1 Introduction

The term *artificial intelligence* (AI) is used to describe machines that demonstrate intelligent behavior. Often AI refers to computers or robots that carry out tasks or solve problems in a way that resembles intelligent human behavior. But an AI does not necessarily mimic the human cognitive system — it could be constructed and operate in a completely different manner, as long as it is in some sense *intelligent*.

Even though computers easily outperform humans on many tasks, not all such tasks require *intelligence*. For example, most of us have probably memorized the value of π to at least three decimal points of precision (it is about 3.14 as you may recall), but the world record holder managed to memorize more then 100 000 decimals. Even though this is a impressive and remarkable achievement for a human being, it can be debated how much intelligence it requires. A computer can easily compute and memorize an almost infinite number of decimals, limited only by the size of its storage capacity. Simply outperforming humans on a difficult task is not enough to label the computer program as intelligent.

1.1 Intelligence

To approach the question of what artificial intelligence is, perhaps we should start by defining more precisely what we mean by the word *intelligence* itself. This might not be so easy, since intelligence is a term which can mean different things to different people.

1.1.1 *Some requisites for defining intelligence*

Some of the aspects of intelligence that we might identify could include the ability to interact, learn from experience, and achieve goals:

Interact It is difficult to imagine an intelligent creature without the ability to interact with the physical world. It seems reasonable to say that a requisite for intelligence is to percieve the world and take actions which affect the world.

> There is no such thing as a disembodied mind. The mind is implanted in the brain, and the brain is implanted in the body.
> —Antonio Damasio

According to the theory of *embodied cognition* human intelligence is strongly influenced by our bodies, and shaped by our abilities to perceive and act.

Learn from experience It is hard to imagine a *static* intelligent creature: The ability to learn and adapt one's behavior to the environment is also a reasonable requisite for defining intelligent behavior. Related to learning are concepts such as forming mental representations of percepts and mental models of the world, as well as remembering and being able to generalize from past events.

Achieve goals Finally, a reasonable requisite of intelligence is to have certain goals and act in a way that helps achieve them. It is difficult to imagine an intelligent creature without purpose, that does not somehow act to obtain some desired outcome. In this sense the creature must be *rational* and take actions that it believes will accomplish its goals.

1.1.2 *A criterion of mind*

In one view, intelligence in both humans an animals has evolved through natural selection, and acts as a mechanism that allows an organism to adapt individually to its environment. George John Romanes defines a *criterion of mind* that indicates intelligent behavior in an organism:

> Does the organism learn to make new adjustments, or to modify old ones, in accordance with the results of its own individual experience?
> —George John Romanes (Animal Intelligence, 1882)

This should be seen in opposition to *instincts* which also involve mental operations that are adapted to the environment in pursuance of a goal, but which do not rely on individual experience and acquired knowledge. Romanes notes that it is impossible to draw a clear line between instinct and reason, and that different levels of intelligent behavior is better described as a continuum. However, in an attempt to make such a distinction, Romanes writes:

> For reason, involving a mental constituent, and besides being concerned in adaptive action, is always subsequent to individual experience, never acts but upon a definite and often laboriously acquired knowledge of the relation between means and ends, and is very far from being always similarly performed under the same appropriate circumstances by all the individuals of the same species.
> —George John Romanes (Animal Intelligence, 1882)

According to this definition, intelligence (or *reason* as Romanes writes) is to take actions adapted to the environment to achieve goals based on individual learned experience.

1.1.3 A working definition of intelligence

Based on the ideas presented so far, we could formulate the following definition of what we mean by *intelligence*:

> Definition 1.1 Intelligence
>
> The ability to learn and apply individual knowledge and skills to achieve goals.

While this definition captures some important aspects of the concept of intelligence, it is clearly also very limited. Within this definition, perhaps even some plants might be considered to be intelligent? Intelligence could also be defined to include other, more advanced aspects, such as logical reasoning, social and emotional aptitude, creativity, as well as self-awareness and consciousness. According to the definition above, an intelligent agent could easily be an uninventive, socially inept, illogical, nonconscious creature—but the simple definition might be a good starting point, if we want to attempt to implement intelligence in a machine.

1.1.4 *Artificial general intelligence*

We can imagine that one day it might be possible to create a machine that can perform all task that humans can perform. Philosophers refer to such imagined systems as *artificial general intelligence* or *strong AI*. In contrast, we might say that all artificial intelligence systems that have been created so far are *artificial specific intelligence* or *weak AI* because they exhibit intelligent behavior only for a specific problem within a limited set of conditions.

Even though no one have yet created a strong AI system, it might be worth to consider the consequences it would have for humanity. Scientists are approaching the creation of strong AI from two directions: One approach is the continuing effort to build more and more advanced weak AI systems, and expand their capabilities to growing domains of applicaions. Another approach is to replicate a biological brain in the form of a computer system, so that its behavior can be simulated in full scale.

On the one hand, we might be afraid that a strong AI would render humans worthless and take over the world:

> The development of full artificial intelligence could spell the end of the human race. [...] It would take off on its own, and redesign itself at an ever-increasing rate. Humans, who are limited by slow biological evolution, couldn't compete and would be superseded.
> —Stephen Hawking (2014)

On the other hand, presumably humans would be in control of the AI—and if things go in the wrong direction,

> [...] you just have to have somebody close to the power cord. Right when you see it about to happen, you gotta yank that electricity out of the wall, man.
> —Barack Obama (2016)

1.2 Learning

It is not easy to define precisely which tasks require intelligence to solve, and which tasks a simple computer program can manage, as the following pithy quotation from John von Neumann points out:

> If you will tell me precisely what it is that a machine cannot do, then I can always make a machine which will do just that!
> —John von Neumann (1950)

This point illustrates that learning and adapting one's behavior to new and unforseen circumstances is a key element of intelligence. If we are able to fully describe a problem, with no detail left vague or undefined, it should be possible to instruct a computer program to solve the problem. One of the important differences between humans and computers is that while humans can be quick to learn from their experience and modify their behavior, computers can basically only do exactly what they are programmed to do. So creating an artificial intelligence seems to require us to program a computer to *learn*.

Learning is the process of acquiring knowledge or skills that can later be put to use. The learning process comes in many forms: We can individually study the world and look for recognizable patterns, we can be instructed or shown examples, or we can interact and experiment with the world around us.

Definition 1.2 Learning

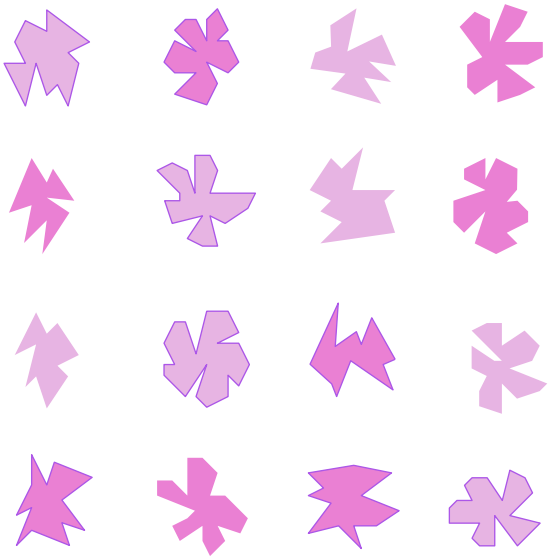
To acquire knowledge or skills by study, instruction, or experience.

1.2.1 *Categorization*

One way to understand and differentiate between different things in the world is through *categorization*. When we observe and interact with the world around us, we come to realize that some things are similar and some things are different in terms of their properties, such as how they look and behave, and what they can do for us. We tend to organize our understanding of the world by forming categories.

Example 1.1 Two kinds of puple objects

Imagine a world that consists of purple objects that all look fairly similar. While all objects are different, some are similar to each other in various ways, and so we can start to categorize them into different kinds. If you look at the following 16 examples of purple objects, can you tell me how many different kinds there are?



If you examine the objects carefully you will likely discover, that there are many different ways they can be categorized. They appear to come in different shades of purple, some are outlined, and the shapes also comes in different varieties.

Learning by categorization is an example of unsupervised learning: There is no teacher that tells us what to look for, and what the correct answer is. All we can do is observe the world around us, and start to look for patterns and regularities that help us struture it.

1.2.2 *Learning by example*

Another way to learn is *by example*. By looking at examples of objects we can learn to recognize them, and learn to generalize our knowledge by figuring out which properties and features are important.

Example 1.2 Kiptic or spurgle?

If I ask you if this unknown purple object is a *kiptic* or a *spurgle*, you probably wouldn't know what to answer.

> In a way, you are seing the object in the same way as a computer would: With no prior information to put the observation into context.
>
> In order to answer the question, you need to know more about kiptics and spurgles. One way to get that information is to simply show you some labelled examples of kiptics and spurgles. Once you have seen a few examples of each type of object, you can use that information to reason about the purple object above.
>
> Take a look at the examples in Figure 1.3. Can you now determine what type of object we are dealing with here?

Most people would agree that the *kiptic or spurgle* problem requires intelligence to solve. By looking at the labelled examples you probably formed a mental representation of the two types of objects. Then, you somehow compared the unknown purple object to this mental representation in order to classify it as a kiptic or a spurgle.

1.2.3 *The perception-action cycle*

We might see learning as a process that transfers external information into the mind of the learner, where it is represented in such a way that it can be accessed and utilized. In this view, learning is a flow from an external environment into the mind of the learner. However, perhaps this view on learning is too limited? Another view is that learning takes place as an interaction between the learner and the environment. In general, humans do not learn simply by being told—we learn by doing. This view of learning as a continuous circular flow of information that occurs when the learner interacts with her environment is known as the *perception-action cycle*.

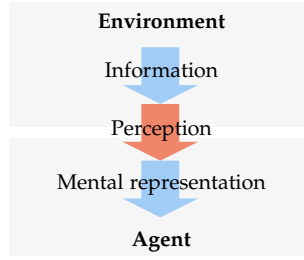


Figure 1.1: Simple learning

In the perception-action cycle (see Fig. 1.2) the learner has a mental representation of how the world works. When presented with a new situation, the mental model guides the learner to choose an appropriate action. By performing an action, the learner interacts with the world, which results in some outcome. The learner perceives the outcome and uses it to update her mental repesentation if necessary. This feedback can be positive or negative: If the world behaves as she expected, the learner can use this information to strengthen her existing beliefs, but if the outcome was unexpected, the learner must more

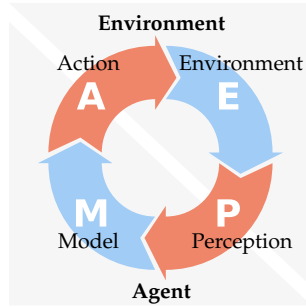


Figure 1.2: The perception-action cycle

fundamentally adjust or reorganize her world model.

