2. Explain why cropping internet photos to one standard size is only a partial fix for varying-length data.

**Hint:** What happens to the parts of the picture outside the crop?

**Answer:** Cropping makes every example the same length, but whatever is cut off is lost, and it may hold what matters for the prediction. And it does not help at all with text, which varies in length even more.

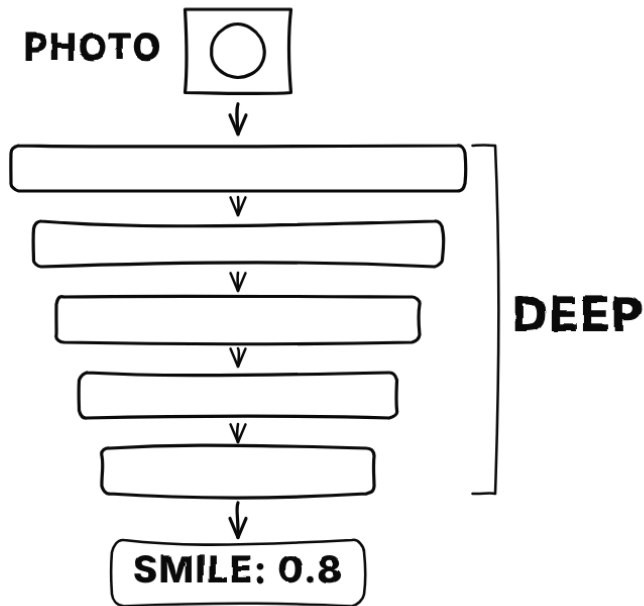
**3.** Give the four parts that every machine learning problem has, and say in a sentence what each one does.

**Hint:** One is what we learn from, one turns it into answers, one scores, one adjusts.

**Answer:** The data is what the machine learns from; the model turns the data into answers; the objective function scores how well or badly the model is doing; and the algorithm adjusts the model's parameters to make that score as good as possible.

# Scoring the machine

Think of a student revising for a final exam with last year's practice papers. After a week she scores full marks on every one of them. Then, in the real exam, faced with questions she has never seen, she struggles. She had not learned the subject; she had learned the answers to those particular papers. Keep her in mind, because learning machines can fail in exactly the same way.

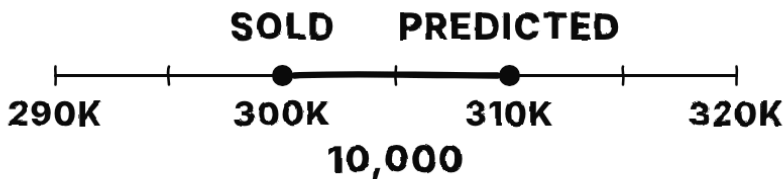


Deep means many steps

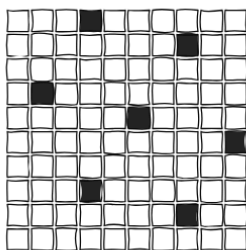
First, what is a model? Most machine learning takes data of one kind and turns it into a prediction, often of another kind. The book's

examples are a system that takes in photos and says how much the person is smiling, and one that takes in sensor readings and says how normal or how unusual they are. The model is the machinery that does this turning, and the models the book cares about are statistical ones, whose knobs are set from data. Simple models are fine for simple problems, but the problems in this book go beyond what the classic methods can do. What sets deep learning apart is its family of powerful models built as many transformations of the data, one after another, chained from top to bottom. That stack of steps is why it is called deep. The book will meet some older, simpler methods along the way too.

Learning was described as getting better at a task with experience. But who decides what counts as better? One person may think a change to the model is an improvement, and another may not. To build machines that learn, we need a precise number that says how good or how bad a model is. That number is called the objective function. By custom it is set up so that lower is better, and then it is also called a loss function, a measure of how much is lost. That is only a convention: any score where higher is better can be turned into one where lower is better just by putting a minus sign in front of it.



$$10,000^2 = 100,000,000$$



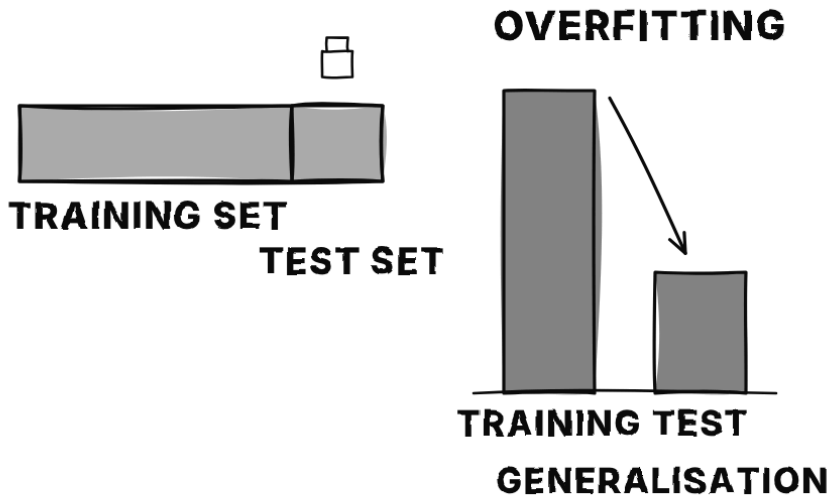
**7 WRONG = 7 %**

How much is lost

The book names the two most common losses. When the model predicts a number, the usual loss is the squared error: take the difference between the prediction and the true value and multiply it by itself. Suppose a model predicts that a flat will sell for three hundred and ten thousand, and it sells for three hundred thousand. The difference is ten thousand; ten thousand times ten thousand is one hundred million, and that is the loss for that one example. When the model sorts things into kinds, the usual measure is the error rate, the share of examples it gets wrong: seven mistakes in a hundred is an error rate of seven percent. Squared error is easy to optimise, but the error rate is hard to improve directly, so in practice one often minimises a stand-in score, called a surrogate objective, that tends to move the same way. A spam filter is judged in the end by the share of

mail it sorts wrongly, even if it is trained on a smoother stand-in score.

While the model is being trained, the examples are held fixed and the loss is treated as depending only on the knobs. Training means finding the knob settings that make the loss on the training examples as small as possible. But doing well on those examples does not guarantee doing well on new ones. So the data is usually split into two piles. The training set is used to set the knobs; the test set is kept aside, unseen, and used only to check the result. Results are reported for both.



Training set and test set

Here the student returns. The score on the training set is her score on the practice papers: encouraging, but no promise. If the model has, in effect, memorised its training examples, it shines on them and stumbles on anything new. Doing well on examples it has never seen is called generalisation. Doing well on the training set but failing to generalise is called overfitting. When a recommendation feed seems to know your past clicks perfectly but suggests nothing you actually want next, that is the kind of failure a test set is meant to catch.

### **Remember**

A model turns data of one kind into predictions, and deep learning's models are long chains of transformations, which is why they are called deep. To say whether a model improved, we need a number, the objective function, usually set up so that lower is better and then called a loss. For predicting numbers the common loss is the squared difference from the truth; for sorting into kinds it is the share of mistakes, often trained through a smoother stand-in. Knobs are tuned on a training set and checked on a separate test set, because success on familiar examples is not success on new ones. A model that shines on its training examples but fails on new ones is overfitting, like a student who memorised the practice papers.

---

## **Questions**

1. Why is deep learning called deep?

- a. Because its models need very large datasets
  - b. Because its models are many transformations of the data chained one after another
  - c. Because it studies hidden properties of data
  - d. Because its losses are very small
2. A score where higher is better is turned into a loss. How?
- a. By squaring it
  - b. By dividing it by the number of examples
  - c. By putting a minus sign in front of it
  - d. By measuring it on the test set
3. Why is a surrogate objective often used instead of the error rate?
- a. Because the error rate is hard to improve directly, while a smoother stand-in tends to move the same way
  - b. Because the error rate is always zero on training data
  - c. Because the error rate only works for numbers
  - d. Because the surrogate is measured on the test set
4. A model scores very well on its training set but poorly on the test set. What is this called?
- a. Generalisation
  - b. A surrogate objective
  - c. A high error rate on training
  - d. Overfitting

## **Answers**

1. **b)** Deep learning's models are built as many transformations of the data chained from top to bottom, and that stack of steps is why it is called deep.
2. **c)** Any score where higher is better can be turned into one where lower is better just by putting a minus sign in front of it.
3. **a)** The error rate is hard to improve directly, so one often minimises a stand-in score that tends to move the same way.
4. **d)** Doing well on the training set but failing to generalise to unseen data is called overfitting.

## Worked examples

**One squared error** A model predicts that a flat will sell for 310,000, and it actually sells for 300,000. What is the squared error for this example?

Find the difference between prediction and truth: 310,000 minus 300,000 is 10,000.

Square it, that is, multiply it by itself: 10,000 times 10,000.

That gives 100,000,000.

The loss is large because squaring makes big misses count far more than small ones.

**Answer:** 100,000,000 (one hundred million)

## Practice

1. A model sorting photos into cats and dogs gets 14 of 200 test photos wrong. What is its error rate?

**Hint:** The error rate is the share of examples it gets wrong.

**Answer:** 14 divided by 200 is 0.07, an error rate of seven percent.

**2.** A model predicts 8 where the truth is 5, and on another example 2 where the truth is 3. Which example adds more to the squared error loss, and by how much each?

**Hint:** Take each difference and multiply it by itself.

**Answer:** The first misses by 3, so its squared error is 9; the second misses by 1, so its squared error is 1. The first adds nine times as much.

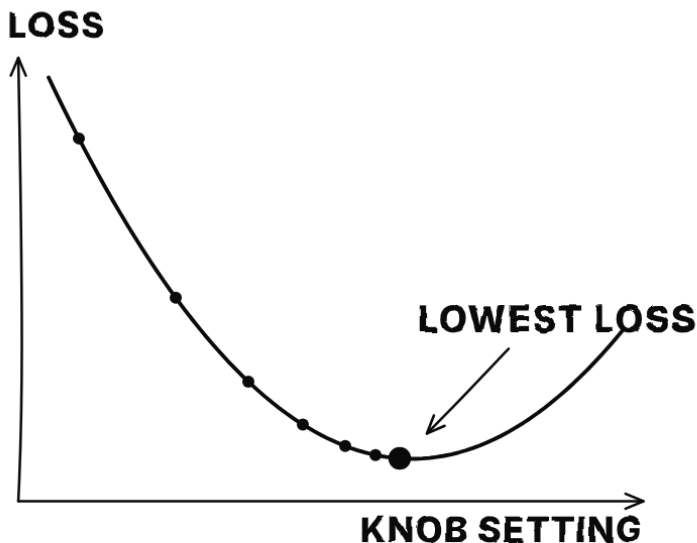
**3.** Explain with the student and the practice papers why we keep a test set apart from the training set.

**Hint:** What does a perfect score on practice papers she has memorised tell you about the real exam?

**Answer:** A model tuned on the training set may simply memorise it, just as the student memorised the practice papers. Only examples it has never seen, the test set, show whether it has really learned something that carries over, that is, whether it generalises.

# Feeling for the way down

Stand on a hillside in thick fog, wanting to reach the valley floor. You cannot see more than a step ahead, so you cannot see where the valley is. Yet you can feel the slope under your feet: one way the ground falls, the other way it rises. Take a small step the way it falls, feel again, and step again. Sooner or later you are standing at the bottom.

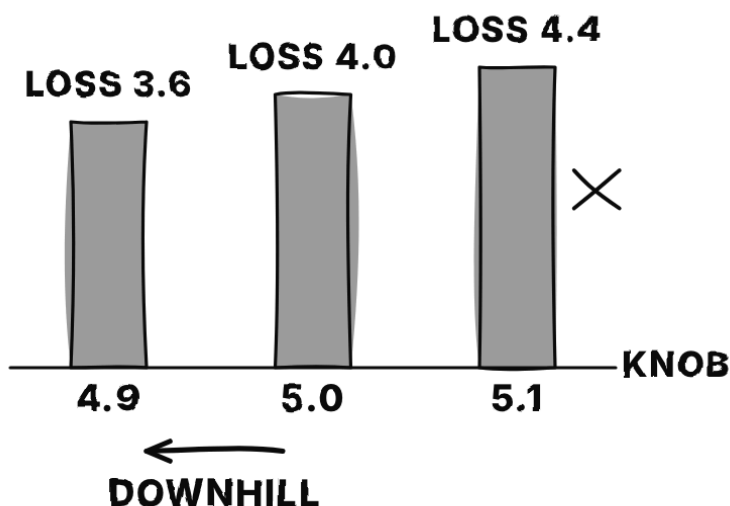


Walking downhill on the loss

That is the idea behind the method the book now names, the last of the four parts. We have data, a model with knobs, and a loss that says how badly it is doing; now we need a way to find the knob settings

that make the loss as small as possible. Most methods used in deep learning are built on gradient descent. At each step it asks, for every knob in turn, how the loss on the training set would change if that knob were turned just a tiny bit. Then it turns each knob a little in the direction that lowers the loss. The loss is the height of the hillside, and the settings of the knobs are where you stand.

Here is one step with numbers, for a model with a single knob. Say the knob is at five, and the loss there is four. Turn it a tiny bit up, to five point one, and the loss rises to about four point four; so up is the wrong way. Turn it instead a tiny bit down, to four point nine, and the loss falls to about three point six. So the method moves the knob downwards, and then checks the slope again from its new place. A real deep learning model has millions of knobs, but each step treats every one of them the same way.



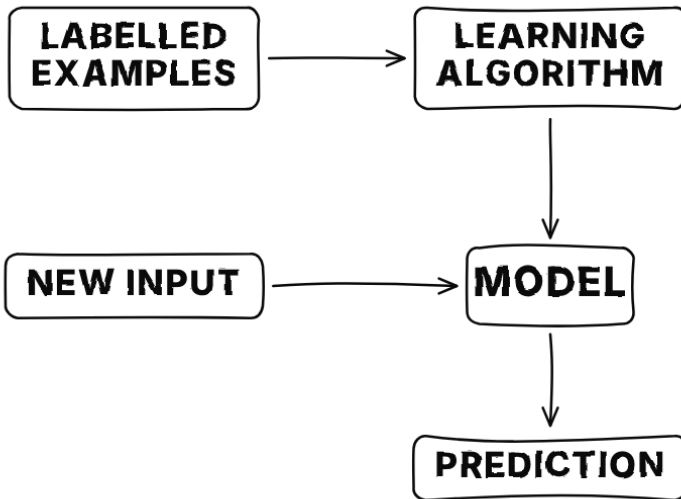
One step with numbers

With the four parts in place, the book turns to the kinds of problem machine learning can take on, starting with the most common one, supervised learning. Here we are given examples whose features are known and whose labels are known too, and the job is to build a model that predicts the label from the features. Each pair of features and label is one example. It is called supervised because we, the supervisors, hand the model labelled examples to choose its knobs from. In the language of probability, what we want is how likely each label is, given the features we can see.

Supervised learning is only one of several approaches, but it accounts for most of the successful uses of machine learning in industry. Many important tasks fit its simple pattern: guess something unknown from

the data at hand. Is there cancer or not, given a scan from a medical scanner? What is the right French translation of this English sentence? What will a share price be next month, given this month's financial reports? Your spam filter is the same pattern: given a message, predict spam or not spam. The pattern is simple, but the forms vary hugely; sequences of any length, for instance, need different models from inputs of a fixed size.