1.3 Reasoning

Where *learning* is the process of aquiring knowledge and skills, *reasoning* is the process of manipulating the knowledge and adapting our skills to answer new questions and solve new problems. Reasoning is closely related the the concept of *rationality*: What we believe and how we act should be consistent with the information we have available and our own goals and objectives.

We can distinguish between three different modes of reasoning, called deduction, induction, and abduction.

Definition 1.3 Deduction, induction, and abduction

Deduction Reasoning from general premises to specific conclusions.

Induction Reasoning from specific facts to general rules.

Abduction Reasoning about the simplest or most likely explanation.

Example 1.3 A long line at the cafeteria

Deduction If we presume that there is always a long line at the cafeteria at noon, and we look at our watch to see that it is 12 o'clock, we can deduce that there is now a long line at the cafeteria. Given that the premise is true, the conclusion must also hold purely by logic; however, if there is a flaw in the premise, the conclusion might not be true.

Induction If we every day observe a long line in the cafeteria at noon, we might induce that it is a general rule that there is always a long line around at 12 o'clock. If, however, we one day go to the cafeteria at noon and there is no line, such a counterexample completely disproves the general rule.

Abduction If we happen to pass by the cafeteria and see that there is a long line, we might abduce that it is lunchtime. Since we know that most people tend to

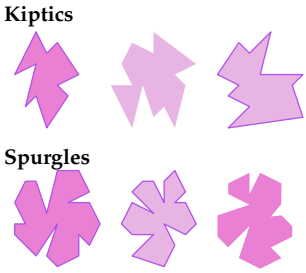


Figure 1.3: Some examples of kiptics and spurgles.

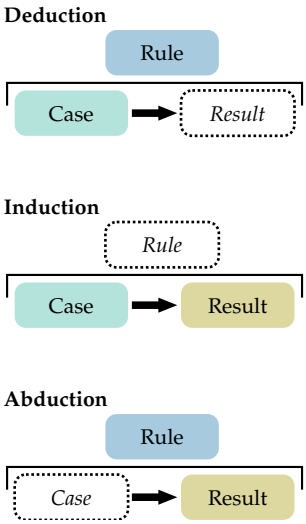


Figure 1.4: Deduction is reasoning from general rules to specific facts. Induction is reasoning from specific facts (observed cases and their results) to general rules. Abduction is reasoning about the most likely explanation.

> go to the cafeteria around noon, it is the simplest and most likely explanation for the long line; however, our abduction might be wrong if there is a flaw in our explanation—maybe the cafeteria are giving out free sandwiches today at 10 o'clock or something.

We can distinguish between *intuitive* and *formal* reasoning. Intuitive reasoning is based on instincts, feelings, and unconscious knowledge, whereas formal reasoning is based strictly on unambigous rules such as logic or mathematics. In practice, most human reasoning probably lies somewhere on the spectrum between these two extremes.

Problems

1. Come up with your own definition of the terms *intelligence* and *artificial intelligence*. Which aspects of these concepts do you think are most important?
2. Do you belive it is possible to create a thinking machine?
3. What is the most significant and profound example of the following topics you can think of?

Superhuman AI	Artificial intelligence that outperforms humans.
Creative AI	Artificial intelligence that emulates human creativity.
Animals intelligence	Intelligent behavior in animals (or perhaps plants).
Augmented intelligence	Enhancing human performance using artificial intelligence.

4. Is artificial intelligence strictly based on *formal reasoning* or can an AI be said to have intuition?

2 Statistics

Statistics can be defined as the collection, analysis, interpretation, and presentation of numerical data. We often distinguish between *descriptive statistics* which is concerned with presenting and summarizing data and *inferential statistics* which is aimed at drawing conclusions under uncertainty and random variation.

2.1 Descriptive statistics

The term *descriptive statistics* denotes methods that describe, summarize, and present data in a useful way, such that patterns, trends, and other characteristics are clearly visible. While presenting the raw data in its entirety can be useful, it might be difficult when the data set is large and complex. Often it is more effective to present and visualize summaries of the data, which focus on their most important aspects. This makes it easier to interpret and understand the data.

When we talk about a descriptive statistic, we simply mean some quantity that we compute from the observed data.

Definition 2.1 Statistic

A *statistic* is any number or quantity that is computed from data.

Two of the most important descriptive statistics are the *central tendency* and the *dispersion*.

The central tendency is a central or typical value which characterizes the data. The most common measure of central tendency is the *mean value*.

The dispersion is a measure of spread or variability. The most common measure of dispersion is the *standard deviation*.

Definition 2.2 Mean and standard deviation

Mean

The average or mean value of a set of numbers is often used to represent the central tendency of the numbers. It can be computed as:

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i.$$

Standard deviation

The standard deviation of a set of numbers is a measure of how much they are spread out. A low standard deviation (close to zero) means that all the numbers are close to the average, whereas a high standard deviation means that the numbers are dispersed on a large range. It can be computed as:

$$\sigma_x = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x)^2}.$$

The squared standard deviation, σ_x^2, is called the *variance*.

Example 2.1 Mean and standard deviation

Consider a data set which consists of five numbers:

$$x = \{1, 8, 4, 10, 2\}.$$

Let us compute the mean and standard deviation of this data:

$$\mu_x = \frac{1}{5}(1+8+4+10+2) = 5$$

$$\sigma_x = \sqrt{\frac{1}{5}\left((1-5)^2 + (8-5)^2 + (4-5)^2 + (10-5)^2 + (2-5)^2\right)}$$
$$\approx 3.46$$

Now, consider another data set which also consists of five numbers:

$$y = \{7, 5, 4, 3, 6\}$$

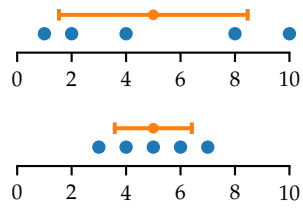


Figure 2.1: Two data sets from Example 2.1 with each observation shown as a dot. The indicated range is the mean values plus/minus one standard deviation.

The mean and standard deviation of this data is:

$$\mu_y = \frac{1}{5}(7+5+4+3+6) = 5$$

$$\sigma_y = \sqrt{\frac{1}{5}\left((7-5)^2+(5-5)^2+(4-5)^2+(3-5)^2+(6-5)^2\right)}$$

$$\approx 1.41$$

We can see that the two data sets have the same mean value, but the first data x have a much higher standard deviation than the second data y. In other words, the two data sets have the same central tendency but x is more dispersed (spread out) than y. The data is visualized in Figure 2.1.

2.2 Inferential statistics

The entire set of data that we want to study is called the statistical *population*. Most often we think of the population as a set of similar items (people, things, events, etc.) under study. A population can be a real and finite set (such as all all students in a class), a hypothetical set (such as all possible hands in a game of poker), or even an infinite, hypothetical set (such as all possible computer programs).

If the population is accessible and not too large, we might be able to gather all the relevant data from the population, and we can then compute any desriptive statistic we are interested in. However, often it is not practical to acquire data from the entire population. As an alternative, we can take a smaller sample from the population and use the sample statistics of the small sample to estimate the likely values of the parameters of the population.

Before we continue, let us define some of the most important terms we have just used.

Definition 2.3 Terms used in inferential statistics

Population	The entire set of items of interest.
Sample	A subset of the population.
Statistic	A numerical value computed from the sample.
Parameter	A characteristic of the population under study.
Estimator	A statistic used to estimate a parameter.

Example 2.2 Exit poll

Consider the following statement that talks about an exit poll conducted after an election has taken place, but before all the votes have been counted. The aim of the exit poll is to give a quick estimate of the results of the entire election:

> "An early exit poll taken among 1000 voters shows that the incumbent president has a small lead of 51 percent against her oponent."

The population is all voters who have voted at the election.

The sample is 1000 voters who were asked in the exit poll.

The statistic from the sample is that 51 percent voted for the incumbent president.

The parameter of the population we are interested in is the percentage of all voters who voted for the incumbent president.

The estimator we use is the 51 percent from the sample, which is used to estimate the population parameter.

There are two reasons why we might not trust that the incumbent president will be re-elected with 51 percent of the votes based on the exit poll:

1. Since only 1000 voters were asked, the estimate of 51 percent will be subject to random variation, since we would probably have got a different result if we had taken a different sample.

2. Maybe the 1000 voters are not even representative of the entire population—it depends on exactly how the sample was chosen.

Clearly, when we use a statistic from a small sample to estimate a parameter of a large population, we cannot expect that our estimate will exactly match the population parameter. While the population parameter is a fixed but unknown value, the estimator will depend on exactly how we have chosen our sample. If we choose a different sample, the estimator will be a different number.

Also, it might not be clear exactly which sample statistic is the best estimator for some population parameter. It turns out that the sample mean is a good estimator of the population mean, but to estimate the population standard deviation, a small correction is made in the formula.

Definition 2.4 Estimators of mean and standard deviation

Sample estimator of mean

The population mean is estimated by the sample mean:

$$m_x = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Sample estimator of standard deviation

The population standard deviation is estimated by the *corrected* sample standard deviation:

$$s_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - m_x)^2}.$$

Note that here we divide by $n - 1$.

When choosing a sample, it is important that the sample is representative of the population. If we are biased in the way we choose the sample, such that some items in the population are more likely than others to be included, the sample will not be representative of the population. One way to avoid any such biases is to choose the sample by a random procedure, such that all items in the population have an equal chance of being included in the sample. This is called a *simple random sample*.

Definition 2.5 Simple random sample

A set of n items chosen from the population such that each item is included with equal probability.

When we use a random sample, the estimator will be a random variable. By *random* we simply mean that the sample statistic used as our estimator will depend on exactly which sample was chosen, and that it would be different had we chosen a different sample. The good news is that we can say something about how much the sample statistic will vary across the different possible samples we can imagine.