The book sketches the process, and draws it as a picture: labelled examples go into a learning method, a model comes out, and new inputs go into that model and come out as predictions. First gather a big collection of examples whose features are known, choose some of them at random, and get the true label for each one. Sometimes the labels are already on record, such as whether a patient died within the following year; sometimes people must be paid to label the data by hand, for instance by sorting pictures into categories. These examples with their labels make up the training set. The training set goes into the learning algorithm, which is itself a function whose output is another function, the trained model. Finally, inputs the model has never seen are fed into it, and its outputs are taken as predictions of their labels.



From labelled examples to predictions

### Remember

Gradient descent is how the knobs are tuned: for each knob, check whether a tiny turn raises or lowers the loss, then turn it a little the lowering way, and repeat. It is like finding the valley in fog by feeling the slope underfoot. Supervised learning means learning to predict labels from features, using examples whose labels are given to the model. Most successful applications in industry are of this kind, from medical scans to translation. The process runs from labelled examples, through a learning algorithm, to a trained model that makes predictions on new inputs.

---

## Questions

1. In the foggy hillside picture of gradient descent, what does the height of the ground stand for?
  - a. The number of knobs
  - b. The loss on the training set
  - c. The size of the dataset
  - d. The number of steps taken
2. A tiny turn of a knob upwards makes the loss rise. What does gradient descent do next?
  - a. Turns the knob further upwards
  - b. Leaves the knob alone for good
  - c. Turns the knob a little downwards, then checks the slope again
  - d. Sets the knob back to a random value
3. Why is this kind of learning called supervised?
  - a. Because a person watches the model while it runs
  - b. Because we hand the model labelled examples to choose its knobs from
  - c. Because the model's mistakes are corrected by hand afterwards
  - d. Because the model works only under a supervisor's control
4. What does the learning algorithm give out at the end?
  - a. A trained model, itself a function that turns new inputs into predictions

- b. A list of labels for the training set
- c. A cleaned copy of the dataset
- d. A single number, the loss

## Answers

- 1. **b)** The loss is the height of the hillside, and the settings of the knobs are where you stand.
- 2. **c)** If up is the wrong way, the method moves the knob downwards and then checks the slope again from its new place.
- 3. **b)** It is called supervised because we, the supervisors, hand the model labelled examples to choose its knobs from.
- 4. **a)** The learning algorithm is a function whose output is another function, the trained model.

## Worked examples

**One step downhill** A model has a single knob set at 5, and its loss there is 4. Turning the knob to 5.1 makes the loss about 4.4; turning it to 4.9 makes the loss about 3.6. Which way does gradient descent move the knob?

Compare each tiny turn with the loss now, which is 4.

Up to 5.1 raises the loss to about 4.4, so up is uphill.

Down to 4.9 lowers the loss to about 3.6, so down is downhill.

Gradient descent turns the knob a little downwards and then checks the slope again.

**Answer:** downwards, towards smaller values of the knob

## Practice

**1.** For each task, name the features and the label: spotting cancer in a scan; translating an English sentence into French; predicting next month's share price from this month's financial reports.

**Hint:** The features are what the model is shown; the label is what it must predict.

**Answer:** Scan: the features are the scan image, the label is cancer or not cancer. Translation: the features are the English sentence, the label is its French translation. Shares: the features are this month's financial reports, the label is next month's share price.

**2.** Put the supervised learning process in order: feed new inputs to the model; get the true label for each chosen example; gather examples and pick some at random; feed the training set to the learning algorithm.

**Hint:** Labels have to exist before anything can be learned.

**Answer:** Gather examples and pick some at random; get the true label for each; feed the labelled training set to the learning algorithm, which gives out a trained model; then feed new inputs to the model and use its outputs as predictions.

# How much, how many

A plumber spends three hours clearing your blocked drains and sends a bill for three hundred and fifty dollars. Your friend hires the same plumber for two hours and pays two hundred and fifty. Now a neighbour asks what a four-hour job will cost. Before reading on, see if you can answer.

$3\text{ H} \rightarrow \$350 \cdot 2\text{ H} \rightarrow \$250 \cdot 4\text{ H} \rightarrow ?$

| AREA | BEDS | BATHS | WALK |
|------|------|-------|------|
| 600  | 1    | 1     | 60   |
| 3000 | 4    | 3     | 10   |

↑

FEATURE VECTOR

Bills, and houses as rows

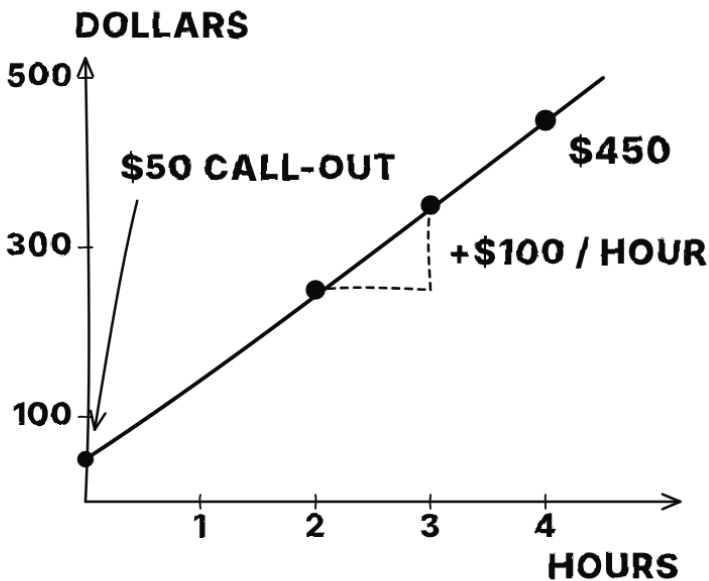
The simplest kind of supervised learning to get your head round is called regression. Picture a table of house sales, one row for each house and one column for each fact about it: floor area, number of

bedrooms, number of bathrooms, and minutes on foot to the town centre. Each house is one example, and its row of facts is its feature vector, a fixed list of numbers in a fixed order. The book jokes that in New York or San Francisco, unless you run one of the giant tech companies, your home might read six hundred square feet, one bedroom, one bathroom, sixty minutes' walk. In Pittsburgh it might read three thousand square feet, four bedrooms, three bathrooms, ten minutes' walk. Most classic learning methods need every example to arrive as such a list, always of the same length.

What makes a problem regression is not the features but the kind of answer. If you are house hunting, you might want to estimate a fair price from facts like these, and a model could learn it from past listings, with the prices the houses actually sold for as the labels. When the label can be any number, even within some range, the problem is regression, and the goal is a model whose guesses come close to the true values. A useful rule of thumb: a question beginning "how much?" or "how many?" is probably regression. How many hours will this operation take? How much rain will fall on this town in the next six hours? How long will this patient stay in hospital? Guessing how many stars you would give a film is regression as well, and the book recalls that doing it well in 2009 could have won you Netflix's prize of a million dollars.

Back to the plumber. You probably assumed two things: more hours cost more, and the bill is a fixed charge for turning up plus so much for each hour. The two bills differ by one hour and by one hundred dollars, so each hour costs one hundred dollars. Three hours at a hundred is three hundred, and the bill was three hundred and fifty, so turning up costs fifty dollars. Check it on your friend's bill: fifty plus two hours at a hundred is two hundred and fifty, exactly right. So the

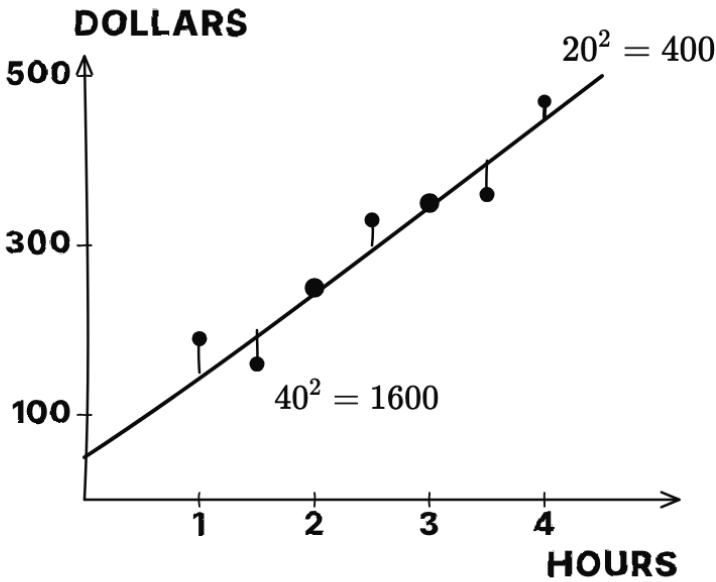
four-hour job costs fifty plus four hundred, which is four hundred and fifty dollars. If you followed that, you already understand the idea behind linear regression: the answer is a fixed amount plus a set amount for each unit of each feature.



A call-out charge plus a rate

Here two parameters, the hourly rate and the call-out charge, matched the bills exactly. Real data is rarely so tidy, because some of the variation comes from things not in the table: a stubborn blockage, a long drive, traffic. Then no setting of the knobs fits every example, and instead we look for the one that keeps the predictions as close as possible to what was actually observed. The book mostly measures closeness by the squared error, the miss multiplied by itself. Suppose a third job, four hours long, was billed at four hundred and seventy:

the rule predicts four hundred and fifty, the miss is twenty dollars, and the squared error is twenty times twenty, four hundred. A miss twice as large would count four times as much, so squaring makes big misses weigh far more than small ones. The authors note that this choice of loss is the natural one when the stray variations behave like the commonest kind of random noise, the bell-shaped kind.



Misses, squared

## Remember

Regression is supervised learning where the answer is a number; "how much?" and "how many?" questions are usually regression. Each example is a feature vector, a fixed-length list of facts, like the floor area, bedrooms, bathrooms and walking time of a house. The plumber's two bills reveal a call-out charge of fifty dollars and a rate of a hundred an hour: the idea of linear regression. When real data does not fit exactly, we choose the parameters that keep predictions closest to the truth, usually by making the squared error as small as possible.

---

## Questions

1. What makes a problem regression rather than some other kind?
  - a. That the features are numbers
  - b. That the label, the answer wanted, is a number
  - c. That the data comes in a table
  - d. That there are only two features
2. Why can a model trained on real plumbers' bills usually not match every bill exactly?
  - a. Because things not in the table, like a stubborn blockage or traffic, also change the bills
  - b. Because there are too many features
  - c. Because squared error cannot be computed for money
  - d. Because each bill has a different call-out charge by law

3. One prediction misses by 10, another by 20. How do their squared errors compare?
- a. The second counts twice as much
  - b. They count the same
  - c. The second counts four times as much
  - d. The second counts half as much

## Answers

1. **b)** What makes a problem regression is not the features but the kind of answer: when the label can be any number, it is regression.
2. **a)** Some of the variation comes from things not in the table, so no setting of the knobs fits every example.
3. **c)** A miss twice as large counts four times as much, because the miss is multiplied by itself.

## Worked examples

**The plumber's pricing** A plumber charged 350 dollars for a 3-hour job and 250 dollars for a 2-hour job. Assuming a fixed charge for turning up plus a rate per hour, find the charge and the rate, and the cost of a 4-hour job.

The bills differ by one hour and by 100 dollars, so the rate is 100 dollars an hour.

Three hours cost 300 dollars of the 350, so the fixed charge is 50 dollars.

Check: 50 plus 2 hours at 100 makes 250, as billed.

A 4-hour job costs 50 plus 400.

**Answer:** a fixed charge of 50 dollars and 100 dollars an hour; a 4-hour job costs 450 dollars

## Practice

**1.** A third 4-hour job was billed at 470 dollars, while the rule of 50 dollars plus 100 an hour predicts 450. A fourth, 1-hour job was billed at 140 dollars. What is the squared error of the rule on each job?

**Hint:** First work out what the rule predicts for one hour, then take each miss and multiply it by itself.

**Answer:** For the 4-hour job the miss is 20 dollars, so the squared error is 400. For the 1-hour job the rule predicts 150, the miss is 10 dollars, and the squared error is 100.

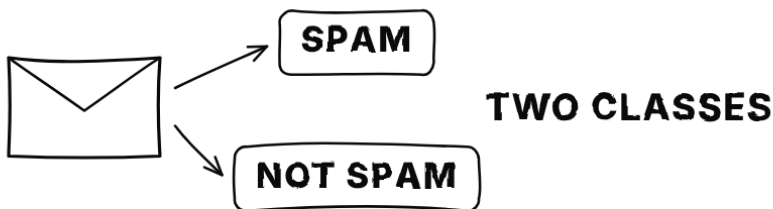
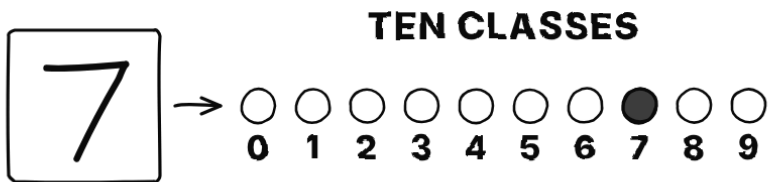
**2.** Write the feature vector of a house with 1,200 square feet, 2 bedrooms, 1 bathroom and a 25-minute walk to the centre, in the book's order, and say what label a regression model would predict for it.

**Hint:** The book's order is floor area, bedrooms, bathrooms, walking minutes.

**Answer:** The feature vector is 1,200, 2, 1, 25. A regression model trained on past sales would predict a number, the house's price.

## Which one, and how sure

Picture a beautiful pale mushroom in your back garden. You photograph it with an app trained to spot poisonous mushrooms, and the app says it is eighty percent sure the mushroom is harmless. Most likely, then, it is safe. Would you cook it for dinner? Keep your answer in mind; the book returns to this very mushroom.



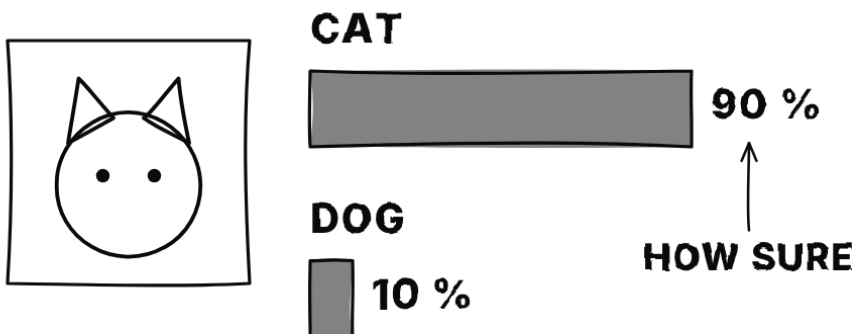
Which class is it?

Regression answers "how much?" and "how many?", but plenty of questions are of another kind. Take a bank that wants its phone app to read a cheque from a photo. Suppose the picture has already been cut

into small patches, one for each handwritten character. What is left is to decide, for each patch, which character out of a known set it shows. These "which one?" questions are called classification. They need tools of their own, though many ideas carry over from regression.

In classification the model looks at the features, such as the pixels of an image, and says which category, also called a class, the example belongs to, out of a fixed set of options. For handwritten digits there are ten classes, nought to nine. With only two classes, such as cat and dog, it is called binary classification. Your spam filter is binary classification: every message ends up in one of two classes, spam or not spam. A regression model, a regressor, gives out a number; a classification model, a classifier, gives out a class. With more than two classes, as in reading handwriting with all ten digits and all the letters, it is called multiclass classification.

A model that can only give a firm answer, "cat" or "dog", turns out to be hard to train, for reasons the book will explain later. So it is usually made to give a probability for each class instead. Shown a photo, a classifier might give the chance of a cat as nought point nine, which we can read as "ninety percent sure it is a cat", and so ten percent for a dog. How large the probability of the chosen class is tells us how uncertain the model is, though the book notes this is not the only measure of uncertainty. Regression was trained by making the squared error small; classification has its own usual loss, called cross-entropy, a name the book promises to explain when it comes to information theory.



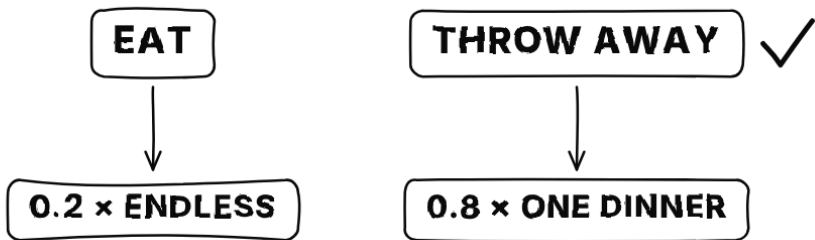
A probability for each class

Now the mushroom. The poison-spotting classifier gives a chance of nought point two that it is a death cap, so it is eighty percent sure it is not. Still, only a fool would eat it, because the sure pleasure of a good dinner is not worth a one-in-five risk of dying. To decide, the book weighs each action: for each possible outcome, multiply its probability by how much harm it would do, and add these up. If you eat the mushroom, there is a twenty percent chance of a harm so great we can call it endless, and an eighty percent chance of no harm; twenty percent of something endless is still endless. If you throw it away, there is a twenty percent chance you lose nothing, because it was poisonous anyway, and an eighty percent chance you lose one dinner, counted as one: nought point eight times one gives a loss of just nought point eight. Nought point eight against endless: throwing it

away wins by far. The most likely class is not necessarily the one to act on. And caution was right, for the mushroom in the book's picture really is a death cap.

 **SAFE 80 %**

 **DEATH CAP 20 %**

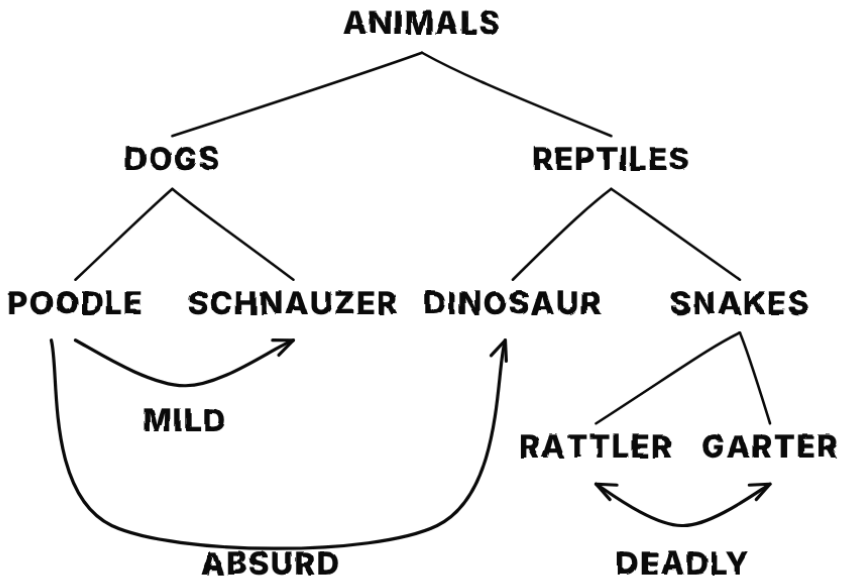


Weigh each action by its harm

### Try it yourself

The next time an app gives you a confidence figure, ask what each outcome would cost you. A ninety percent "you'll like this song" is plenty to press play; a ninety percent "this medicine is fine with your other pills" is not nearly enough on its own. The same probability can call for opposite actions when the costs differ.

Classification can be more intricate still. Sometimes the classes form a family tree, much as the naturalist Linnaeus arranged animals in nested groups. Then not all mistakes are equal: if the model must get it wrong, better to land on a close relative than on a distant one. This is called hierarchical classification. Mistaking a poodle for a schnauzer is a small slip; mistaking a poodle for a dinosaur deserves a huge penalty. But which tree counts depends on what the model is for. Rattlesnakes and harmless garter snakes sit close together in the family tree of snakes, yet taking a rattler for a garter snake could kill you.



Some mistakes are worse

## Remember

Classification answers "which one?", placing each example in one class out of a fixed set: binary with two classes, multiclass with more. Classifiers usually give a probability for each class, which also tells how sure they are, and they are commonly trained with a loss called cross-entropy. Acting well means weighing each choice by the probability of each outcome times its harm, not just picking the likeliest class. For the mushroom, eating carries an endless expected harm and throwing it away only nought point eight of a dinner. When classes form a tree, some mistakes are worse than others, and which ones depends on the use, as with the rattlesnake and the garter snake.

---

## Questions

1. A classifier says a photo is a cat with probability 0.9. How should this be read?
  - a. Nine in ten photos it sees are cats
  - b. It is ninety percent sure this photo shows a cat
  - c. The photo is ninety percent cat and ten percent dog
  - d. It was right on ninety percent of its training set
2. Why is it wise to throw away a mushroom the app thinks is eighty percent likely to be safe?
  - a. Because eighty percent means the app is broken

- b. Because the weighed harm of eating, a small chance of death, far outweighs the weighed harm of losing a dinner
  - c. Because the most likely class is always wrong
  - d. Because mushrooms cannot be classified
- 3. In hierarchical classification, which mistake deserves the smallest penalty?
  - a. Mistaking a poodle for a dinosaur
  - b. Mistaking a poodle for a schnauzer
  - c. Mistaking a dog for a fish
  - d. All mistakes deserve the same penalty
- 4. Why does confusing a rattlesnake with a garter snake count as a serious mistake though they are close relatives?
  - a. Because the cost of the mistake depends on what the model is used for, and here it could kill
  - b. Because they belong to different families of animals
  - c. Because the model was trained on too few snakes
  - d. Because snakes are always multiclass

## Answers

1. **b)** A chance of a cat of nought point nine can be read as "ninety percent sure it is a cat".
2. **b)** Weighing each action by the probability of each outcome times its harm, eating carries an endless expected harm and throwing away only nought point eight.
3. **b)** If the model must get it wrong, better to land on a close relative: mistaking a poodle for a schnauzer is a small slip.

4. a) Which tree counts depends on what the model is for; taking a rattler for a garter snake could be deadly.

## Worked examples

**To eat or not to eat** A classifier gives a probability of 0.2 that a mushroom is a death cap and 0.8 that it is not. Eating a death cap is an endless harm; eating a good mushroom costs nothing. Throwing away a death cap costs nothing; throwing away a good mushroom costs one dinner, counted as 1. Which action has the smaller expected harm?

For each action, multiply each outcome's harm by its probability and add.

Eating: 0.2 times an endless harm, plus 0.8 times 0; the first term alone is endless.

Throwing away: 0.2 times 0, plus 0.8 times 1, which is 0.8.

0.8 is far smaller than an endless harm, so throw the mushroom away, though "safe" is the likelier class.

**Answer:** throwing it away: an expected harm of 0.8 against an endless one for eating

## Practice

1. Another mushroom app gives only a 0.01 chance of a death cap. Using the same harms, what is the expected harm of eating and of throwing away? What does this tell you?

**Hint:** Any positive chance times an endless harm is still endless.

**Answer:** Eating: 0.01 times an endless harm is still endless. Throwing away: 0.99 times 1 is 0.99. Throwing away still wins: when one outcome is catastrophic, even a small chance of it outweighs a certain small loss.

**2.** Say whether each is binary, multiclass, or regression: reading which of the digits 0 to 9 is in a cheque patch; deciding cat or dog; predicting a house price; reading any letter or digit.

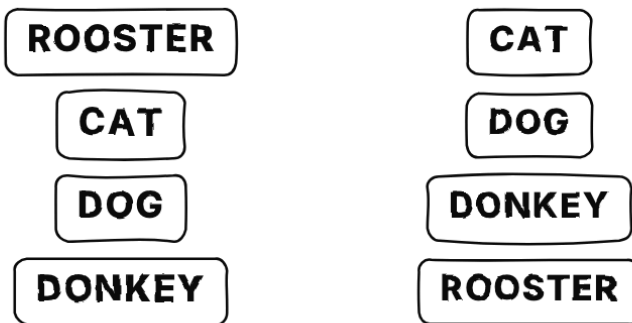
**Hint:** Count the classes, or ask whether the answer is a number.

**Answer:** Digits 0 to 9 and letters plus digits are multiclass classification; cat or dog is binary classification; the house price is a number, so it is regression.

# Many labels, and the best first

Picture the Town Musicians of Bremen from the old German fairy tale: a donkey, a dog standing on its back, a cat on the dog, and a rooster on top, with trees behind them. Now show that picture to a classifier trained to answer "cat or dog?". These days such a classifier is easy to build and can be very accurate. Yet here it is bound to be wrong, because it must pick exactly one answer, and the true answer is "both, and a donkey and a rooster too".

## MULTI-LABEL

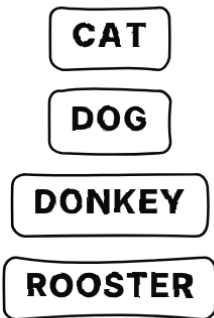


One picture, several labels

The trouble is not the model but the question. If pictures like this are expected, choosing one class out of many is the wrong way to set up the problem. The model should be allowed to say: a cat, and a dog, and a donkey, and a rooster. Predicting classes that can be true at the same time is called multi-label classification.

Here is one way to see how much bigger this question is. With four animals and the rule "pick exactly one", there are four possible answers. If each animal may be present or absent independently, each has two possibilities, so there are two times two times two times two, which is sixteen possible answers, from no animal at all to all four. Each extra label doubles the count again.

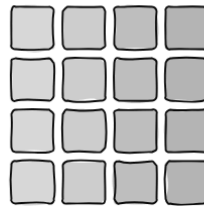
### **PICK ONE**



**= 4**

### **ANY OF THEM**

$$2 \times 2 \times 2 \times 2 = 16$$

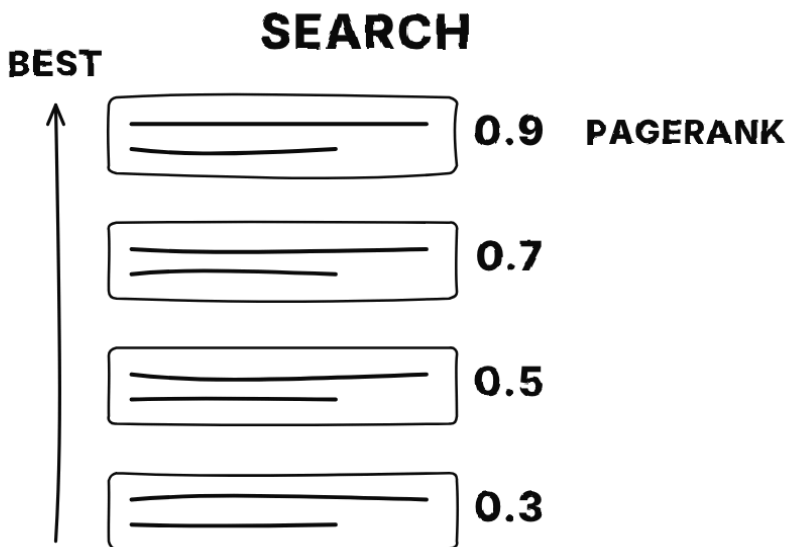


**= 16**

Multi-label classification is the natural description of automatic tagging. Think of the tags on posts in a technology blog: "machine learning", "technology", "gadgets", "programming languages", "cloud computing". A typical article might carry five to ten of them. And tags come in company: posts about cloud computing tend to mention Amazon's cloud service, posts about machine learning tend to mention the graphics chips it runs on. The suggested hashtags an app offers when you post a photo are this kind of guess: several tags at once, not one.

Sometimes the set of tags is huge. At America's National Library of Medicine, trained staff tag every article entering its great medical index with terms from a list of roughly twenty-eight thousand subject headings. Good tags let researchers find everything ever written on a question, but the work is slow, and an article typically waits about a year to be tagged. Machine learning can supply provisional tags until a person reviews each article, and for several years there have been competitions on doing exactly this.

Search turns the question round again. For a web search, there are usually many relevant pages; the real question is which of them to show first. One way is to give every page a score and return the highest-scoring ones. PageRank, the original secret ingredient of Google's search engine, was an early score of this kind. Strangely, PageRank did not depend on the words of the search at all: a simple filter first picked out the relevant pages, and PageRank then put the more authoritative ones at the top. Today search engines use machine learning and models of how people behave to compute scores that do depend on the search. There are whole academic conferences devoted to this one problem.



Search ranks by score

## Remember

A classifier forced to pick one class fails on a picture of a donkey, a dog, a cat and a rooster together. Multi-label classification lets several classes be true at once, and with four labels there are already sixteen possible answers. It is the natural form of automatic tagging, from blog posts to the twenty-eight thousand medical headings of a great library. Search is about ranking: give relevant items a score and show the best first, as PageRank once did without even looking at the search words.

---

## Questions

1. Why must even a very accurate cat-or-dog classifier be wrong on the Bremen picture?
  - a. Because the animals are drawn, not photographed
  - b. Because it must pick exactly one class, while the true answer is several at once
  - c. Because there are trees in the background
  - d. Because donkeys were not in its training set
2. With five possible labels, each of which may be present or absent, how many possible answers are there?
  - a. 5
  - b. 10
  - c. 25
  - d. 32
3. What did a search engine with PageRank do with the words you typed?
  - a. A simple filter used them to pick the relevant pages, and PageRank then ordered those pages without looking at the words
  - b. PageRank scored every page by how often your words appeared
  - c. It ignored them and showed the same pages to everyone
  - d. It used them to tag each page with multiple labels

## Answers

1. **b)** It must pick exactly one answer, and the true answer is both, and a donkey and a rooster too: the trouble is the question, not the model.
2. **d)** Each extra label doubles the count: four labels give sixteen answers, so five give thirty-two.
3. **a)** PageRank did not depend on the words of the search: a simple filter picked the relevant pages, and PageRank put the more authoritative ones on top.

## Worked examples

**Counting possible answers** A picture may contain any of four animals: a donkey, a dog, a cat and a rooster. How many possible answers are there if the model must pick exactly one animal, and how many if each animal may be present or absent?

Picking exactly one of four animals gives 4 possible answers.

If each animal may be present or absent, each has 2 possibilities.

The choices for the four animals combine: 2 times 2 times 2 times 2.

That gives 16, from no animal at all to all four together.