2.2.1 *Standard error of the mean*

Let us say that we are interested in estimating the mean value in a population. Let μ and σ denote the mean and standard deviation of the population. Now, we take a simple random sample of size n, $\{x_1, x_2, \ldots, x_n\}$ and compute the sample mean

$$m_x = \frac{1}{n}(x_1 + x_2 + \cdots + x_n).$$

Since each item in the sample is independently chosen from the population, the variance of m_x can be computed as[1]

$$\sigma^2_{m_x} = \frac{\sigma^2}{n}.$$

Since we do not know the population variance σ^2, we can substitute in the sample variance s^2_x. Taking the square root gives us the final formula for the standard deviation of the sample mean, also known as the *standard error of the mean*:

Definition 2.6 Standard error of the mean

The standard deviation of the sample mean can be estimated as

$$\sigma_{m_x} = \frac{s_x}{\sqrt{n}}.$$

The formula tells us that when we estimate the mean of a population using the mean of a sample of size n, the standard deviation of our estimate is proportional to the standard deviation of the sample. We can see that if we increase the size of the sample n, the standar error will decrease. If we want a low

[1] This can be proven mathematically, but we will not do this here. It can be a good idea to check the formula empirically by simulating several random samples from a population, computing the sample mean for each sample, and finally computing the standard deviation of the sample means.

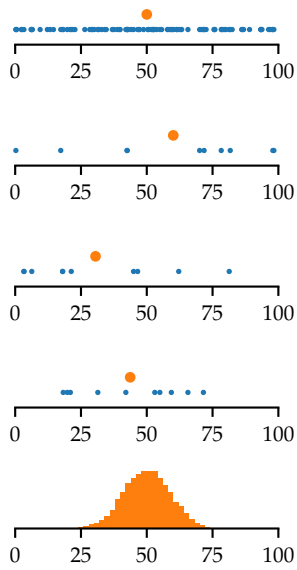


Figure 2.2: Top: A population of 100 numbers between 0 and 100 with population mean $\mu = 50$ and. Three rows in the middle: Three different simple random samples of size $n = 10$ from the population with the mean of in each sample indicated. Bottom: Histogram of sample means from 10 000 simple random samples of size $n = 10$ from the population.

error on our estimate, we need a large sample, but of course a large sample is typically more expensive or difficult to gather, so there is a trade-off to make.

Example 2.3 An AI system that estimates age

Let us imagine an AI system that can estimate the age of a person based on a photo. While the system is not expected to estimate the age correctly for everybody, it is supposed to be correct on average: This means that the error in the estimate should be zero on average. We decide to examine this by letting the system estimate the age of randomly chosen 10 people. We then compute the estimation error by subtracting the true age from the estimate. We get the following errors

$$x = \{7.1, -5.2, -2.6, -2.2, -6.4, -6.7, 1.5, -5., 6.8, 1.8\}.$$

Based on the measurements, we compute the mean error

$$m_x = -1.09.$$

We see that the mean error is negative, so it appears that the system tends to underestimate the age a bit. But it could also just be due to random variation that occured because we only examined 10 randomly chosen people. Maybe, if we had chosen 10 other people, the result would be very different.

To look further into this, we compute the sample estimate of the standard deviation to be $s_x = 5.16$, and use this to compute the standard error of the mean

$$\sigma_{m_x} = \frac{s_x}{\sqrt{n}} = \frac{5.16}{\sqrt{10}} \approx 1.63$$

This standard error tells us something about the variation we expect in our computation of the mean, when we examine a sample of 10 people. The variation in our estimate of the mean due to the random choice of sample is so large ($\sigma_{m_x} \approx 1.63$) that we know that the estimate $m_x = -1.09$ is subject to a lot of random variation, so we should be careful with our conclusions.

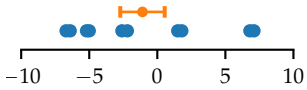


Figure 2.3: Data from Example 2.3 with each observation shown as a dot. The indicated range is the mean value plus/minus one standard error of the mean.

2.2.2 *Confidence intervals*

We have seen how we can use a statistic computed on a sample to estimate a population parameter, and we have discussed how the standard deviation of the statistic tells us something about how much the sample statistic would vary across different possible random samples. We can use these ideas to create a *confidence interval* which roughly speaking is a range of potential values in which the population parameter is estimated to lie.

A confidence interval is also a sample statistic, which means that it is an interval that is computed from a sample that we use to infer something about the population parameter. Just like any other sample statistic, the confidence will also be subject to random variation caused by the selection of the random sample. In other words, if we had chosen another sample, we would also get another confidence interval.

A procedure for computing a confidence interval has an associated *confidence level* or *coverage*. The confidence level specifies the proportion of confidence intervals (across all possible samples we can imagine) that actually contain the true population parameter. If a procedure for computing a confidence interval has a confidence level of 95%, it means that if we imagine taking 100 random samples from the population, each of which has the same sample size, and we compute the confidence interval for each of the samples, we would expect that 95 of the computed confidence intervals would include the true population parameter.

Usually we only have a single sample from our population, and so we can only compute a single confidence interval. Then we have no way to determine whether or not the population parameter actually is in our interval or not. With a confidence level of 95%, all we can say is that 95% of the times we use our procedure to compute a confidence interval, it will include the true value, and for a single confidence interval estimate there is only a 5% chance that we were so unlucky with our random sample that the population parameter is outside our interval.

Why not set the confidence level to 100% then? It is actually very easy to make a procedure for computing a confidence interval with 100% confidence level: We can simply take the interval from minus infinity to infinity, which of course is guaranteed to include the population parameter. But such an

interval is obviously not very useful. It is more interesting to find a narrow interval, but in order to do this we need to make a compromise by accepting a lower confidence level.

There are many different methods for constructing confidence intervals that are based on different assumptions. Here we will consider the most simple confidence interval which is based on the assumption that the sampling distribution of the mean is approximately a normal distribution (see Figure 2.4). In a normal distribution with zero mean and standard deviation equal to one, the probability of getting a value less than -1.96 or greater than 1.96 is 5%. So to construct a 95% confidence interval, we can simply take an interval around the sample estimate of the mean that extends to ± 1.96 times the standard error.

Definition 2.7 95% confidence interval for population mean

A 95% confidence interval for the population mean can be computed as:

$$m_x \pm 1.96 \cdot \sigma_{m_x} = 1.96 \cdot \frac{s_x}{\sqrt{n}}$$

Note that this interval is only precise when the sample size is sufficiently large ($n \geq 30$) and constitutes a small fraction of the total population ($n \leq N \cdot 10\%$).

2.2.3 *Estimating a proportion*

In some cases, the population parameter we are interested in is a *proportion*, i.e., how many items out of the total population that have some property of interest. If we had access to the entire population, we could compute the proportion π as[2]

$$\pi = \frac{\eta_x}{N},$$

where η_x is the number of items in the population that has the property, and N is the total number of items in the population. We can think of a proportion as a special kind of mean value by defining an *indicator variable*

$$z_i = \begin{cases} 1 & \text{if object } i \text{ has the desired property} \\ 0 & \text{otherwise.} \end{cases}$$

This indicator takes the value 1 if and only if the ith item has the property we are looking for. We can now compute the

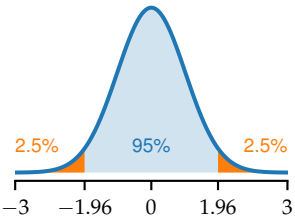


Figure 2.4: Normal distribution with zero mean $\mu = 0$ and unit standard deviation $\sigma = 1$. 5% of the probability mass lies beyond the critical value ± 1.96.

[2] If the population were infinite, we would need to define the proportion as an appropriate limit.

proportion as the mean value of the indicator

$$\pi = \frac{1}{N}\sum_{i=1}^{N} z_i.$$

To estimate a population proportion from a sample, we can use the observed sample proportion.

$$p = \frac{n_x}{n},$$

where n_x is the number of items in the sample that has the property, and n is the sample size. Since the proportion is a mean value, we can compute the standard error of the proportion with the formula in Definition 2.6. The sample estimate of the standard deviation of the proportion can be computed as

$$s_p = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(z_i - p)^2} \approx \sqrt{p(1-p)}.$$

Plugging this into the definition of the standard error gives us the following:

Definition 2.8 Standard error of a proportion

The standard deviation of the sample estimate of a proportion can be estimated as

$$\sigma_p = \sqrt{\frac{p(1-p)}{n}}.$$

Similar to the confidence interval for the population mean, we can create a confidence interval for the population proportion:

Definition 2.9 95% confidence interval for a proportion

A 95% confidence interval for the population proportion can be computed as:

$$p \pm 1.96 \cdot \sigma_p = 1.96 \cdot \sqrt{\frac{p(1-p)}{n}}$$

Note that this interval is only precise when the sample size is sufficiently large ($n \geq 30$) and constitutes a small fraction of the total population ($n \leq N \cdot 10\%$), and the population proportion is not too extreme ($10\% < p < 90\%$).

When the population proportion is either very large or very small, say less than 10% or greater than 90%, the confidence interval above is not very good. As a better alternative, we can use the Agresti-Coull interval, which is very similar but uses a correction in the estimation of the population proportion. When estimating the proportion from the sample, the correction consists of adding two to the numerator and four to the denominator,

$$\tilde{p} = \frac{n_x + 2}{n + 4}.$$

This corresponds to having a sample size of $\tilde{n} = n + 4$ where we artificially add four extra items of which two have the property that the proportion measures. With this correction, the confidence interval is given by:

Definition 2.10 Agresti-Coull interval

$$\tilde{p} \pm 1.96 \cdot \sqrt{\frac{\tilde{p}(1-\tilde{p})}{\tilde{n}}}$$

$$\tilde{p} = \frac{p \cdot n + 2}{n + 4}$$
$$\tilde{n} = n + 4$$

2.3 Sample size calculation

Now that we know how to compute a confidence interval both for a population mean and proportion, we can use the same formulas in reverse to calculate how large a sample size is needed to estimate a parameter within a certain desired margin or error. By isolating the sample size in Definition 2.7 we arrive at the following formula for the sample size n:

Definition 2.11 Sample size calculation for a mean

At a 95% confidence level, the required sample size to estimate a mean within a given margin of error is:

$$n = 1.96^2 \frac{s_x^2}{e^2}$$

n Required sample size

s_x^2 The expected sample variance. This is an unkown quantity, but we can substitute in our best guess, perhaps based on a small pilot study.

e Desired margin of error (half width of the confidence interval).