**Answer:** 4 answers with one class; 16 with multi-label classification

## Practice

1. A blog uses 10 possible tags, and any post may carry any of them. How many different sets of tags could a post have?

**Hint:** Each tag is either on or off, and each extra tag doubles the count.

**Answer:** 2 multiplied by itself 10 times, which is 1,024, counting the post with no tags at all.

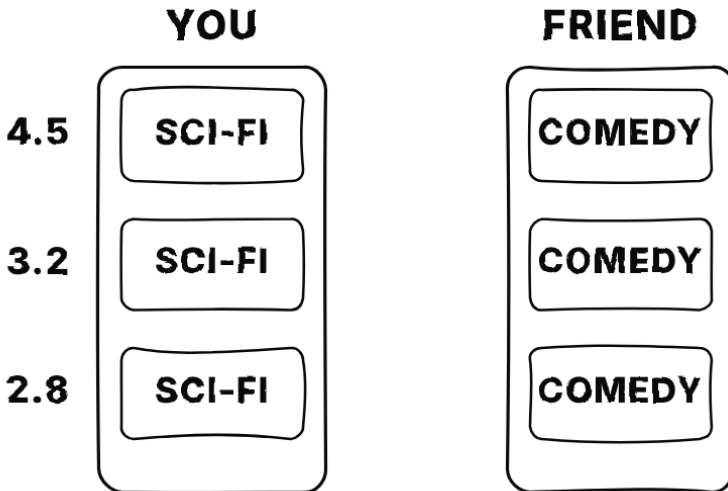
**2.** Explain why a search engine needs ranking, not just a yes or no for each page.

**Hint:** How many relevant pages does a typical search find?

**Answer:** A search usually finds many relevant pages, so knowing whether each page is relevant is not enough; the engine must decide which to show first. It gives each page a score and shows the highest-scoring ones at the top.

# Your feed, and things that unfold in time

Open the film app on your phone and then hand it to a friend with very different tastes. On your screen the top row is science fiction; on theirs, old comedies. Same app, same films in its library, two different pages. How does it decide what to put first for each of you?



Same app, different first rows

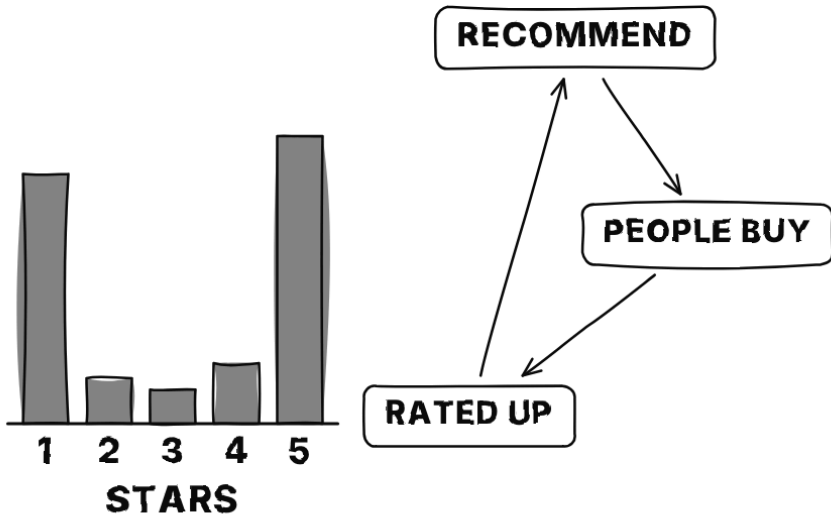
Recommender systems are close cousins of search and ranking: both pick a set of items and decide which to show first. The difference is that a recommender is personal. A science fiction fan and a lover of

Peter Sellers comedies should get very different results pages, and the same is true for shopping, music and news.

To learn what people like, the system listens to two kinds of signal. Sometimes people say it outright, with star ratings and reviews on a shop, a film site or a book site; that is explicit feedback. More often they say it only through what they do, such as skipping a song on a playlist; that is implicit feedback. A skip may mean dislike, or only that the song did not suit the moment. When your recommendation feed reshuffles after you skip a few videos, it is acting on this kind of quiet signal. In the simplest version, the system learns a score for every pair of a person and an item, such as the number of stars that person would probably give, or the chance they will buy it. Then for each person it shows the items with the highest scores. Suppose the model expects you to give one film four point five stars, another three point two and a third two point eight: it puts them in that order, the four and a half first. Real systems are far more elaborate, taking in detailed activity and the features of each item; the book shows the deep learning books a big online shop suggested to one of its own authors.

These systems are hugely valuable, yet built naively on top of a predictive model they have serious flaws. First, the feedback is censored: people mostly rate what they feel strongly about. On a five-star scale you see plenty of ones and fives and strangely few threes, so the ratings do not show what an average viewer thought. Second, what people buy today is partly the result of what the system recommended yesterday, and the learning method may not allow for that. So a feedback loop can form: an item is pushed, so more people buy it, so it looks better, so it is pushed even more. How to handle

censored feedback, incentives and such loops are still open research questions.



Lopsided ratings, self-feeding loops

Everything so far has had a fixed shape: the same list of facts for each house, the same grid of pixels for each photo, one fixed set of classes, a star rating from just a person and a product. And once the model had answered for an example, it forgot it, as if each new case had nothing to do with the last. That assumption, that successive examples are independent, often fails. A video clip can have any number of frames, and what is happening in one frame is much clearer if you look at the frames before and after. Language is the same: in machine translation a whole sentence comes in and a whole sentence must go out. In a hospital's intensive care unit, a model

might watch patients and raise an alarm whenever the risk of dying in the next twenty-four hours passes some threshold, and it would be foolish to throw away the patient's history every hour and judge only by the latest readings.

## **FRAMES AROUND IT**



**ANY LENGTH**



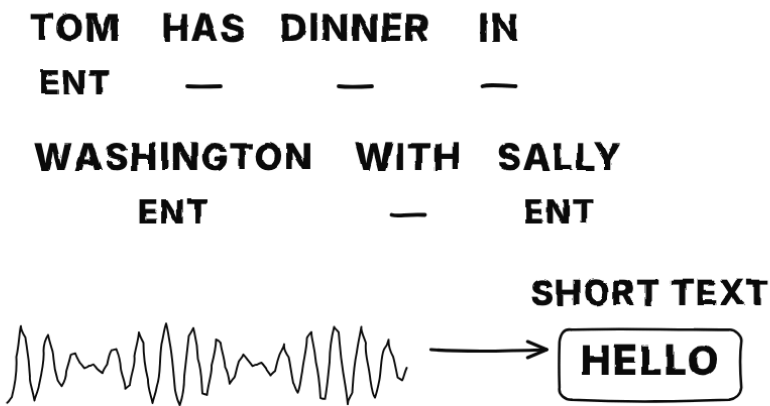
**PATIENT HISTORY**

One frame misleads, frames tell

Problems like these are called sequence learning: the model takes in a sequence, or gives one out, or both. When both input and output are sequences of any length, it is called sequence-to-sequence learning, as in translation, or in turning speech into text. The book names some special cases.

The first is tagging and parsing, where every word of a text gets a label, and input and output are aligned: as many labels as words, in

the same order. One might label each word with its part of speech, or mark which words belong to names of people, places or organisations. In the sentence "Tom has dinner in Washington with Sally", the words Tom, Washington and Sally would be marked as names, and the others not.



**THOUSANDS OF NUMBERS**

Tags in order; long sound, short text

The second is speech recognition: a recording goes in and a written transcript comes out. Sound is usually measured eight thousand or sixteen thousand times a second. At sixteen thousand a second, a word that takes half a second to say is eight thousand measurements, yet on the page it may be only five or six letters. So there is no one-to-one match between sound and text, and the output is far shorter than the input. The book's figure shows the words "deep learning" spread

out along a sound wave, each letter smeared across a stretch of it. Every time your voice assistant writes your question out on the screen, it has squeezed thousands of numbers into a few words. People are remarkably good at this, even with a crackly line; for computers it is a formidable challenge.

### **Remember**

Recommender systems rank items for each person, learning from explicit feedback such as ratings and implicit feedback such as skips. The simplest version scores every person and item pair, for instance by expected stars, and shows the top scores. Censored ratings and feedback loops, where recommendations feed their own success, are serious flaws that remain open problems. When examples depend on what came before, as in video, language or a patient's history, we need sequence learning, and sequence-to-sequence learning when both sides are sequences of any length. Tagging keeps one label per word, while speech recognition squeezes thousands of sound measurements per word into a few letters.

---

## **Questions**

1. You skip a song on a playlist. Why can the recommender not be sure you dislike it?
  - a. Because a skip is implicit feedback and may only mean the song did not suit the moment

- b. Because skips are never recorded
  - c. Because only star ratings count as feedback
  - d. Because the song was recommended to someone else
2. Why are there strangely few three-star ratings on a five-star scale?
- a. Because the system rounds them to one or five
  - b. Because feedback is censored: people mostly rate what they feel strongly about
  - c. Because three stars are not allowed on most sites
  - d. Because recommended items are always excellent
3. Why is a patient monitor in intensive care a sequence learning problem?
- a. Because each reading is independent of the last
  - b. Because it outputs a single class
  - c. Because judging only the latest readings would throw away the patient's history, which matters
  - d. Because the readings come as a fixed photo
4. Why is the output of speech recognition much shorter than the input?
- a. Because the recognizer drops half the words
  - b. Because each word is many thousands of sound measurements but only a few letters
  - c. Because transcripts are summaries
  - d. Because sound is measured only once a second

## Answers

1. **a)** A skip is a quiet, implicit signal: it may mean dislike, or only that the song did not suit the moment.
2. **b)** People mostly rate what they feel strongly about, so you see plenty of ones and fives and few threes.
3. **c)** It would be foolish to throw away the patient's history every hour and judge only by the latest readings.
4. **b)** At sixteen thousand measurements a second, a half-second word is eight thousand measurements but only five or six letters.

## Worked examples

**Marking the names** In the sentence "Tom has dinner in Washington with Sally", mark each word as a name (Ent) or not (-), as in simple named-entity tagging. How many labels are there, and how many are names?

Tagging is aligned: one label per word, in the same order.

The sentence has seven words, so there are seven labels.

Tom, Washington and Sally name a person, a place and a person; the other words do not.

The labels read: Ent, -, -, -, Ent, -, Ent.

**Answer:** seven labels, three of them names: Tom, Washington and Sally

## Practice

1. Sound is measured 8,000 times a second on a phone line. How many measurements make up a word that takes a quarter of a second to say, and why does this make speech recognition a problem of unequal lengths?

**Hint:** A quarter of a second is 8,000 divided by 4.

**Answer:** 8,000 times a quarter is 2,000 measurements, for a word of perhaps four or five letters. The input is thousands of numbers and the output only a few letters, so there is no one-to-one match and the output is far shorter than the input.

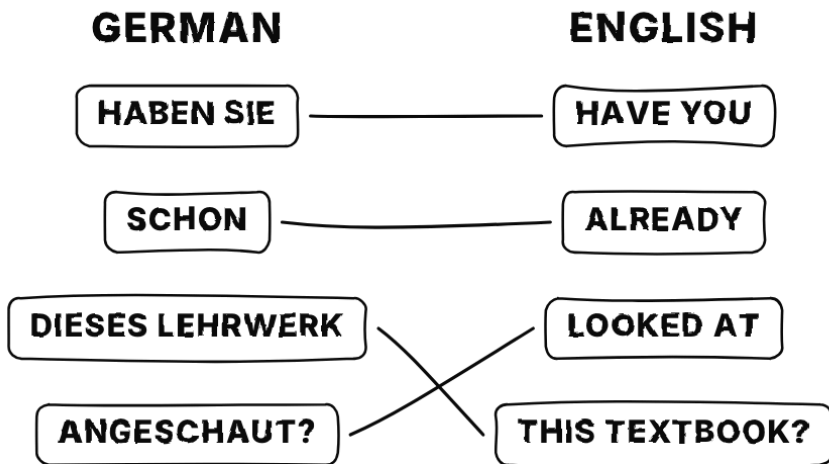
**2.** A model expects you to give three films 3.2, 4.5 and 2.8 stars. In what order does a simple recommender show them, and how could a feedback loop then make the first film look even better than it is?

**Hint:** Show the top scores first; then think about what happens to films that are shown more.

**Answer:** It shows the 4.5-star film first, then the 3.2 and then the 2.8. Being shown first, the top film is watched or bought more, those purchases make it look even better to the system, and it is pushed still harder: a feedback loop.

# Turning things round, and learning without answers

Type a short sentence into a translation app, say "Have you already looked at this excellent textbook?", and press the button that reads the German aloud. Two things happen that pull in opposite directions. The words are rearranged, because German puts things in a different order. And a handful of letters turns into a long stream of sound. Both are sequence problems, and both are harder than they look.



**VERB AT THE END**

Translation changes the order

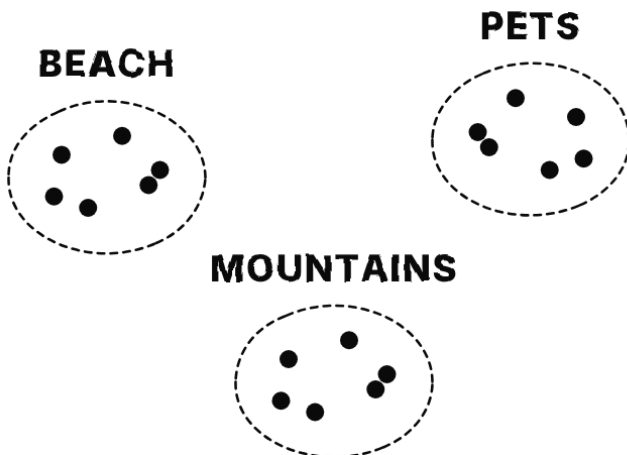
Reading text aloud is called text to speech, and it is speech recognition turned round. Now the input is text and the output is a sound recording, so the output is far longer than the input. At sixteen thousand measurements a second, a sentence that takes two seconds to say comes out as thirty-two thousand numbers, though it went in as a few dozen letters. When your voice assistant answers you out loud, it is doing exactly this.

Machine translation adds a new difficulty. In speech recognition, the sounds and the letters at least come in the same order. In translation, input and output can have different lengths, and the matching pieces can come in a different order. The book teases the German habit of putting the verb at the end. "Haben Sie sich schon dieses grossartige Lehrwerk angeschaut?" means "Have you already looked at this excellent textbook?" Kept in German order, word for word, it becomes "Have you yourself already this excellent textbook looked at?", which no English speaker would say. The book adds that related problems turn up elsewhere: working out the order in which someone reads a web page is a puzzle in two dimensions, and holding a conversation means drawing on knowledge of the world and on everything said long before. The chatbot you type to has to manage exactly that, keeping the whole conversation in view to decide what to say next.

All of this was supervised learning: the model was always given the right answers. The book compares it to a job with a dictatorial boss who stands over your shoulder and tells you exactly what to do in every situation. Dull, but easy to please: spot the pattern quickly and copy the boss. The opposite job is a boss who has no idea what they want, hands you a huge dump of data and says, "Do some data science with it!" That is vague because it is vague. Learning from data with no

labels at all is called unsupervised learning, and the questions one can ask are limited only by imagination.