Prior to gathering data from a sample of the population, we can use this to give us an indication of how large a sample is needed in order to estimate a population parameter within a certain margin of error. Since we don't know the variance in advance, we can substitute in our best guess, or estimate it using a small pilot study.

Similarly, we can isolate the sample size in Definition 2.9 to arrive at a formula for computing a sample size for a proportion.

Definition 2.12 Sample size calculation for a proportion

At a 95% confidence level, the required sample size to estimate a proportion within a given margin of error is:

$$n = 1.96^2 \frac{p(1-p)}{e^2}$$

n Required sample size

p The expected sample proportion. This is an unkown quantity, but we can substitute in our best guess or use $p = 50\%$ as a worst case estimate.

e Desired margin of error (half width of the confidence interval).

The required sample size is greater when estimating proportions close to 50%, so to be most conservative we can adjust our guess a bit away from the extremes or simply plug in 50% as a

worst case guess.

Example 2.4 Sample size for image recognition accuracy

Let's say we have an image recognition system that has been tested thoroughly and has a recognition accuracy of 90%. We have now developed a new system, which we think is better than the old system. To examine this, we decide to make an experiment, where we classify a bunch of images with the new system to verify its accuracy. How many images should we test the new system on?

We can calculate the required sample size using the formula in Definition 2.12. We think that the new system has an accuracy of around 95%, so we decide that we need to estimate the accuracy within a margin of error of at least $\pm 5\%$. Based on this, we get

$$n = 1.96^2 \frac{0.95(1-0.95)}{0.05^2} \approx 73.$$

We then classify 73 images and end up with, say, 67 correct classifications. The sample estimate of the accuracy is then

$$p = \frac{67}{73} \approx 91.8\%$$

and we can compute the confidence interval as

$$p \pm 1.96 \cdot \sqrt{\frac{p(1-p)}{n}} \approx 91.8\% \pm 6.3\%$$

In other words, our estimated range for the accuracy is 85.5%–98.1%. Based on this we can not even tell if the new system is better than the old one or not.

On second thought, we are not so sure the new system will actually be 5 percentage points better than the old system. To be on the safe size, we decide to reduce the margin of error to $\pm 1\%$ and assume that the accuracy is only around 90%. With these more conservative assumptions the required sample size is

$$n = 1.96^2 \frac{0.9(1-0.9)}{0.01^2} \approx 3457$$

We new classify 3457 images and end up with, say, 3170 correct classifications. The sample estimate of the accuracy

is then

$$p = \frac{3170}{3457} \approx 91.7\%$$

and we can compute the confidence interval as

$$p \pm 1.96 \cdot \sqrt{\frac{p(1-p)}{n}} \approx 91.7\% \pm 0.9\%$$

With an estimated range for the accuracy of 90.8%–92.6% we are now confident that the new system is indeed a few percentage points better than the old system.

Problems

1. Consider a data set that consists of six numbers:

$$x = \{1, 5, 2, 7, 3, 8\}$$

   What is the *mean* and *standard deviation* and *variance* of the data?

2. John plans on taking a taxi home from work every day next year. He would like to know how long the commute will be on average, and to find out he takes a taxi home five times during.

   (a) In this scenario, what would be the population, sample, statistic, parameter, and estimator?

   (b) The five taxi rides take $t = \{42, 34, 29, 42, 48\}$ minutes. Compute a confidence interval for the average time.

3. A face recognition system has been designed to identify members of the public as potential criminals. When testing the system on 100 000 random people, the developers found that it had identified 1 500 people as potential criminals. Out of these 1 500 people, further analysis showed that 1 350 were actually innocent, whereas the remaining 150 were indeed criminals. Now, considering if we want to employ the system, we are interested in finding out how large a proportion of the people identified as criminals are actually criminals.

   (a) In this scenario, what would be the population, sample, statistic, parameter, and estimator?

   (b) Compute a confidence interval for the proportion.

Solutions

1. $\mu \approx 4.33$, $\sigma \approx 2.56$, $\sigma^2 \approx 6.56$.

2.(a) The population is all (hypothetical) taxi rides from work to home that John will take next year; the sample is the five rides; the statistic is the mean duration of the ride; the parameter is the mean duration of the rides next year; and the estimator is that we use the mean of the five rides to estimate the mean of next year's rides.

(b) $m_t = 39 \pm 6.56$

3.(a) The population is all (hypothetical) matches (people identified as criminals) when the system is employed; the sample are the 1 500 matches; the statistic is the sample proportion $p = \frac{150}{1\,500}$; the parameter is the proportion of correctly identified criminals when the system is employed; and we use the sample proportion as estimator.

(b) $p = 10\% \pm 1.5\%$

3 Algorithms and complexity

An algorithm is a complete and unambiguous specification of a procedure to solve a problem, expressed in a well-defined formal language. Sometimes people liken an algorithm with a cake recipe, but that is perhaps an ill-chosen metaphor, because cake recipes are often incomplete, ambiguous, and expressed in idiomatic and informal jargon. For example, in a cake recipe one might encounter instructions such as *season to taste* or *bake until golden*. In addition to being unequivocal, an algorithm must also begin with well defined initial conditions and be guaranteed to produce an output through a finite number of steps.

3.0.1 High level and implementation description

When we describe an algorithm we often have two competing goals:

1. To explain the general idea, motivation, approach and method used in the algorithm to solve the particular problem at hand.

2. To provide sufficient detail to allow others to implement the algorithm and reproduce its results.

In a *high level description* of an algorithm, we describe the algorithm in normal language and ignore implementation details that are not necessary to understand the general approach. In an *implementation description*, we spell out exactly which actions must be performed and detail the initial conditions etc.

An implementation description is often specified in a formal language, such as (pseudo) computer code or in the form of a flowchart.

Example 3.1 Finding the smallest of three numbers

Let us say that we have three numbers, a, b, and c, and we are interested in finding the smallest number of the three. While we may say that the previous sentence is a high level description of the *problem*, it does not describe an algorithm. Let us formulate a high level description of an algorithm that can solve the problem:

> We go through each of the numbers and at each step we keep track of the smallest number seen so far. Once we have gone through all three numbers, the smallest number we have seen will be the smallest number of the three.

While this description characterizes the general idea, it leaves out many details. Next, we can spell out the implementation details, here in the form of a Python program:

```
def smallest_of_three(a, b, c):
    """Returns the smallest number amongst a, b, and c"""
   smallest = a
   if b < smallest:
      smallest = b
   if c < smallest:
      smallest = c
   return(smallest)
```

3.0.2 *Comparing algorithms*

There are usually many ways to solve a problem, so there might be many different algorithms that give the same result, but differ in terms of other properties. The two most important properties of algorithms are:

Efficiency How efficient is the algorithm in terms of its computational complexity and use of resources?

Simplicity How simple is the algorithm in terms of comprehensibility and amenability to be implemented on a computer?

Example 3.2 Multiplication of complex numbers

We know that two complex numbers can be multiplied together according to the following rule:

$$(a + ib) \cdot (c + id) = (ac - bd) + i(bc + ad).$$

Thus, the most simple and direct way to implement this would be something like the following.

```
def complex_multiplication(x, y):
    """Multiplies the complex numbers x and y"""
    (a,b), (c,d) = x, y
    real = a*c - b*d
    imag = b*c + a*d
    return((real, imag))
```

This algorithm requires four multiplications and two additions/subtractions. However, another algorithm that produces the exact same result was invented by Carl Friedrich Gauss, requiring only three multiplications and five additions/subtractions.

```
def gauss_complex_multiplication(x, y):
    """Multiplies the complex numbers x and y"""
    (a,b), (c,d) = x, y
    k1 = c * (a+b)
    k2 = a * (d-c)
    k3 = b * (c+d)
    real = k1 - k3
    imag = k1 + k2
    return((real, imag))
```

In the good old days, multiplication was a much more expensive and time-consuming computation compared to addition and subtraction, both when it was carried out by hand and on a computer. For that reason, Gauss' algorithm is more efficient but certainly less easy to understand.

3.1 Algorithmic complexity

In the analysis of algorithms, *algorithmic complexity* is an approach to classifying algorithms according to their computational demands. The *time complexity* is a measure of how fast the algorithm will run on a computer. Since not all computers are the same, the time complexity is most often measured by the required number of elementary operations, such as multiplications and additions etc. Similarly, the *storage complexity* of an algorithm measures how much memory the computer needs in order to run the algorithm. In the following we will focus on time complexity, but most of the discussion applies to analysis of storage complexity as well.

3.1.1 *Dependence on input*

For many algorithms, the running time will depend on the input: If the input to some algorithm is a list of numbers, the running time might depend both on the length of the list (how many numbers the algorithm needs to process), but also on the specific values on that input list.

> Example 3.3 Linear search
>
> Let us consider an algorithm designed to identify whether or not a particular number `a` occurs somewhere in a list of numbers `x`. Such an algorithm might be implemented by the following Python code:
>
> ```
> def contains_value(x, a):
>     """Returns True only if list x contains value a"""
>     for val in x:
>         if val==a:
>             return(True)
>     return(False)
> ```
>
> The algorithm will be very fast in case there is a value `a` somewhere in the beginning of the list. Once the algorithm has found an `a`, it can return in the affirmative, whereas if there is no `a` anywhere on the list, the algorithm must visit all the values on the list `x` before it can return its negative result.

3.1.2 *Best, average, and worst case*

To take into account that the complexity of an algorithm can depend on the value of the input, we can for example specify the best case, average case, or worst case complexity.

The best case complexity is the minimal complexity attainable given the most favorable input.

The worst case complexity is the maximal complexity attainable given the most unfavorable input.

The average case complexity is the complexity attained for some reasonable definition of an average use case. The average case might be analyzed through a set of typical inputs, or computed as the average complexity over all possible inputs.

We can then quantify the complexity by a function $T(n)$ that counts the number of operations required for the algorithm to process an input of length n.

Example 3.4 Best, worst, and average case complexity

Consider again the algorithm in Example 3.3 that determines whether or not a value `a` is somewhere on a list.

In the best case the value `a` occurs as the first element on the list, and the algorithm will finish after making a single variable comparison. We can then say that the best case time complexity is $T(n) = 1$.

In the worst case the value `a` does not occur on the list, and the algorithm will finish only after comparing `a` to all input values. This requires n comparisons, where n is the length of the input list. We say that the worst case time complexity is $T(n) = n$.

In the average case we would need a further assumption about how input lists might look like in a typical use case. For example, we might assume that about half the time, the list does not contain `a`, and the other half of the time it contains a single `a` at some random position. In the fist case, it requires n comparisons, whereas in the second case it requires $\frac{1}{2}n$ comparisons on average. In total, under these assumptions the average case would require $\frac{1}{2}(n + \frac{1}{2}n) = \frac{3}{4}n$ comparisons. We can then say that the average case time complexity is $T(n) = \frac{3}{4}n$.

3.1.3 *Big O-notation*

Big O-notation is useful for analyzing how much time or how many operations it takes to compute an algorithm. It is an asymptotic notation, that expresses how the computational complexity grows as the input grows in size. Formally, the big O-notation can be defined as follows.

Definition 3.1 Big O-notation

The computational complexity is

$$T(n) \in O\big(f(n)\big)$$

if and only if there exists a constant c such that $T(n) < c \cdot f(n)$ for all $n > n_0$. We say $f(n)$ is an asymptotic upper bound for $T(n)$.

Here, $T(n)$ is the time complexity that measures the number of computational operations required to run the algorithm. The big O notation focuses only on how the complexity grows with n: It ignores any multiplicative constants and only considers the fastest growing term in $T(n)$.

Example 3.5

Let's say that we have determined that an algorithm requires

$$T(n) = 2n^2 + 10n + 50$$

computational operations to run. Now, let us determine the asymptotic complexity: We can verify that

$$T(n) < 6n^2,$$

for all $n > 5$ (see Figure 3.1) so the computational complexity is "big O of n squared",

$$T(n) \in O(n^2).$$

We can easily get to this conclusion simply by identifying the fastest growing term and ignoring any multiplicative constants.

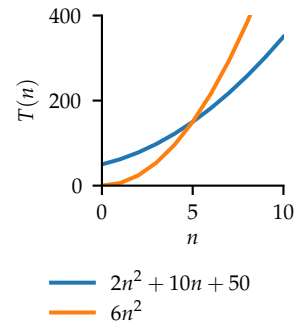


Figure 3.1: Example of running time complexity.

Example 3.6 Big O for linear search

Consider again the algorithm in Example 3.3 that determines whether or not a value a is somewhere on a list. In Example 3.4 we determined under certain assumptions that the average case complexity of the algorithm was

$$T(n) = \tfrac{3}{4}n$$

Considering Definition 3.1 we can see that $f(n) = n$ is an asymptotic upper bound for $T(n)$, and so we can say that the computational complexity is "big O of n",

$$T(n) \in O(n).$$

Example 3.7 Duplicates on a list

The following algorithm goes through each unique pair of values in a list `x`, and determines whether or not the list contains one or more duplicates. As soon as the algorithm finds a duplicate, it will stop and return `True`. If the list does not contain any duplicates, the algorithm will return `False` after comparing all possible pairs of values.

```
def contains_duplicates(x):
    """Returns True if list x contains any duplicates"""
   for i in range(len(x)-1):
      for j in range(i+1, len(x)):
         if x[i] == x[j]:
            return(True)
   return(False)
```

If the list contains n numbers, the total number of unique pairs to compare is $n(n-1)/2$ (can you show this?). Thus, the worst case computational complexity occurs in the situation where the list contains no duplicates. Counting the number of variable comparisons needed to run the algorithm, we end up with a worst case computational complexity of

$$T(n) = \frac{n(n-1)}{2} = \tfrac{1}{2}n^2 - \tfrac{1}{2}n.$$

We can see that $T(n)$ is upper bounded by $c \cdot n^2$ for all $n > n_0$ with $c = \frac{1}{2}$ and $n_0 = 0$. Therefore we can say that the computational complexity is "big O of n squared",

$$T(n) \in O(n^2).$$

Problems

1. Consider the algorithm for finding the smallest of three values described in Example 3.1. Counting the number of *variable assignments* and *variable comparisons*, what is the best case, average case, and worst case time complexity of the algorithm.

Solutions

1. In the best case `a` is the smallest, so we only need to make the first variable assignment and compute the two comparisons, so $T = 3$. In the worst case `c` is smallest and `b` is smaller than `a`, so all code lines will be run and $T = 5$. In the average case, we can consider all six possible orderings of the three numbers: On average we have $T = 3.83$.

4 Symbolic AI

One approach to creating artificial intelligence is based on creating a symbolic representation of the problem and solve it using logic and search. By *symbolic* we mean a human-understandable representation, for example through well defined states, rules, and actions to take. This general approach is known as *symbolic AI*. Once we have formulated a symbolic representation of a problem, we can solve it by using logic to deduce the logical consequences of the rules of the system, or by searching through the space of possibilities to find the most optimal solution.

4.1 Logic

Symbolic AI is based on the rules of logic. Many different formal systems of logic have been developed through the history of mathematics. Here we will consider the fundamental algebraic logic that deals with reasoning about binary variables.

4.1.1 Boolean algebra

In Boolean algebra (named after George Boole) variables can only take two values: true or false. Often the values are denoted by 1 (true) and 0 (false), but these values should not be confused with the usual integers 0 and 1. Our usual mathematical operations, such as addition and mulitiplication, do not really make sense for values which can only either be true or false. Instead, we have the three operations *and*, *or*, and *not*. Perhaps a little confusingly, the operations *and* and *or* are usually denoted by a multiplication and addition symbol, whereas the negation is often denoted by a line above the expression.

Definition 4.1 Boolean operators

$$
\begin{aligned}
\text{and:} \quad a \cdot b &= \begin{cases} 1 & a = b = 1 \\ 0 & \text{otherwise} \end{cases} \\
\text{or:} \quad a + b &= \begin{cases} 0 & a = b = 0 \\ 1 & \text{otherwise} \end{cases} \\
\text{not:} \quad \overline{a} &= \begin{cases} 1 & a = 0 \\ 0 & a = 1 \end{cases}
\end{aligned}
$$

Example 4.1 Truth table

The two Boolean variable a and b can each take two values: 0 or 1. Thus in total there are four different combinations of values that a and b can take. We can list these combinations in a table with four rows. In addition, we can also list the results of applying the functions *and* and *or* to the two variables. This results in the *truth table* show in Figure 4.1.

a	b	$a \cdot b$	$a + b$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Figure 4.1: Truth table for the *and* and *or* operators.

The Boolean *and* and *or* operators follow almost the same algebraic rules as the usual multiplication and addition operators: The commutative, associative, and distributive laws hold as we are used to. On top of that there are a number of other laws of Boolean algebra, that can be used to manipulate and simplify Boolean expressions.

Definition 4.2 Properties of Boolean algebra

Commutative law	$a + b = b + a$
	$a \cdot b = b \cdot a$
Associative law	$a + (b + c) = (a + b) + c$
	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$
Distributive law	$(a \cdot b) + (a \cdot c) = a \cdot (b + c)$
	$(a + b) \cdot (a + c) = a + (b \cdot c)$
Double negative law	$\overline{\overline{a}} = a$
Identities	$a + 0 = a, \quad a + 1 = 1$
	$a \cdot 1 = a, \quad a \cdot 0 = 0$
Complement law	$a + \overline{a} = 1$
	$a \cdot \overline{a} = 0$
Absorbtion law	$a + a \cdot b = a$
	$a + \overline{a} \cdot b = a + b$
DeMorgan's law	$\overline{a \cdot b} = \overline{a} + \overline{b}$
	$\overline{a + b} = \overline{a} \cdot \overline{b}$

4.1.2 *Boolean functions*

A Boolean function $f(x_1, x_2, \ldots, x_n)$ takes as input n of these true/false values and produces an output in the form of a single true/false value. Any such a function can be created by combining the three Boolean operators.

Example 4.2 Crossing the road with a Boolean function

Let us consider a Boolean function that describes the logic behind how an individual might decide whether or not to cross the road: This individual will walk if there is a green light and no traffic to be seen. In addition, she will also walk if there is no traffic and no police to be seen, regardless of the color of the traffic light. This behavior can be described by the function:

$$\text{walk} = (\text{green and not traffic}) \text{ or } (\text{not traffic and not police})$$

Using the Boolean notation introduced above, we might equally write

$$w = (g \cdot \overline{t}) + (\overline{t} \cdot \overline{p}),$$

where we now have used short-hand notation for each of the Boolean variables (w = walk etc.). We can simplify this expression by using the laws of Boolean algebra

$$w = \overline{t} \cdot (g + \overline{p}).$$

What would happen, if there is no traffic ($t = 0$), the light is not green ($g = 0$), and the police is present ($p = 1$)? We can simply insert these truth-values in the equation, and compute w as follows:

$$w = \overline{0} \cdot (0 + \overline{1}) = 1 \cdot (0 + 0) = 1 \cdot 0 = 0.$$

So in this situation our pedestrian would not walk ($w = 0$).

4.2 Rule based systems

According to our definition of intelligence, an intelligent system has the ability to learn and apply knowledge and skills to achieve goals. For now, let us set aside the question of how we learn and aquire knowledge and skills, and focus on the subproblem of how we can apply knowledge to solve problems. To answer this, we need to define more precisely what we mean by *knowledge*, and consider how we can represent that in a computer system.

One way to think of knowledge is to envision a human *expert* who posesses all the knowledge that we need to solve our task. If we had access to such an expert, we might be able to extract the knowledge, and distill it into some practical and operational format. This is the idea behind *expert systems*.

One way to represent knowledge is as a collection of *facts* and *rules*, which describe all the things that exist as well as their properties and relations. Of course, in practise we would need to limit the scope to handle only a limited number of facts and rules within some problem domain. In a classical expert system, the rules are specified through a (possibly large) number of *if-statements*. An if-statement contains two parts called the *condition* and the *consequent*.

if <condition>

then <consequent>

The condition is a logical statement, and if the condition is true the consequent is executed. In this case, we say that the rule has *fired*. If the condition is false (the rule does not fire), the consequent is not executed and the rule is not applied.

Example 4.3 Crossing the road with an expert system

Continuing Example 4.2, let let us define the possible observable facts, that we need to take into consideration:

light	The state of the traffic light can either be red, amber, or green.
road	The state of the road can be traffic or none.
surveillance	The state of surveillance can be police, or none.