## CLUSTERING

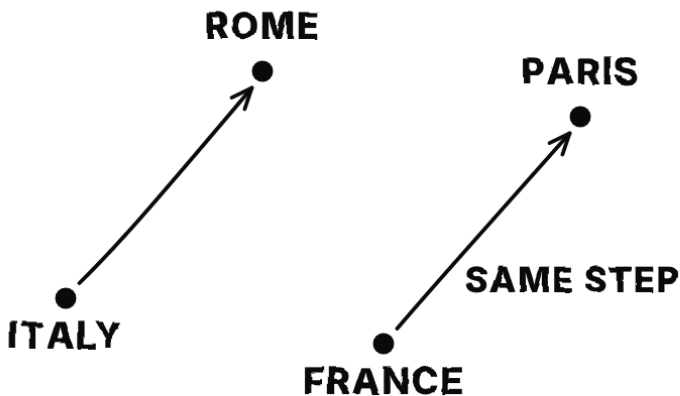


Groups found without answers

The book offers a taste. Can the data be summed up by a few typical examples, so that photos fall into landscapes, dogs, babies, cats and mountain peaks, or shoppers into groups that browse alike? That is clustering. Can a few numbers capture what matters? A ball's flight is well described by its speed, size and mass, and tailors fit clothes with a small set of body measurements; finding such numbers is called subspace estimation, and when the relation is a straight-line one, principal component analysis. Can things be placed as points in space so that relations become directions? Then the step from "Italy" to "Rome" is the same as the step from "France" to "Paris": take Rome,

subtract Italy, add France, and you arrive at Paris. Can we find root causes, such as how house prices, pollution, crime, schooling and wages depend on one another, from the data alone? That is the field of causality and probabilistic graphical models.

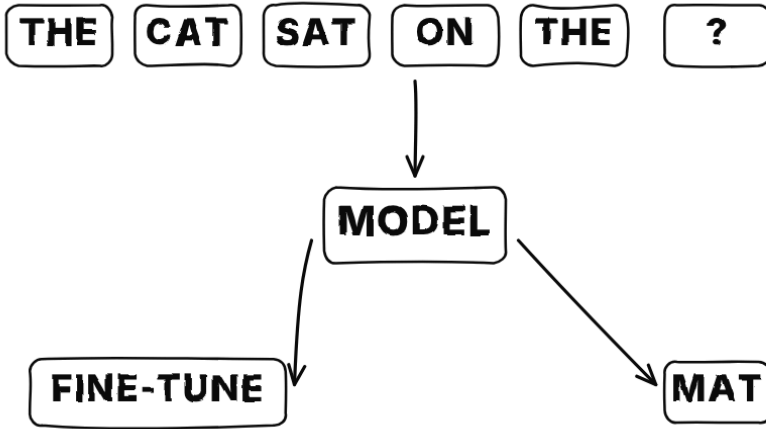
## A MAP OF WORDS



Words as points on a map

A recent and exciting branch is deep generative models. They learn how the data is spread out so well that they can score how likely an example is, or produce brand new examples of their own. The early breakthroughs were variational autoencoders and then generative adversarial networks; more recent are normalizing flows and diffusion models. The image generators that paint a picture from a sentence you type belong to this family.

Last comes self-supervised learning, which uses the unlabelled data itself to make questions with known answers. For text, hide a word at random and train the model to guess it from the words around it. In "the cat sat on the blank", the missing word is almost certainly "mat", and the true answer was in the text all along, so huge amounts of text can be used with no labelling effort. For pictures, a model can be asked where two cut-out patches of one image sat relative to each other, to fill in a covered part from the rest, or to tell whether two altered images came from the same original. What such a model learns, its way of describing the data, is often reused: a little more training, called fine-tuning, adapts it to the particular task one actually cares about. Chatbots began this way, trained first on enormous amounts of text to predict missing or next words, and only then tuned to answer questions.



Hide a word, guess it back

## Remember

Text to speech turns a few letters into a long sound recording, the reverse of speech recognition. Translation is harder still, since input and output differ in length and their pieces can come in a different order, like the German verb at the end.

Unsupervised learning works from data with no labels: clustering, finding a few telling numbers, placing words so that Rome minus Italy plus France lands on Paris, and searching for causes. Generative models learn the data well enough to create new examples, as diffusion models do. Self-supervised learning makes its own questions from the data, such as hidden words, and the result is then fine-tuned for the task in hand.

---

## Questions

1. What makes machine translation harder than speech recognition, according to the lecture?
  - a. Its input is much longer
  - b. The matching pieces of input and output can come in a different order
  - c. It has no labels
  - d. It works only for German
2. Why can self-supervised learning use huge amounts of text with no labelling effort?

- a. Because the hidden word's true answer was already in the text
  - b. Because text has no structure to learn
  - c. Because people label text for free
  - d. Because it ignores the words around the blank
3. Which of these is a clustering question?
- a. Which few numbers describe a ball's flight?
  - b. Does pollution cause higher crime?
  - c. Can these shoppers be split into groups that browse alike?
  - d. What is the next word in this sentence?
4. What is fine-tuning?
- a. Training a model from scratch on labelled data
  - b. Removing the parts of a model that are not needed
  - c. Making a generative model produce sharper pictures
  - d. A little more training that adapts a self-supervised model to the task one cares about

## Answers

- 1. b)** In translation, input and output can have different lengths, and the matching pieces can come in a different order.
- 2. a)** In "the cat sat on the blank", the true answer was in the text all along, so no one has to label anything.
- 3. c)** Summing up the data by a few typical examples, such as groups of shoppers who browse alike, is clustering.
- 4. d)** What a self-supervised model learns is often reused: a little more training, called fine-tuning, adapts it to the particular task.

## Worked examples

**From letters to sound** A text-to-speech system produces sound at 16,000 measurements a second. A sentence takes 2 seconds to say. How many numbers does it produce, and why is text to speech a problem where the output is much longer than the input?

Each second of sound is 16,000 measurements.

Two seconds give 16,000 times 2, which is 32,000.

The sentence went in as a few dozen letters.

So a short input becomes a very long output, the reverse of speech recognition.

**Answer:** 32,000 numbers for a sentence of a few dozen letters

## Practice

**1.** Using the idea that relations become directions, complete: Paris minus France plus Italy gives what? And Rome minus Italy plus France?

**Hint:** Removing a country and adding another moves you from one capital to the other's capital.

**Answer:** Paris minus France plus Italy lands on Rome, and Rome minus Italy plus France lands on Paris: the step from a country to its capital is the same direction in both cases.

**2.** Explain with the book's two bosses the difference between supervised and unsupervised learning.

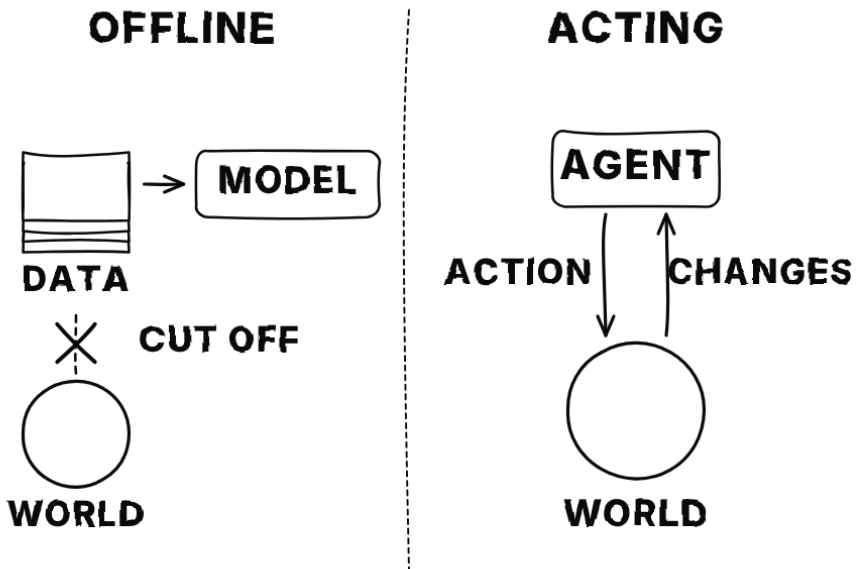
**Hint:** One boss tells you everything; the other tells you nothing.

**Answer:** In supervised learning the boss stands over your shoulder and gives the right answer for every situation, so you only need to copy the pattern. In unsupervised learning the boss just hands over a pile of data

with no answers and asks you to find something useful in it, so you must discover the structure yourself.

# When the machine acts

Watch a small child building a tower of blocks. No one gives them a sheet of right answers. They set one block on another, the tower wobbles, stands or falls, and they try something different. Every move changes what they see next, and the only verdict on all their choices is whether the tower stands at the end.



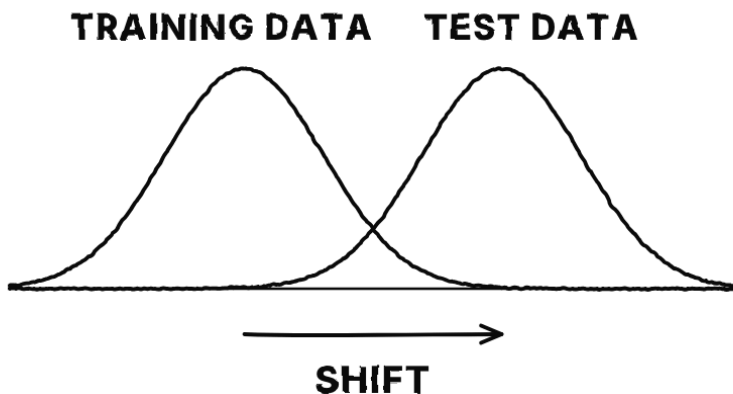
Learning cut off, or acting

So far, the book admits, it has not asked where data comes from, or what happens once a model gives its answer. Supervised and unsupervised learning both work the same simple way: grab a big pile

of data up front, then set the pattern-finding machine running without ever touching the world again. Because all the learning happens after the machine is cut off from the world, this is called offline learning. The book draws supervised learning as a simple picture: the world supplies examples, someone labels them, and the model learns from the finished pile.

Offline learning has its charms, for one can study pattern recognition on its own, without the complications of a world that moves and responds. But it is limiting. Readers who grew up on Asimov's robot novels picture intelligent machines that do not only predict but act in the world. The book wants to think about agents, not just predictive models, and about choosing actions, not just making predictions. A prediction leaves the world as it was; an action changes it. To train an agent, we must account for how its own actions will change what it sees next, and for that, offline learning is the wrong tool.

Once there is an exchange with the world, new questions pile up. Does the world remember what we did before? Does it want to help, like a person reading text slowly and clearly into a speech recogniser? Does it want to beat us? Spammers are the book's example: they keep rewriting their messages to slip past your spam filter, so the filter's world fights back. Does the world change over time, by itself or because of our own tools, so that future data no longer looks like the past? All of these lead to distribution shift, where the data at test time differs from the data at training time. The book's example will be familiar to any student: homework written by the teaching assistants, then an exam written by the lecturer.

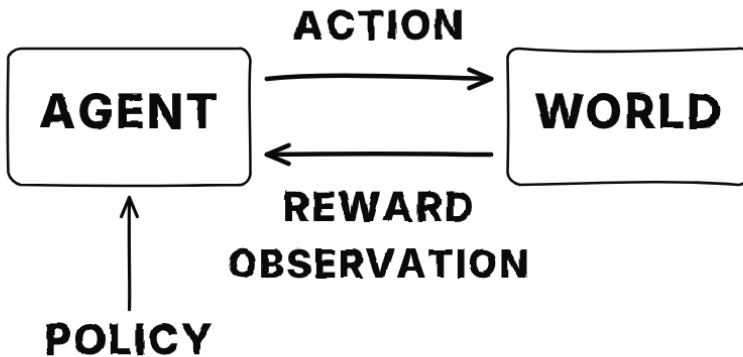


Test data can drift

The main framework for learning to act is reinforcement learning. It is used in robotics, in systems that hold conversations, and in the computer opponents of video games. Combined with deep learning, as deep reinforcement learning, it has made headlines. A program called the deep Q-network learned to beat people at old Atari arcade games while seeing nothing but the screen. And AlphaGo defeated the world champion at the board game Go.

The book draws reinforcement learning as a loop. At each step the agent receives an observation of the world and chooses an action, which is passed back to the world through some mechanism, sometimes called an actuator. The world then gives the agent a reward, and a new observation, and the agent chooses again, step

after step. The agent's behaviour is governed by its policy, simply a rule that maps what it observes to what it does. The goal of reinforcement learning is to find good policies.

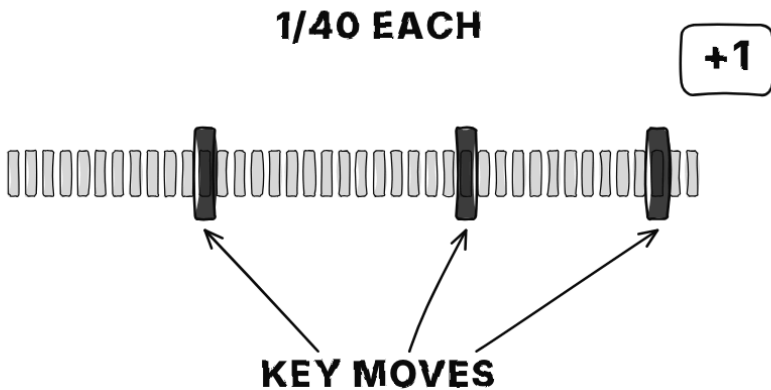


The reinforcement learning loop

The framework is extraordinarily general. Even ordinary classification can be dressed up as reinforcement learning: give the agent one action for each class, and let the world hand back a reward equal to the loss of the original problem. But reinforcement learning also covers problems supervised learning cannot. In supervised learning every training input comes with the correct label. In reinforcement learning the world does not say what the best action was; it hands out only a reward, and often does not even say which action earned it.

Chess makes this vivid. The only real reward arrives at the end: one point for a win, say, and minus one for a loss. A game might last forty moves by each player. If we naively spread a win equally over all forty, each move would get one fortieth of a point, two and a half hundredths, whether it was the brilliant move that won or a careless one that nearly lost. That is plainly wrong, and finding which actions truly deserve the credit or the blame is called the credit assignment problem. People face it too: someone promoted on the eleventh of October owes the promotion to many good choices over the past year, and to be promoted again they must work out which choices those were.

## CREDIT ASSIGNMENT



Who earned the win?

## Remember

Offline learning gathers data first and learns afterwards, cut off from the world; an agent that acts changes the very data it will see. When the world remembers, helps, fights back or changes, the test data can differ from the training data, which is called distribution shift. Reinforcement learning is a loop: observe, act, receive a reward, observe again, with a policy deciding the actions. Deep reinforcement learning has beaten people at Atari games and at Go. Rewards often come late and do not say which actions earned them, and sorting out credit and blame is the credit assignment problem.

---

## Questions

1. Why is offline learning the wrong tool for training an agent?
  - a. Because an agent's actions change the world and so the data it will see next
  - b. Because agents need less data
  - c. Because offline learning cannot recognise patterns
  - d. Because agents only work with text
2. What is the main way reinforcement learning differs from supervised learning in what it is told?
  - a. It is told the best action at every step
  - b. It gets only a reward, often without knowing which action earned it

- c. It gets no feedback at all
  - d. It is given labelled examples in advance
3. Why is spreading a chess win equally over all forty moves a poor way to assign credit?
- a. Because forty moves is too few
  - b. Because the reward of a win is zero
  - c. Because it gives a brilliant winning move and a careless one the same credit
  - d. Because chess has no rewards
4. Spammers rewrite their emails to evade a spam filter. Which question about the world does this answer yes to?
- a. Does the world want to help us?
  - b. Does the world have no memory?
  - c. Is the world fully observed?
  - d. Does the world want to beat us?