With these three observables which can each take two or three different values, our system can be in $3 \cdot 2 \cdot 2 = 12$ different possible states. Now, let us define a set of expert rules inspired by the individual in Example 4.2.

if *light* is green and *road* is none
then *walk* is true

if *surveillance* is none and *road* is none
then *walk* is true

At first we might think that these two rules describe the same function as in Example 4.2, but they do not. The difference is that the expert rules are only used to positively define the cases that are described in the two conditions—all other situations are left undefined. In a rule based expert system, we need to define all the needed actions as rules. We might then add another rule, which says when to not cross the road:

if *road* is traffic
then *walk* is false

Even with that rule in place, there are still some situations that are left undefined. To get a better overview, we can list all the 12 different possibles states in a table, and write in the cases that are handled by our expert rules. If we do this, we get the following table for the *walk* variable:

	road	traffic		none	
	surveillance	police	none	police	none
light	red	0	0	?	1
	amber	0	0	?	1
	green	0	0	1	1

Here we see, that there are still two situations in which the result is undefined, namely when there is no traffic, there is police present, and the traffic light is not green.

4.2.1 *Resolving conflicts*

Once we start creating more complex expert systems, it becomes difficult to avoid conflicting rules. If we have two rules which fire at the same time, but which have conflicting consequences, we need some mechanism to decide which of the rules should have preference. It would not be very useful to require that all expert rules are free from conflict, since this would make the process of writing down the rules very cumbersome and unintuitive. Another idea is to use a simple conflict resolution mechanism. One such mechanism is to order the rules according to their priority. If two or more rules are in conflict, the rule with greatest priority wins, and the rest are ignored.

Example 4.4 Conflict resolution

Let us consider a simple system with just two rules

if *weather* is rain
then *recommend* is stay_home

if *weather* is sun or *clothes* is rain_coat
then *recommend* is go_out

Here we might have a conflict: If it rains and we are wearing a rain coat, both rules will fire. According to the first rule, the recommendation is to stay home, and according to the second rule, we should go out.

If we assume that the rules are listed in increasing order of priority (i.e. the second rule has priority over the first), then the conflict is resolved, and the recommendation is to go out in the rain with the rain coat on.

4.2.2 *Forward and backward chaining*

In the examples considered so far, the variables considered in the conditions have not been affected by any of the consequents. When that is the case, an expert system can simply be implemented by running each rule that applies and observing the outcome. However, if some of the consequents influence the conditions of other rules, the situation is more complicated.

We can distinguish between two modes of reasoning in a rule based system, called forward and backward chaining.

Forward chaining is resoning from the data: The rules are applied from the top, and once the first applicable rule fires its consequent generates a new piece of information which might influence which rules are applicable. Then the rules are applied again from the top, and again the top most rule that applies is fired. Each rule is allowed to fire only once, and the process continues until no more rules can fire.

Backward chaining is reasoning aimed at proving an outcome: First, the set of rules are searched to find the rules that are able to generate the outcome we are interested in. Such rules must have the desired outcome in their consequent. If there exists one or more such rules, we examine each of them in turn. We look at the condition, and if it is true, our outcome is proven. Otherwise we repeat the process by looking for rules that have these conditions as consequents and continue the recursion backward. Once there are no more rules to consider, we have either proven or not proven our outcome.

Example 4.5 Walk or drive to work

Let us consider an expert system that helps decide if we should walk or drive to work. The observable facts related to our car and the weather can take the following values:

fuel_tank	empty, not_empty
battery	flat, not_flat
birds	singing, not_singing
sun	shining, not_shining

We might then create the following set of rules, listed in order of increasing priority:

1. **if** *weather* is not_nice
   then *action* is drive

2. **if** *car* is dead or *weather* is nice
   then *action* is walk

3. **if** *fuel_tank* is empty or *battery* is flat
   then *car* is dead

4. **if** *sun* is shining and *birds* is singing
   then *weather* is nice

5. **if** *sun* is not_shining
   then *weather* is not_nice

Let us assume we are given the following initial facts: *fuel_tank* is not_empty, *battery* is not_flat, *sun* is not_shining, and *birds* is singing.

Forward chaining Let us examine what we can derive from the initial facts. In the first round, the top most rule that applies is rule 5, which gives us the new fact *weather* is not_nice. In the second round, the top most rule that applies is rule 1, which gives us the new fact *action* is drive. In the third round, no further rules apply. Thus, the two facts we have derived are the full set of facts that can be derived from the initial facts.

Backward chaining Let us see if we can prove that *action* is drive. The only rule that can generate this outcome is rule number 1, so we need to prove its condition *weather* is not_nice. The only rule that can prove this condition is rule number 5, so we need to prove its condition *sun* is not_shining. Since we have that fact in our initial set of facts, we have now proven the statement *action* is drive by backward chaining.

4.3 Search

In the expert systems discussed so far, the goal has been to write down a set of rules that would lead to the best decision or action based on the observable facts. We can think of the observable facts as the *state* or the environment we are acting in, and our job is to make an AI system that can map any state to an appropriate action. In many situations it can be difficult to

define what the optimal action is: Rather, our desire is often to act in such a way that we achieve some goal in the long run.

By taking actions, we can modify the state of the environment, and our goal might be formulated in terms of reaching a certain desirable environment state. In this setup, we can formulate the problem using a graph where each node is a state and each edge between two states corresponds to an action. The objective is then to traverse the graph to reach the goal state starting from a given initial state.

Example 4.6 Removing paper jam

Consider a printer which can either be jammed or clear as well as have an open or closed lid. In total, the printer can be in four different states as illustrated in Figure 4.2. In each state we have three possible actions: We can *open* or *close* the lid of the printer, or we can *remove jam*. We can only remove the paper jam if the lid is open, and obviously, opening the lid if it is already open has no effect, so it does not change the state of the printer.

Initially, the printer is in the jammed, closed state. Our objective is to find the sequence of actions which will bring the printer into the cleared, closed state. This is easy, of course: We just have to open the lid, remove the jam, and close the lid again. Examining the state diagram, we can easily get to this solution by tracing the path from the initial state to the goal state. The reasons why this is an easy problem are:

1. We have *semantic* information (we know stuff about printers) that helps us solve the problem.
2. We have a simple state transition diagram of managable size in which we can visually trace the optimal path.
3. There happens to be exactly one unique way to get to the goal state.

If the state diagram was more complicated, it might not be so easy to spot the solution, or to determine if a solution even exists. In such a case, we could let a computer algorithm *search* through the state space for a solution.

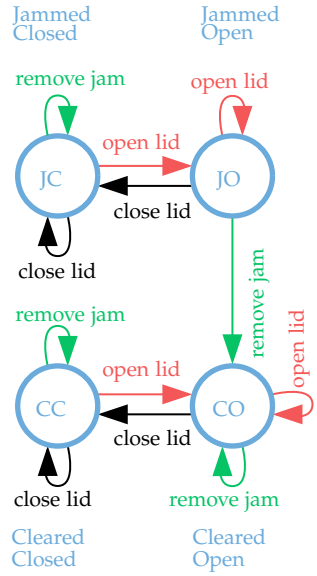


Figure 4.2: Example of a system with four states and three actions: A printer can be either Jammed or Cleared as well as Open or Closed. In combination, the printer can be in four different states, *JC*, *JO*, *CC*, and *JC*. In each state there are three possible actions: *remove jam*, *open lid*, and *close lid*. Starting in state *JC* the goal is to get to the state *CC*.

4.3.1 A search algorithm

The idea behind most search algorithms is fairly simple. We start at an initial state and explore all the neighboring states that we can go to by taking different actions. This gives us a new set of states to explore. If one of the neighboring states is a goal state, we have found a solution. If not, we need to explore one step further, so we take one of the neighbor state and explore all its neighbors and so on until we find the goal state. Since the state diagram can have cycles, we need to keep track of which states we have already seen, and make sure we do not explore each state more than once.

To make the algorithm more specific, we envision that we always have a *stack* of states that we need to explore. In the beginning, the only state in our stack is the initial state. In each step of the algorithm we remove the state at the top of our stack and check if it is the goal state. If not, we find all its neighbors, check which of them have already been examined, and add all un-examined neighbor states to the bottom of our stack. Then we simply repeat the process: Remove the state at the top of the stack, check if it is the goal state, and if not add all its unexamined neighbors to the stack. The algorithm ends either when we have found the goal state, or if the stack becomes empty, which means that we have explored all possible paths without reaching the goal, which proves that the problem cannot be solved. The algorithm can be described by the following Python code.

Definition 4.3 Breadth-first search

```
def bf_search(links, start, goal):
    """Given a dict where keys are states,
    and values are lists of connected states,
    finds a path from start to goal using
    breadth-first search"""
    state_list = [start]
    visited = [start]
    parent = {}
    while state_list:
        state = state_list[0]
        if state==goal:
            return parent
        state_list.remove(state)
        for neighbor in links[state]:
            if not neighbor in visited:
                visited.append(neighbor)
                state_list.append(neighbor)
                parent[neighbor] = state
```

4.3.2 *Breadth-first and depth-first search*

In our description of our search algorithm, at each step we removed the state at the top of the stack, found its non-visited neighbors and added them to the bottom of the stack. This means that for examples neighbors two levels away from the initial node will only be examined after all neighbors one level from the initial node have been considered. Thus, the algorithm expands in a *breadth-first* manner. Alternatively, we could have designed the algorithm to explore in a *depth-first* manner, simply by taking the next node to examine from the bottom of the stack rather than the top.

Example 4.7 Missionaries and cannibals

Three missionaries and three cannibals are on the left side of a river with a boat. The boat can only hold one or two people at once, and the goal is for all six people to cross the river to the right side. However, at no time must the missionaries be outnumbered by the cannibals at either side of the river. This type of problem can be solved using search, and to do so we need to define a representation of the states as well as the possible actions in each state.

State representation We can define the state by three numbers (m, c, b) that represent the number of missionaries (m) and cannibals (c) on the left side, and whether or not the boat (b) is on the left side. The initial state is $(3, 3, 1)$ and the goal state is $(0, 0, 0)$.