## Answers

1. **a)** To train an agent, we must account for how its own actions will change what it sees next, and offline learning cannot.
2. **b)** In reinforcement learning the world does not say what the best action was; it hands out only a reward, often without saying which action earned it.
3. **c)** Spread equally, each move gets one fortieth of a point, whether it was brilliant or careless, which is plainly wrong.
4. **d)** Spammers keep rewriting their messages to slip past the filter: the filter's world fights back.

## Worked examples

**Naively sharing out a win** A chess game earns a reward of 1 for a win. The winner made 40 moves. If the reward were shared equally among all the moves, how much credit would each move get, and why is that a problem?

The only reward, 1, arrives at the end of the game.

Shared equally over 40 moves, each move gets 1 divided by 40.

1 divided by 40 is 0.025.

Every move gets the same 0.025, whether it won the game or nearly lost it, so equal sharing cannot tell good moves from bad: this is the credit assignment problem.

**Answer:** 0.025 per move, the same for good and bad moves

## Practice

**1.** Describe the reinforcement learning loop for a computer opponent in a video game: what does it observe, what are its actions, what might its reward be, and what is its policy?

**Hint:** Observe, act, receive a reward, observe again.

**Answer:** It observes the game screen or game state, its actions are the moves or button presses it can make, and its reward might be points scored or winning the level. Its policy is the rule that picks an action from what it currently observes, and learning aims to make that rule good.

**2.** Explain how classification can be recast as reinforcement learning, as the book describes.

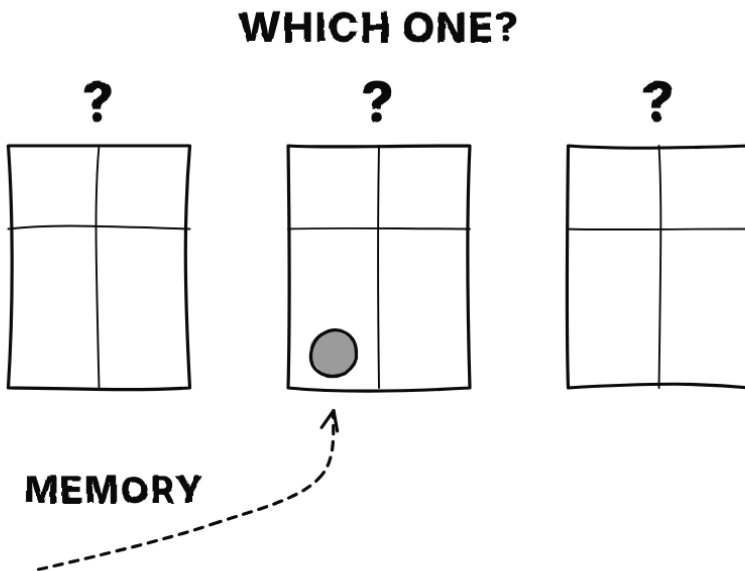
**Hint:** What would the actions and the reward be?

**Answer:** Give the agent one action for each class, so choosing an action means choosing a class, and let the world return a reward equal to the

loss of the original classification problem. The agent then learns to pick the classes that earn the best reward, which is the same as classifying well.

# Blind spots, gambles and old roots

Imagine your robot vacuum has wandered into one of several identical cupboards in your house and got stuck. From inside, every cupboard looks exactly the same: four walls, a door, a shelf. What it sees right now cannot tell it which cupboard it is in. Only by remembering what it saw before it went in can it, or you, work out where it is.

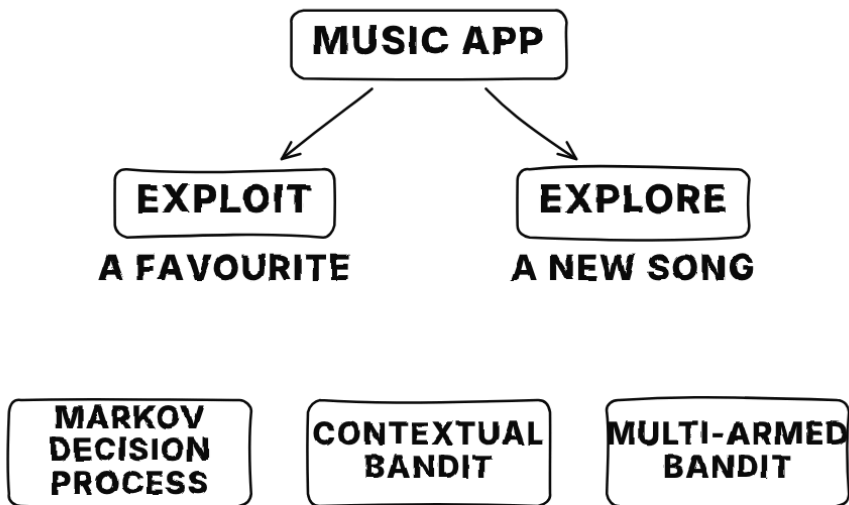


Identical cupboards

This is the problem of partial observability. In reinforcement learning, the agent's current observation may not tell it everything about the

state it is in. To act well, it may need to piece together the present from what it has seen before.

There is another difficulty. At any moment the agent may know one strategy that works reasonably well, while there may be many better ones it has never tried. It must keep choosing between exploiting the best strategy it knows and exploring others, perhaps giving up some reward now in exchange for knowledge. Your music app faces this every day: play a song it is sure you like, or gamble on a new one you might love or might skip.



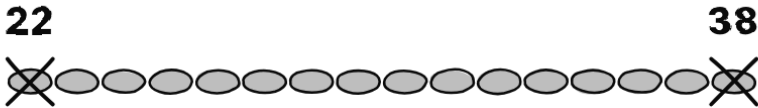
Exploit or explore; simpler cases

Put together, the general problem is very broad: actions change what the agent sees next, rewards are seen only for the actions actually

taken, and the world may be fully or only partly visible. Taking all of that on at once may be asking too much, and many real problems do not have all of it, so researchers study special cases. 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 the agent's earlier actions, it is a contextual bandit problem. And when there is no state at all, only a set of possible actions whose rewards are at first unknown, it is the classic multi-armed bandit problem, named after a row of slot machines, the one-armed bandits.

With this tour of problems done, the book turns to history. Many deep learning methods are recent, but the ideas behind learning from data have been studied for centuries. The wish to analyse data and foresee what comes next lies at the root of much of science and mathematics. Two patterns of chance carry the names of great mathematicians: the Bernoulli distribution, after Jacob Bernoulli, who died in 1705, and the Gaussian distribution, after Carl Friedrich Gauss. Gauss also devised the least squares method, which fits a rule to measurements by making the squared misses as small as possible, and which is still used from insurance to medical diagnosis. Such tools strengthened the experimental sciences; Ohm's law, linking the current through a resistor to the voltage across it, is perfectly described by a straight-line model.

## A TYPICAL FOOT



$$480 \div 16 = 30$$

$$420 \div 14 = 30$$

## TRIMMED MEAN

Sixteen feet, a trimmed mean

Even earlier, people had a good feel for estimates. A sixteenth-century geometry book by Jacob Köbel shows how to find the length of a typical foot. As people came out of church, sixteen grown men lined up heel to toe, the total length of their feet was measured, and the total was divided by sixteen. Say the sixteen feet came to four hundred and eighty centimetres: divided by sixteen, that gives thirty centimetres. The method was later improved to cope with odd feet: the man with the shortest feet and the man with the longest were sent away, and only the other fourteen were averaged. If those two measured twenty-two and thirty-eight centimetres, the remaining fourteen total four hundred and twenty, and four hundred and twenty divided by fourteen is again thirty. This is one of the earliest examples

of a trimmed mean, an average that throws out the extremes so that one odd value cannot spoil it.

Statistics really took off once data was collected in quantity. Ronald Fisher, one of its pioneers, shaped both its theory and its use in genetics, and some of his methods and ideas are still at the base of modern statistics. Even his data lasted: the measurements of iris flowers he published in 1936 are still sometimes used to try out learning methods. Fisher was also a promoter of eugenics, a reminder that the morally dubious use of data science has as long a history as its useful one.

Other influences came from Claude Shannon's theory of information and Alan Turing's theory of computation. In a famous paper Turing asked, "Can machines think?", and proposed what is now called the Turing test: a machine counts as intelligent if a judge, exchanging only written messages, finds it hard to tell its replies from a person's. Chatting with a modern chatbot, many people have found that test less hypothetical than it once seemed. Neuroscience and psychology added more, since humans plainly behave intelligently and one may hope to understand and copy how. Donald Hebb proposed in his book on the organisation of behaviour that nerve cells learn by positive reinforcement, an idea now called the Hebbian learning rule. It inspired Rosenblatt's perceptron and laid the ground for the gradient methods used in deep learning today: strengthen what leads to good behaviour, weaken what leads to bad, and so arrive at good settings for the knobs. That biological inspiration is where neural networks get their name; for more than a century, since models by Bain and Sherrington, researchers have tried to build circuits that resemble networks of nerve cells.

## Remember

An agent may see only part of its situation, like a robot stuck in one of many identical cupboards, and must use what it saw before. It must also balance exploiting the best known strategy against exploring others that might be better. Special cases of reinforcement learning are the Markov decision process, the contextual bandit and the multi-armed bandit. Learning from data is centuries old, from Bernoulli and Gauss's least squares to sixteen men's feet averaged outside a church, later as a trimmed mean. Statistics, information theory, the theory of computation and the study of the brain all fed machine learning, and neural networks take their name from nerve cells.

---

## Questions

1. Why can a robot stuck in one of several identical cupboards not tell where it is from its current view?
  - a. Because its sensors are broken
  - b. Because every cupboard looks the same, so the current observation does not reveal its whole situation
  - c. Because it has no policy
  - d. Because it is exploring
2. Which special case of reinforcement learning has no state at all, only actions with unknown rewards?
  - a. The Markov decision process

- b. The contextual bandit problem
  - c. The multi-armed bandit problem
  - d. Offline learning
3. Why did later foot-measurers send away the men with the shortest and longest feet?
- a. So that one odd value could not spoil the average
  - b. To make the foot longer
  - c. Because sixteen men were too many to measure
  - d. To follow Gauss's least squares
4. What principle from Hebb's idea still guides how deep learning tunes its knobs?
- a. Measure every nerve cell directly
  - b. Copy the brain's wiring exactly
  - c. Keep all knobs fixed after the first try
  - d. Strengthen what leads to good behaviour and weaken what leads to bad

## Answers

1. **b)** This is partial observability: the agent's current observation may not tell it everything about the state it is in.
2. **c)** When there is no state, only a set of actions whose rewards are at first unknown, it is the classic multi-armed bandit problem.
3. **a)** A trimmed mean throws out the extremes so that one odd value cannot spoil it.
4. **d)** Hebb's idea laid the ground for today's gradient methods: strengthen what leads to good behaviour, weaken what leads to bad.

## Worked examples

**Köbel's foot, and its trimmed version** Sixteen men's feet, lined up heel to toe, measure 480 centimetres in total. The shortest foot is 22 centimetres and the longest 38 centimetres. Find the plain average, and the trimmed mean after sending away those two men.

The plain average divides the total by the number of men: 480 divided by 16 is 30 centimetres.

For the trimmed mean, remove the shortest and longest feet: 480 minus 22 minus 38 is 420 centimetres.

Fourteen men remain, so divide by 14.

420 divided by 14 is 30 centimetres; here the two extremes balanced out, but a single very odd foot would no longer pull the answer.

**Answer:** 30 centimetres both ways

## Practice

**1.** Ten men's feet total 300 centimetres, but one of them, measuring 50 centimetres, wore huge boots by mistake, and the shortest foot is 25 centimetres. Compare the plain average with the trimmed mean that drops the shortest and longest.

**Hint:** For the trimmed mean, take away both extremes and divide by eight.

**Answer:** The plain average is 300 divided by 10, which is 30 centimetres. Dropping 25 and 50 leaves 225 centimetres over 8 men, about 28.1 centimetres. The boot pulled the plain average up; the trimmed mean is not fooled by it.

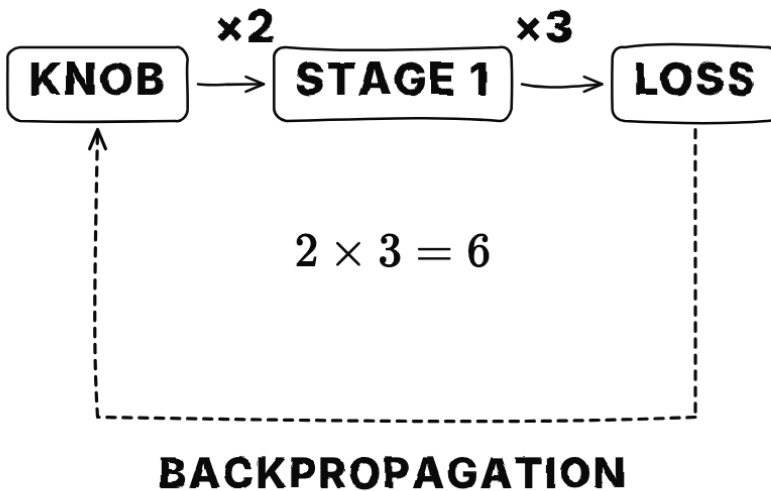
**2.** Explain with your own example the choice between exploring and exploiting.

**Hint:** Think of a restaurant, a route to work, or a playlist.

**Answer:** Going to your favourite restaurant is exploiting: you know you will eat well. Trying a new one is exploring: it may be worse, but it might be better, and you only find out by trying. A learner that never explores can get stuck with a merely good choice; one that always explores never enjoys what it has learned.

# From a dead end to an explosion

Think of the first computer game you ever played, and then of a game on a phone today. The jump in what a small machine can draw, and how quickly, is enormous. Here is the twist: the chips built to make games look good turned out to be exactly what neural networks had been waiting for. To see why, we need the two ideas at the heart of these networks, and the story of why they once stalled.