Actions The five possible state transitions are to transport one or two missionaries, one or two cannibals, or one of each to the other side. If the boat is on the left side, these transitions correspond to subtracting the following values from the current state:

$$(1,0,1) \quad (2,0,1) \quad (0,1,1) \quad (0,2,1) \quad (1,1,1)$$

Each of these transitions is possible only if it does not render m or c negative: For example it not possible to transport two cannibals, if there is only one left.

If the boat is on the right side, the situation is similar and the possible transitions correspond to adding the values above to the current state. Here, each transition is possible if it does not render m or c greater than 3.

Based on this, we can solve the problem by searching the state space for a solution. The full state diagram for the problem is shown in Figure 4.3.

Here is a question to think about: How would breadth-first and depth-first search solve this problem? Consider in which sequence the two methods will visit the states.

4.3.3 *Monte Carlo search*

In many practical problems the state space can be so large that it is not feasible to search through all possible action sequences. But still, if a problem can be represented as a graph search, it might be possible to use a suitable approximation to look for a solution.

One such approximative method is Monte Carlo[1] search. Rather than explore all possible state paths, the idea is to explore a smaller number of sample paths by choosing a random action at each step. These K randomly selected paths can either be explored all the way to the end (for example until we reach a dead end) or they can be explored only up to som finite

[1] Monte Carlo methods are a broad class of algorithms that rely on random sampling. These methods are named after the famous casino in Monte Carlo, Monaco.

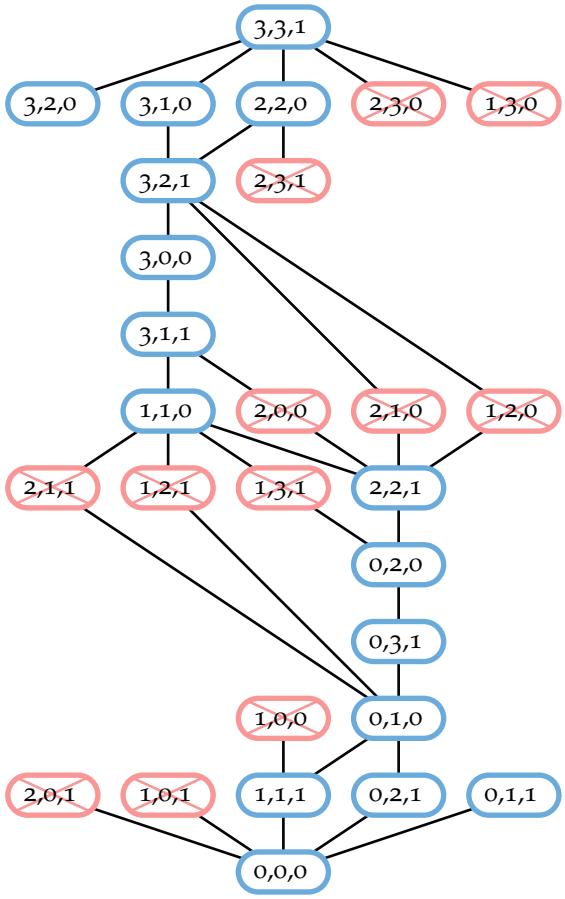


Figure 4.3: State diagram for the missionaries and cannibals problem. Each state is denoted by three numbers (m, c, b) denoting the number of missionaries m, cannibals c, and the position of the boat b where 0 and 1 indicate the left and right side of the river. Crossed-out boxes are invalid states where the cannibals outnumber the missionaries on one side of the river.

maximum number of steps.

Once we have explored K random paths, we need a mechanism to score them to measure which paths are most promising. If the problem we are trying to solve has a well-defined cost or reward associated with the states, we can use that to score the explored paths. Otherwise, we might score the paths according to their length, under the assumption that reaching far-away areas of the state space is desirable. Deciding upon a reasonable way to score the paths depends on the nature of the problem, and in some cases it is obvious what to do whereas in other cases it might be more difficult.

Once we have scored the random paths we have explored, we take the *first action* that leads to the highest score *on average*. This will move us one single step in the state diagram. Since we only have a small random sample of paths, we cannot trust that any of them are optimal, so we do not want to follow any of them in their full length. But taking just a single step chosen from the best performing set of paths is reasonable. After moving to the new state, the Monte Carlo search continues by exploring K new random paths from the new initial state. The algorithm continues in this manner until we reach a goal state or get stuck in a dead end.

Example 4.8 Shut the Box

The dice game *Shut the Box* (see Fig. 4.4) consists of nine wooden boxes numbered 1–9. The player rolls two dice and is allowed to close any combination of boxes such that their sum equals the sum of the dice. For example, if the player rolls 1+5=6 she can choose to close box 6, 1+5, 2+4, or 1+2+3. Once a box is closed it remains shut for the rest of the game. If the the boxes 7+8+9 are all closed, the player rolls only a single die. The game ends when all boxes are closed or if there is no combination of open boxes that match the sum of the dice. The final score is the sum of the open boxes at the end of the game, and the objective is to get as low a score as possible. A perfect game ends with a score of zero.

Figure 4.4: The dice game *Shut the Box*.

If the game is played completely at random, the average score is around 20 and a perfect game occurs with approximatly 2% chance. How do I know that? I wrote a small computer program that simulates the game and used it to

play 100 000 random games. But we can do much better than random with Monte Carlo search.

Consider a state in the game where the boxes 1+4+6+7+9 are open and we roll 5+2=7. We now have two possible moves: close box 1+6 or close box 7.

- First, we consider closing box 1+6 leaving 4+7+9 open. We now play, say, 1 000 random games from that position and record the final scores. We then compute the average score over the random games. (In a simulation I got an average score of 16.2.)
- Next, we consider closing box 7 leaving 1+4+6+9 open. Again we play 1 000 random games and compute the average score. (In a simulation I got 12.5.)

Based on this, we choose the second of the two possible moves, and close box 7.

We can now repeat the process. We throw the dice, and say we get 5+5=10. Again it happens that we have two options, to close 1+9 or 4+6. To decide which to go with, we repeat the Monte Carlo simulation as above. If we play the game this way using Monte Carlo search, the average score is around 11 and a perfect game occurs with approximately 6% chance.

As opposed to the search algorithms discussed previously, Monte Carlo search does not come with any guarantees. If a goal state can be reached Monte Carlo search might not find it, and if Monte Carlo search terminates without finding a goal state it does not mean that there is no solution.

Problems

1. Write down the truth table for the Boolean function $f(a, b) = (a \cdot \bar{b}) + (\bar{a} \cdot \bar{b})$.

2. Simplify the expression $f(a, b) = (a \cdot \bar{b}) + (\bar{a} \cdot \bar{b})$ as much as you can.

3. With n variables, how many different Boolean functions are possible? Self-check: With a single input, $n = 1$, there are four possible functions (can you list them?) and with 5 inputs there are more than 4 billion.

4. Which expert rule should you add to the three rules in Example 4.3 to make the system behave exactly as the Boolean function in Example 4.2, including the two undefined cases?

5. With the expert system in Example 4.5 and the initial facts *fuel_tank* is not_empty, *battery* is not_flat, *sun* is shining, and *birds* is singing, can you prove the statement *action* is walk using backward chaining? Which rules do you examine?

6. How can you modify the breadth-first search algorithm in Definition 4.3 so that it becomes a depth-first search algorithm?

7. Fig. 4.5 shows a state diagram with 16 states numbered 0 to 15 and possible transitions marked with arrows. If you want to search for a path from state 0 to state 15 using breadth-first and depth-first search search, which paths would be discovered? Assume that last seen states are visited first in the search.

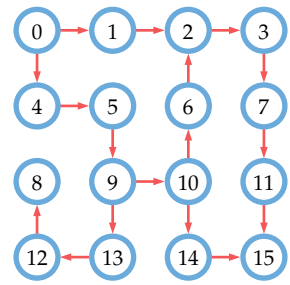


Figure 4.5: State diagram for a system with 16 states.

Solutions

1. The truth table for $f(a,b) = (a \cdot \bar{b}) + (\bar{a} \cdot \bar{b})$ is given by

a	b	$f(a,b)$
0	0	1
0	1	0
1	0	1
1	1	0

2. $f(a,b) = \bar{b}$.

3. There are 2^{2^n} possible Boolean functions with n inputs.

4. We know from Example 4.3 that the individual will not walk in the two undefined cases, so we might add the following rule which targets those specific cases:

   if *surveillance* is police and *road* is none and *light* is not green
   then *walk* is false
   Alternatively, we can rely on conflict resolution and add a simple rule at the top of the list (with lowest priority)

   if *surveillance* is police
   then *walk* is false

5. Yes, we can prove the statement. First we examine rule 2, which has the condition *car* is dead or *weather* is nice. These statements are possible consequents of rule 3 and rule 4 respectively, so these need to be examined. The condition of rule 3 is not true according to our initial facts, but the condition of rule 4 is true. This, in turn, makes rule 2 fire, which proves the statement.

6. We can do this by changing the line `node = node_list[0]` to `node = node_list[-1]` so that the search algorithm explores the node at the end of the stack rather than at the top of the stack.

7. For breadth-first search, we start at state 0. Let us keep track of the list of states we consider—to begin with, we just have state 0, so we write [0]. Now we explore state 0, which has two options, 1 and 4. We thus remove state 0 and append state 1 and 4 to the list, [0]→[1,4]. Now we explore the first state on the list, state 1, which

can only lead to state 2. Thus we remove state 1 and append state 2, [1,4]→[4,2]. Continuing this process we get [0]→[1,4]→[4,2]→[2,5]→[5,3]→[3,9]→[9,7]→[7,10,13]→[10,13,11]→[13,11,14,6]→[11,14,6,12]→[14,6,12,15], and we are done. Thus the search found the path 0→1→2→3→7→11→15.

For depth-first search, we get the sequence [0]→[1,5]→[1,9]→[1,10,13]→[1,10,12]→[1,10,8]→[1,10]→[1,6,14]→[1,6,15] and we are done. Thus the search found the path 0→4→5→9→10→14→15.

5 Data representations

Most artificial intelligence systems operate with some type of data. For example, systems might be designed to understand, act upon, or generate sound, images, or text. Exactly how the data is represented and processed can have a large influence on how the system can interact with the data, so it is important to understand and think carefully about data representation.

5.1 Data representations in the computer

In a computer all data is represented in a binary form, as a sequence of zeros and ones. When an analogue signal from the real world (such as a sound or an image) is captured by a sensor, it is transformed into a binary representation before it goes into the computer.