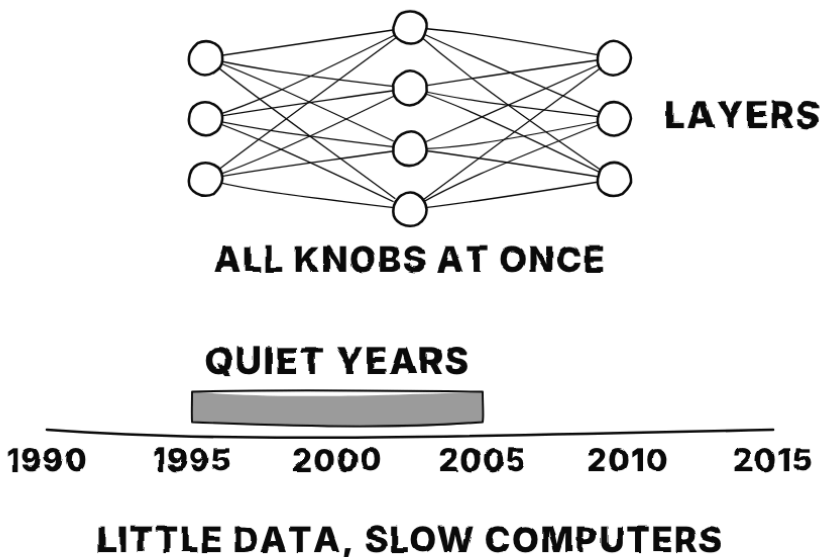


The chain rule, in numbers

Nearly every neural network today rests on two principles. The first is layers: stages of processing that alternate between simple straight-line

combinations of numbers and simple bends, so that each stage reshapes what it receives a little before passing it on. The second is a way of adjusting the knobs of the whole network at once, using the chain rule of calculus, a method called backpropagation. The chain rule says, in words, that if one stage feeds the next, the effect of an early knob on the final loss is the product of the effects along the way. For instance, if turning a knob a little raises the first stage's output by two units per unit of turn, and each unit of that output raises the loss by three, then the knob raises the loss by two times three, six units per unit of turn. Backpropagation applies this from the end of the network back to the start, and so learns for every knob, in one sweep, which way to turn it.

After early rapid progress, neural network research languished from about 1995 to 2005, for two reasons. Training a network takes a great deal of computing, and while memory was plentiful by the end of the century, computing power was scarce. And datasets were small: Fisher's iris flowers of 1936 were still a popular test, and a collection of sixty thousand handwritten digits was thought huge. With so little data and computing, other strong statistical tools did better in practice, among them kernel methods, decision trees and graphical models. Unlike neural networks, they did not need weeks to train, and they came with predictable results and solid mathematical guarantees.



Layers, and the quiet years

What changed was data. The web, companies serving hundreds of millions of people online, cheap and good sensors everywhere, and storage that kept getting cheaper together produced data on a vast scale. Computing kept getting cheaper too, and above all the graphics chips designed for computer games, GPUs, transformed what deep learning could compute. Suddenly algorithms and models that had seemed hopeless were within reach.

The book sets this out in a table, decade by decade. In 1970 a typical dataset was the iris flowers, about a hundred examples; a computer had a kilobyte of memory and did about a hundred thousand calculations a second. By 1990 the typical dataset was ten thousand scanned characters, with ten megabytes of memory and ten million

calculations a second. By 2020 a dataset might be a whole social network, about a trillion items; memory had reached a hundred gigabytes, and a single machine packed with graphics chips did about a million billion calculations a second. Compare the growth over those fifty years. Data grew from a hundred to a trillion, ten billion times; memory grew from a kilobyte to a hundred gigabytes, a hundred million times. So memory fell a hundredfold behind the data. Computing power, the book says, outpaced even the data. That pushed the best methods towards models that are thrifty with memory but can spend lavish computing on tuning their knobs, and the sweet spot moved from linear models and kernel methods to deep neural networks. It also explains why many of deep learning's mainstays, such as stacked perceptrons, convolutional networks, long short-term memory networks and Q-learning, were essentially rediscovered in the last decade, after lying dormant for years.

# GROWTH SINCE 1970

**DATA ×10 BILLION**



**COMPUTING: EVEN MORE**



**MEMORY ×100 MILLION**

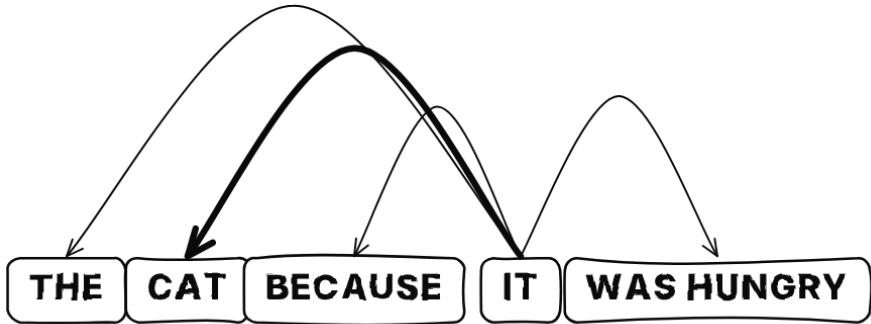


**EACH TICK: ×10**

What grew, and how much

Yet the progress was more than old recipes on faster machines. The book compares it to the Cambrian explosion, the burst in which animal life suddenly diversified, and lists a few of the new ideas. Dropout fights overfitting by injecting random noise throughout the network during training. Attention solved an old problem: how to give a system more memory and complexity without adding more knobs. Instead of squeezing a whole sentence into a fixed-size summary before translating it, a model with attention keeps a kind of learnable pointer back to the part it needs, and long sentences became far more accurate.

# ATTENTION



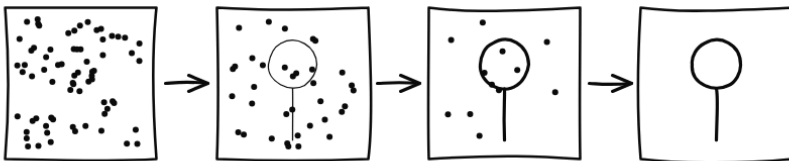
## TRANSFORMER: ATTENTION ONLY

Attention looks back

The Transformer, built entirely on attention, keeps getting better as the data, the model and the computing grow. It has succeeded in language, vision, speech, reinforcement learning and more; a single Transformer trained on text, images, robot joint movements and button presses can play Atari, caption images, chat and control a robot. Language models predict text from other text, and as they were scaled up they gained more and more abilities. The chatbot you type to is the book's own example: ChatGPT, a language model aligned with what people want, which can help debug computer code or write creatively in conversation. Multi-stage designs, such as memory networks, let a network change its internal state step by step, carrying out a chain of reasoning much as a processor updates its memory during a computation.

Generative models made leaps as well. Generative adversarial networks set a generator, which makes fake examples, against a discriminator, which tries to tell fake from real; the generator's knobs are tuned until the discriminator cannot tell them apart. The results include galloping zebras made from horses and convincing fake celebrity faces, and even amateur doodlers can turn a rough sketch into a photograph-like picture. More recently, diffusion models learn to reverse the gradual adding of noise, building a picture step by step out of static, and they power text-to-image systems such as DALL-E 2 and Imagen. An image generator that paints whatever you describe is this idea at work.

## DIFFUSION



## REMOVE NOISE

Picture out of noise

Engineering mattered too. The workhorse optimiser, stochastic gradient descent, works on small batches of examples at a time, but small batches leave graphics chips underused. Spread training over one thousand and twenty-four chips, each taking thirty-two images, and each step handles thirty-two thousand seven hundred and sixty-eight images, about thirty-two thousand. Later work pushed this to sixty-four thousand and brought the training of a standard image-recognition network, ResNet-50 on the ImageNet collection, down to under seven minutes, where it had first taken days. Parallel computing also helped reinforcement learning reach superhuman play at Go, Atari games and StarCraft, wherever a simulator can provide plenty of states, actions and rewards to learn from.

Finally, frameworks, free toolkits for building networks, spread the ideas. The first generation, Caffe, Torch and Theano, carried many landmark papers. Then came TensorFlow, often used through Keras, along with CNTK, Caffe 2 and MXNet. The third generation is imperative: you write a model the way you write ordinary numerical Python, an idea sparked by Chainer and taken up by PyTorch, MXNet's Gluon and JAX.

## Remember

Neural networks alternate straight-line and bending layers, and backpropagation uses the chain rule, multiplying effects along the way, to adjust all the knobs at once. They stalled from about 1995 to 2005 because computing was scarce and datasets small, so kernel methods and decision trees won. From 1970 to 2020 data grew ten billion times while memory grew a hundred million times, and computing grew faster still, which favoured deep networks. Dropout, attention, the Transformer, language models, adversarial networks and diffusion models drove a burst of progress. Training across thousands of chips, and frameworks such as PyTorch, made it fast and widely available.

---

## Questions

1. What does backpropagation use to find, for every knob at once, which way to turn it?
  - a. Random trials of each knob
  - b. The chain rule, multiplying the effects along the stages from the end back to the start
  - c. A table of the best settings found before
  - d. Human judgement of each layer
2. Why did kernel methods and decision trees beat neural networks around 1995 to 2005?
  - a. Because neural networks had been proved wrong

- b. Because memory was too scarce for them
  - c. Because computing was scarce and datasets small, and these methods trained faster with solid guarantees
  - d. Because they used graphics chips
3. From 1970 to 2020, data grew far more than memory. What did that push the best methods towards?
- a. Models thrifty with memory that spend lots of computing tuning their knobs
  - b. Models that store every example
  - c. Models with no knobs at all
  - d. Smaller datasets
4. What problem did attention solve in translation?
- a. It made training data unnecessary
  - b. It removed noise from images
  - c. It replaced the graphics chip
  - d. It let the model point back to the part of the sentence it needs instead of squeezing the whole sentence into a fixed summary

## Answers

- 1. b)** Backpropagation applies the chain rule from the end of the network back to the start and so learns, for every knob in one sweep, which way to turn it.
- 2. c)** With so little data and computing, other tools did better; they did not need weeks to train and gave predictable results with guarantees.
- 3. a)** That pushed the best methods towards models thrifty with memory but able to spend lavish computing on tuning their knobs.

**4. d)** Instead of squeezing a whole sentence into a fixed-size summary, a model with attention keeps a learnable pointer back to the part it needs.

## Worked examples

**One huge step across many chips** Training is spread over 1,024 graphics chips, and each chip takes a batch of 32 images at every step. How many images does one step handle altogether?

Each chip handles 32 images per step.

All 1,024 chips work at the same time, so multiply.

1,024 times 32 is 32,768.

That is about 32,000 images per step, which keeps every chip busy.

**Answer:** 32,768 images, about 32,000

## Practice

**1.** Using the chain rule as the lecture tells it: turning a knob raises a layer's output by 4 units per unit of turn, and each unit of that output lowers the loss by 0.5. What is the knob's effect on the loss, and which way should gradient descent turn it?

**Hint:** Multiply the effects along the way, keeping track of raises and lowers.

**Answer:** 4 times minus 0.5 gives minus 2: each unit of turn lowers the loss by 2. Since turning the knob up lowers the loss, gradient descent turns it up.

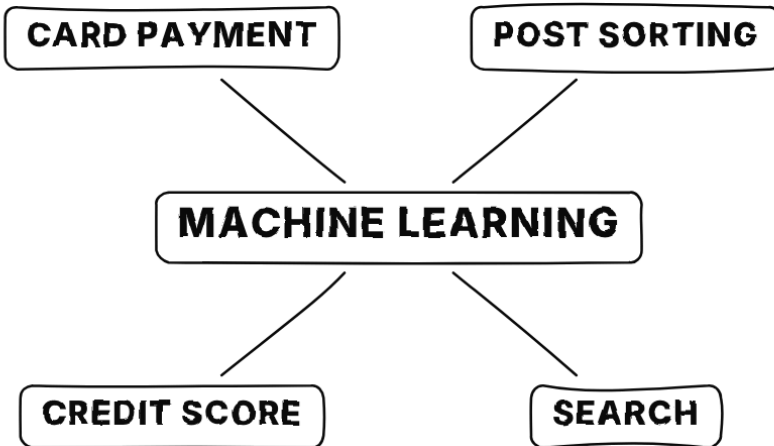
**2.** Memory grew from 1 kilobyte in 1970 to 100 gigabytes in 2020, while datasets grew from about 100 examples to about a trillion. By what factor did each grow, and what does this say about storing whole datasets in memory?

**Hint:** A gigabyte is a million kilobytes.

**Answer:** 100 gigabytes is 100 million kilobytes, so memory grew 100 million times; data grew from 100 to a trillion, 10 billion times. Data grew a hundred times faster, so whole datasets fit in memory less and less, and methods had to become thrifty with memory.

# Quiet successes and real worries

Pay for something online with a card and, in the instant before the payment goes through, a program has already judged whether it looks like fraud. Post a letter, and a machine has very likely read the handwritten address. Neither feels like artificial intelligence. Yet both are machine learning, and both have been running for years, mostly out of sight.



Machine learning, out of sight

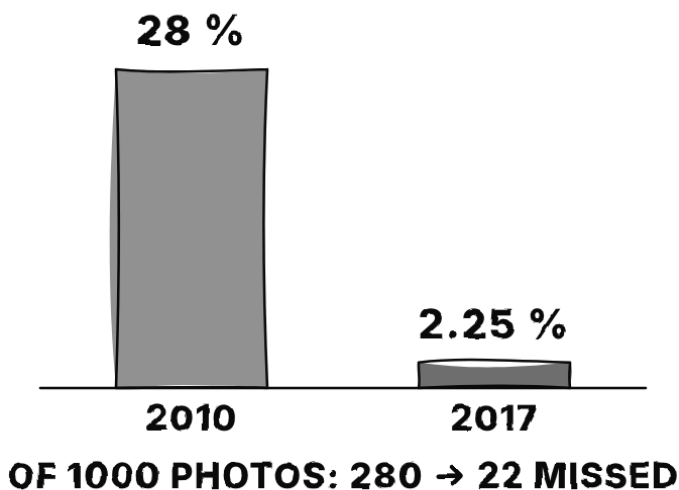
One reason the field has moved so fast is a division of labour. Some researchers build better tools, others build better models, and each

side benefits from the other. The book gives a telling example. In 2014, training a linear logistic regression, one of the simplest classifiers, was still a homework problem worth setting to new doctoral students in machine learning at Carnegie Mellon University. Today it takes fewer than ten lines of code, well within reach of any programmer.

Some successes are old. Mail-sorting machines that read handwritten addresses have been in use since the 1990s, and that is where the famous collection of handwritten digits came from. Machines read cheques for bank deposits and score whether loan applicants are creditworthy. Payments are checked for fraud automatically, forming the backbone of payment systems such as PayPal, Stripe, Visa and Mastercard. Chess programs have been competitive for decades. And machine learning feeds search, recommendation, personalisation and ranking across the internet. The order of the videos in your recommendation feed is one of these quiet results. In short, machine learning is everywhere, though often hidden.

Artificial intelligence has stepped into the limelight only recently, because it began to solve problems once thought intractable, problems that ordinary consumers meet directly, and many of these advances are owed to deep learning. Voice assistants such as Siri, Alexa and Google's assistant answer spoken requests with reasonable accuracy, from switching on a light to booking a barber's appointment, and that is probably the most noticeable sign of artificial intelligence in daily life. Their key ingredient, speech recognition, has improved to the point of matching people in certain uses.

## TOP-5 ERROR



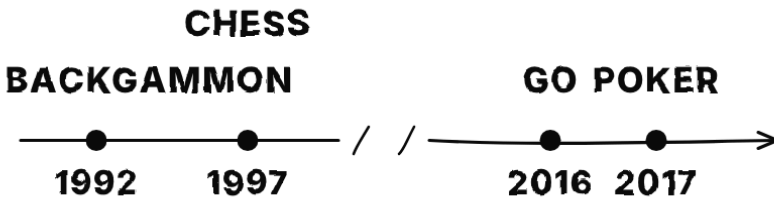
ImageNet errors, 2010 to 2017

Object recognition has come just as far. On the ImageNet test, a model counts as wrong on a photo if the right answer is not among its top five guesses, and the share of such misses is its top-five error rate. In 2010 a team from NEC Labs and the University of Illinois reached a top-five error rate of twenty-eight percent. By 2017 it had fallen to two point two five percent. Out of a thousand photos, that is a drop from two hundred and eighty misses to about twenty-two, more than twelve times fewer. There have been similarly striking results in identifying birds by their song and in diagnosing skin cancer.

Games were once a measuring stick for human ability, and machines have climbed it. TD-Gammon learned backgammon by a form of reinforcement learning. Chess has far more positions and moves, and

Deep Blue beat Garry Kasparov with massive parallel computing, special hardware and an efficient search through the tree of possible games. Go is harder still, with an even vaster number of positions, and AlphaGo reached human level in 2015 by combining deep learning with a way of sampling likely continuations of the game, called Monte Carlo tree sampling. Poker adds a different difficulty: the state is large and only partly visible, since you cannot see the other players' cards, yet a program called Libratus surpassed human players with carefully structured strategies.

## **GAMES WON BY MACHINES**



Games machines learned to win

Self-driving vehicles are another sign of progress. Full autonomy is not yet within reach, but companies such as Tesla, NVIDIA and Waymo ship products that drive themselves part of the time. Full

autonomy is hard because good driving requires perceiving, reasoning and following rules; at present deep learning mostly handles the perceiving, and engineers tune the rest by hand. Beyond these, robotics, logistics, computational biology, particle physics and astronomy owe some of their most impressive recent advances at least partly to machine learning, which is becoming an everyday tool for engineers and scientists.

Popular articles often raise fears of an artificial intelligence apocalypse, the so-called singularity, in which machines become conscious and decide our fate without their programmers. To some extent artificial intelligence already affects people's lives directly: creditworthiness is judged automatically, autopilots do most of the navigating of many vehicles, and decisions on granting bail use statistical data. More frivolously, we can ask a voice assistant to switch on the coffee machine.

## HEADLINES

**ROBOT UPRISING**

**NO SIGN OF IT**

## REAL CONCERNS

**AUTOMATIC CREDIT**

**JOBS AUTOMATED**

**BIASED MODELS**

Headline fears, real concerns

The authors are reassuring about the apocalypse. First, today's systems are engineered, trained and deployed for specific goals; however general their behaviour may look, underneath is a combination of rules, rules of thumb and statistical models. Second, there are simply no tools at present for an artificial general intelligence that could improve itself, reason about itself and redesign its own structure while solving general tasks.

The pressing concern, they say, is how artificial intelligence is used in daily life. Many routine tasks now done by people can and will be automated. Farm robots will probably lower costs for organic farmers, but they will also automate the harvest, and since such manual work provides much of the employment in many countries, this stage of the

industrial revolution could have deep consequences for large parts of society. And statistical models applied without care can be biased by race, gender or age, which raises real concerns about fairness when they drive decisions with serious consequences. A loan or job decision made by a model trained on yesterday's unfair choices can repeat them automatically, as we saw with the hiring example. These algorithms must be used with care, and the authors judge this far more urgent than any malevolent superintelligence.

### **Remember**

Better tools turned what was a doctoral homework problem in 2014 into fewer than ten lines of code. Machine learning has long worked quietly in mail sorting, cheque reading, fraud detection and online ranking. Deep learning brought visible leaps: voice assistants, speech recognition, image recognition with errors cut from twenty-eight percent to about two, and victories at backgammon, chess, Go and poker. Self-driving cars and many sciences are advancing with it, though full autonomy is not yet here. The real worries are not a conscious machine but lost jobs and biased automated decisions, which demand careful use.

---

## **Questions**

1. What does a top-five error rate of twenty-eight percent mean?
  - a. The model's first guess is wrong for 28 photos in 100

- b. For 28 photos in 100, the right answer is not among the model's five best guesses
  - c. The model takes five tries to reach 28 percent accuracy
  - d. 28 percent of the photos have five objects in them
2. Why is full self-driving still out of reach, according to the lecture?
- a. Because driving needs perceiving, reasoning and following rules, and deep learning mostly handles only the perceiving
  - b. Because cars cannot carry graphics chips
  - c. Because there are no data about roads
  - d. Because laws forbid it everywhere
3. Why are the authors not worried about a conscious, scheming AI today?
- a. Because AI systems are too slow
  - b. Because AI is banned from important decisions
  - c. Because today's systems are built for specific goals and there are no tools for a self-improving general intelligence
  - d. Because AI cannot play games
4. Which worry do the authors consider most pressing?
- a. The singularity
  - b. Computers running out of memory
  - c. Robots playing chess too well
  - d. Automation of routine jobs and biased decisions by carelessly used models

## **Answers**

1. **b)** A model counts as wrong on a photo if the right answer is not among its top five guesses, and the share of such misses is the top-five error rate.
2. **a)** Good driving requires perceiving, reasoning and following rules; deep learning mostly handles the perceiving, and engineers tune the rest by hand.
3. **c)** Systems are engineered for specific goals, and there are simply no tools at present for an artificial general intelligence that could improve and redesign itself.
4. **d)** The pressing concern is how AI is used in daily life: routine jobs automated, and models applied without care biased by race, gender or age.

## Worked examples

**How far image recognition improved** On the ImageNet test, the top-five error rate was 28 percent in 2010 and 2.25 percent in 2017. How many misses is that out of 1,000 photos in each year, and roughly how many times fewer?

28 percent of 1,000 photos is 280 misses.

2.25 percent of 1,000 photos is 22.5, about 22 misses.

Divide the old count by the new: 280 divided by 22.5 is about 12.4.

So the misses fell to about one twelfth.

**Answer:** 280 misses against about 22; more than twelve times fewer

## Practice

1. A model has a top-five error rate of 5 percent on 2,000 test photos. On how many photos is the right answer missing from its top five guesses?

**Hint:** Take 5 percent of 2,000.

**Answer:** 5 percent of 2,000 is 100 photos.

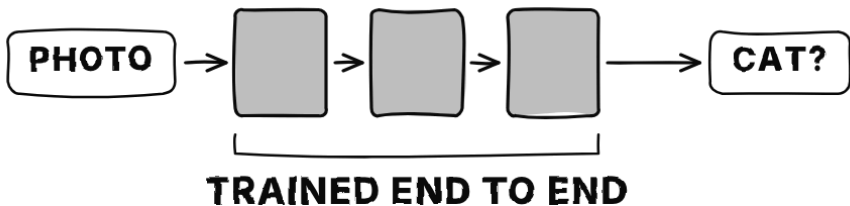
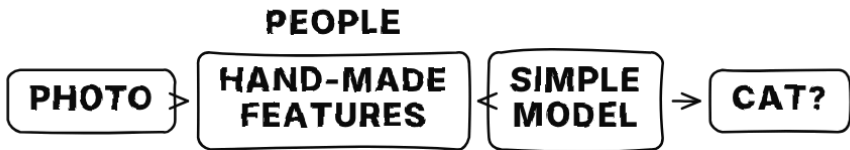
**2.** Explain how better tools changed who can use machine learning, using the book's example of logistic regression.

**Hint:** Compare 2014 with today.

**Answer:** In 2014, training a logistic regression was a homework problem worthy of new doctoral students; today, thanks to tools built by systems researchers, it takes fewer than ten lines of code, so any programmer can do it. The division of labour between tool builders and model builders opened the field to far more people.

# What makes it deep

Imagine teaching a computer to spot cats the way it was done fifteen years ago. First, clever people write a program that finds edges, corners and little textured patches in a photo. Its findings, a list of numbers, go to a simple learning model that only has to decide "cat" or "not cat". If the hand-made first step overlooks something that matters, nothing the model learns afterwards can bring it back. Deep learning removes that split between a hand-made front and a learned back, and that, in the end, is what this whole introduction has been leading to.



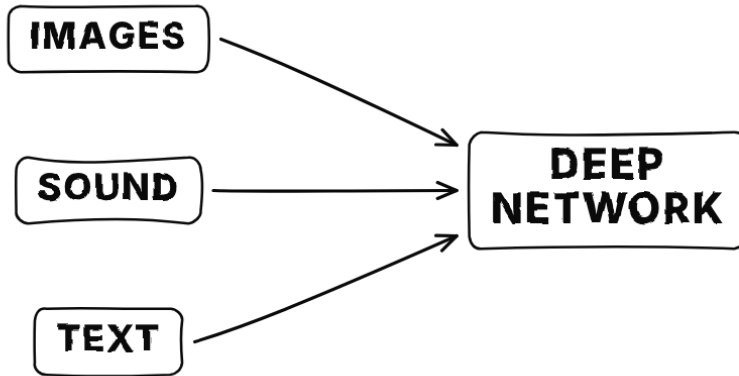
Hand-made front, or end to end

Deep learning is the part of machine learning concerned with models built from neural networks with many layers. It is deep in exactly this sense: its models learn many layers of transformations, each reshaping what it receives and passing it on. That may sound like a narrow definition, yet it has produced a dizzying range of models, techniques, problem types and uses. In a way, all machine learning has several layers of computation, the first being some processing of the features. What sets deep learning apart is that the operations at each of its many layers of representation are learned jointly, all together, from the data.

That is why deep learning excels exactly where traditional methods faltered: learning from raw sound, from the raw pixel values of images, or translating sentences of any length into another language. These many-layered models can handle low-level perceptual data in a way earlier tools could not. The most important thing deep learning methods share is end-to-end training: instead of assembling a system from parts that are each tuned on their own, you build the whole system and tune it all together.

Computer vision shows the change. For years, scientists kept feature engineering, the hand design of ways to turn images into numbers, apart from building the learning model. Two hand-designed methods, the Canny edge detector and Lowe's SIFT feature extractor, reigned for more than a decade as the standard ways of turning an image into a feature vector. In those days, the crucial part of applying machine learning was inventing clever hand-made transformations that a shallow model could cope with. But human ingenuity can only go so far against an algorithm that consistently evaluates millions of choices automatically. When deep learning took over, the hand-made feature

extractors were replaced by filters tuned automatically, and accuracy went up.

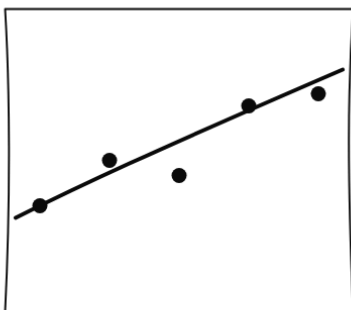


One kind of network for every field

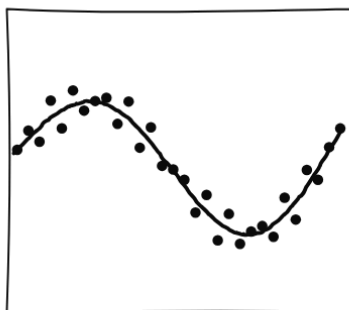
So deep learning replaces not only the shallow model at the end of the old pipeline but also the laborious feature engineering before it. And because much of that preprocessing was specific to one field, deep learning has removed many of the boundaries between computer vision, speech recognition, language processing, medical informatics and other areas, offering one shared set of tools for very different problems. That is why the voice assistant that hears you and the chatbot that writes back to you can be built from the same family of models.

There is a further shift, from parametric models to fully nonparametric ones. When data is scarce, a useful model has to rest on simplifying assumptions about reality, a few parameters describing a fixed shape. When data is abundant, such assumptions can give way to models that follow the data more freely. The book compares this to physics in the middle of the last century: once computers arrived, physicists could simulate the equations of how electrons behave numerically, instead of solving simplified approximations by hand. The models became far more accurate, though often harder to interpret.

**FEW: ASSUME**



**MANY: FOLLOW**



Assume less when data is plenty

Deep learning also differs from earlier work in its attitude. It accepts solutions that are good rather than provably best, tackles optimisation

problems with many hills and valleys, and is willing to try things before proving them. This empiricism, together with a rapid influx of talent, has brought fast progress in practical methods, though sometimes at the cost of reinventing tools that had existed for decades. And the community prides itself on sharing tools across universities and companies, releasing libraries, models and trained networks freely; in the same spirit, the book's own notebooks are free to use and share.

The book's summary draws the threads together. Machine learning studies how computer systems use experience, usually data, to get better at specific tasks, combining ideas from statistics, data mining and optimisation, and it is often the means of building artificial intelligence. Representation learning, one kind of machine learning, is about finding automatically the right way to represent data, and deep learning is representation learning in many levels, through many learned layers. Its recent progress was set off by abundant data from cheap sensors and internet-scale applications, by great advances in computing, mostly through graphics chips, and by efficient frameworks that make it much easier to design and optimise whole systems. That brings the introduction to a close: from programs that follow fixed rules to systems that learn their own layers of description from data.

## Remember

Deep learning is machine learning with many-layered neural networks, in which every layer of representation is learned jointly from data. Its hallmark is end-to-end training: the whole system is tuned together instead of in separately tuned parts. It replaced both the shallow models and the hand-made feature engineering, such as edge detectors, and joined vision, speech and language under one set of tools. With abundant data it can drop simplifying assumptions, and it favours trying things in practice over proving them first. Its rise rests on plentiful data, fast graphics chips and free, efficient frameworks.

---

## Questions

1. Many machine learning systems have several stages. What makes deep learning different?
  - a. Its stages run on graphics chips
  - b. The operations at all its many layers are learned jointly from the data
  - c. It has exactly three layers
  - d. It needs no data preparation of any kind
2. Why did automatically tuned filters replace hand-designed extractors such as SIFT?
  - a. Because hand-designed extractors were too slow to run

- b. Because an algorithm that evaluates millions of choices can outdo human ingenuity, and accuracy went up
  - c. Because SIFT stopped working on new cameras
  - d. Because filters need no training
3. When can a model drop simplifying assumptions and follow the data more freely?
- a. When data is scarce
  - b. When the model is small
  - c. When data is abundant
  - d. When the problem is about physics
4. What did the move to computer simulation in physics gain and lose, in the book's comparison?
- a. It gained speed and lost accuracy
  - b. It gained accuracy but often lost ease of interpretation
  - c. It gained interpretation but lost accuracy
  - d. It lost both accuracy and interpretation

## Answers

- 1. b)** What sets deep learning apart is that the operations at each of its many layers of representation are learned jointly, all together, from the data.
- 2. b)** Human ingenuity can only go so far against an algorithm that evaluates millions of choices automatically; the tuned filters gave better accuracy.
- 3. c)** When data is scarce, models rest on simplifying assumptions; when data is abundant, those assumptions can give way to models that follow the data.

4. b) The models became far more accurate, though often harder to interpret.

## Practice

1. Explain in your own words what end-to-end training is and why it can beat a pipeline of separately tuned parts.

**Hint:** Think of the cat detector with a hand-made edge finder in front.

**Answer:** End-to-end training builds the whole system and tunes all of it together on the data, instead of tuning each part on its own. In a pipeline, a hand-made first step can throw away something that matters, and no later learning can recover it; when every layer is learned together, the early layers learn to keep exactly what the later ones need.

2. Summarise in three sentences the three conditions the book credits for deep learning's recent progress.

**Hint:** Data, hardware and software.

**Answer:** Abundant data from cheap sensors and internet-scale applications gave models enough examples. Advances in computing, mostly graphics chips, made training large networks feasible. Efficient frameworks made it much easier to design, implement and optimise whole systems.