5.1.1 Numbers

A *bit* is a single value, zero or one, in the computer. Usually multiple bits are put together to represent a number. Most computers can work with two fundamentally different types of numbers:

Integer An integer is a whole number. If we use 8 bit to represent an integer, we can make $2^8 = 256$ different numbers. The bit sequence 0000 0000 represents the number 0, 0000 0001 represents 1, and 0000 0010 represents 2 and so on. An integer can either be defined as signed or unsigned: An unsigned 8-bit integer can represent any value between 0 and 255, whereas a signed 8-bit integer can represent values between -128 and 127. If we need to represent larger values, we can use a 16-bit integer (or 32 bits or 64 bits etc.)

Floating point A floating point number is a way to approximately represent real numbers. The numbers that can be represented are on the form (significand $\times$ 2^{exponent}) where the significand and exponent are signed integers. In a typical 64-bit floating point number, the significand is represented by 53 bits and the exponent has 11 bits. This means that numbers have around 16 decimal digits of precision (because $2^{53} \approx 10^{16}$) and that the largest number that can be represented is around 10^{308} (because $2^{2^{10}} \approx 10^{308}$).

5.1.2 *Characters*

Characters (letters and symbols) are represented as integers in the computer. Each possible character is given a unique number, and the list of which number corresponds to which character is called a *code page*. One of the classical code pages is ASCII, which is a 7-bit code designed to represent the 26 letters a–z, the capital letters A–Z, the numbers 0–9, a number of special graphical symbols, as well as some control characters such as a line shift character etc. For example, in the ASCII code page the letter 'a' has the integer value 97.

The ASCII code page cannot represent many of the special characters and symbols used in different languages, so many extensions have been developed and standardized. Many of the modern code pages remain compatible with the ASCII table. *Unicode* is a modern standard for character encoding that can represent most of the characters needed in all of the World's writing systems. The Unicode standard contains different encoding formats, one of which is UTF-8. This is a variable width character encoding, where most common characters are encoded by 8 bits, whereas more rarely used characters require up to 32 bits.

5.2 Measurement

In order to operate computationally or mathematically on data, we need to carefully consider how we measure. Roughly speaking, *measurement* is the process of assigning numbers to observations. To determine how to do this best, requires both knowledge about the data (what it means, how it is recorded, etc.) and how it is to be processed subsequently.

5.2.1 *Quantitative and qualitative data*

One way to characterize data is to consider whether it is *qualitative* or *quantitative*.

Qualitative Data that is non-numerical and recorded in free form is called *qualitative*. It is often gathered to gain insight in some phenomenon and answers questions such as "why?", "how?", and "what?".

Quantitative Data that is numerical data and recorded in a standardized way is called *quantitative*. It is often gathered to explain, predict, or control some phenomenon and answers questions such as "how many?", "how much?", and "how often?".

Example 5.1 A picture of some food items

Let us consider the photo in Fig. 5.1. An example of *qualitative* data describing the picture could be:

> A brown cupcake with the American flag on a plate, and a half-full glass bottle of Coke.

An example of *quantitative* data describing the picture could be:

Number of cupcakes	1
Color of cupcakes	brown
Number of bottles	1
Amount of Coke	17 [cl]

Figure 5.1: A picture of some food items.

Figure 5.2: A picture of some bottles.

5.2.2 *Types of quantitative data*

Quantitative data can be further characterized as either *numeric* or *categorical* data.

Numeric Data that is represented by numbers is called *numeric*. Numeric data can further be characterized as discrete or continuous.

Discrete data can only take certain values, and includes things we can count or assign to a category.

Continuous data can take any numerical value (within a range), and includes things we can measure in a continuum.

Categorical Data that puts each observation into one of a set of mutually exclusive groups is called *categorical*. Categorical data can be further characterized as either nominal or ordinal.

Ordinal categories can be logically arranged in a sequence.

Nominal categories serve only to identify or label each group, and there is no further structure in the set of categories.

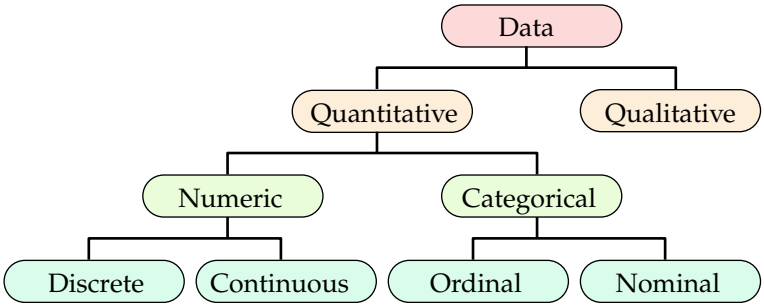


Figure 5.3: Categorization of different data types.

Fig. 5.3 outlines the different types of data discussed above. [1]

[1] In the research literature, there are also more advanced and detailed ways to characterize different data types with "Stevens's typology" perhaps the best known classification scheme (also known as a *level of measurement* or a *scale of measure*).

Example 5.2 Examples of types of quantitative data

Numeric-Discrete

- Number of students in a class.
- The outcome of rolling a die.

Numeric-Continuous

- Height of a person.
- Time of a 100 meters race.

Categorical-Ordinal

- Ratings on a scale (e.g. Good, Average, Bad).
- Letter grades (e.g. A, B, C, D, F)

Categorical-Nominal

- Brand of car (e.g. Seat, VW, Toyota, etc.)
- Marital status (e.g. Married, Single, Widowed).

5.3 Text representation

In the computer, a text string is simply a sequence of characters. In many applications, the natural level of analysis is the level of *words*, so often we think of text as a sequence of words. Just like characters can be represented using a code page where each character is assigned a unique number, *words* can also be represented in a similar manner. The total number of different words that exist is large but finite, so in theory one could simply use a huge dictionary of possible words to encode each word as a number. However, one would quickly run into problems with rare words, such as uncommon names, names of businesses, and made-up words that might be used in some literature. Nevertheless, this approach is often used in practise.

5.3.1 Bag-of-words

The *bag-of-words* representation is a very simple way to represent a text document. We simply count how many times each possible word occurs in the document, and store only the counts, ignoring the order in which the words occur.

Some words carry more information than others: For example, common words such as "the" and "and" might occur in most documents, but are not so important. In many practical representations, such common words are simply ignored.

Another issue is that many words essentially carry the same information: For example, different spellings or grammatical tenses of the same word are often semantically interchangable. To handle this issue, such words can simply be combined so that for example all words starting with *reali* are combined which would cover the words *realise*, *realize*, *realised*, *realization*, etc. This methods is known as *stemming* as it reduces each word to its stem.

Example 5.3 Document retrieval

One of the important tasks in natural language processing is information retrieval, where one wants to find a docu-

ments that match some search query. For example, this is the task that Internet search engines solve when we search for web pages that macth a search string.

With the bag-of-words representation we could find the documents that are relevant for the query simply by counting how many words the document and the query have in common. However, a simple count of the number of co-occuring words might not be optimal if the documents have very different lengths, because long documents would always tend to have more words and thus a higher chance of matching the query. This issue could be handled by normalizing the counts by the document length.

5.3.2 *n-grams*

While the bag-of-words representation is attractive because of its simplicity, it might be too crude to completely ignore the order in which the words occur in the text. One way to keep the representation simple while not completely ignoring the word order is to use an *n-gram* representation: Here we count the number of occurences of each possible length n sequence. For example, in a a bi-gram (2-gram) representation we would count how many times each possible pair of words occur in sequence.

Example 5.4

Bag-of-words and bi-gram representations Let us consider the following small piece of text:

> To be or not to be—that is the question.

In a bag-of-words representation, this would be represented as follows (in no particular order, but here alphabetized)

> be, is, not, or, question, that, the, to

Whereas in a bigram representation, we have the following

> be or, be that, is the, not to, or not, that is, the question, to be

Problems

1. Descibe the picture in Fig, 5.2 qualitatively and quantitatively.

2. Consider the following bag of words

   am, and, can, check, commencing, control, earth, eight, far, five, for, four, god, ground, here, i, liftoff, major, moon, nine, now, one, planet, seven, six, take, tell, ten, this, though, three, tom, two, you, your, a, above, am, and, be, blue, can, capsule, circuit, countdown, dare, dead, different, do, door, engines, feeling, floating, go, grade, hear, helmet, her, here, hundred, if, ignition, in, is, it, know, knows, leave, look, love, made, may, me, miles, most, much, my, nothing, on, one, papers, past, peculiar, pills, protein, put, really, round, she, shirts, sitting, something, spaceship, stars, stepping, still, the, there, think, thousand, through, time, tin, to, today, very, want, way, wear, which, whose, wife, with, world, wrong, you, your

   What do you think the original text is about?

Solutions

1. Example of qualitative description:

   > Four full glass bottles of Coke.

   Example of quantitative description:

Number of cupcakes	0
Color of cupcakes	*not applicable*
Number of bottles	4
Amount of Coke	132 [cl]

2. It is the lyrics from the David Bowie song *Space oddity* which is about an astronaut who is cut off communication with earth and floats into space.

6 Machine learning

The term *machine learning* is used to describe algorithms that can solve a problem or task by *learning from data* as opposed to being programmed explicitly for the job. In order for a machine learning problem to be well specified, we must determine the following three elements:

Task Which task to solve

Data Which data to use

Loss How to measure how well we solve the task

Once these elements have been specified, the machine learning problem is well defined. The next step is to create an algorithm that uses the data to solve the task in a way that minimizes the loss.

> Example 6.1 Predicting a person's weight from their height
>
> Let us say, that we want to create a machine learning system for the *task* of predicting how much a person weighs based on a measurement of their height. The *data* we would need to gather are measurements of height and weight for a representative sample of people. Finally we could for example define our *loss* as the absolute difference between our estimated weight and the true weight.
>
> Once we have made this decisions, the machine learning problem is completely specified. This means that we can now design and compare different methods.

6.1 Types of learning problems

We can distinguish between different types of learning problems, including supervised, unsupervised, and reinforcement

learning.

Definition 6.1 Supervised, unsupervised, and reinforcement learning

Supervised	Learning a mapping from features (input) to a target (output).
Unsupervised	Learning about patterns and structure in a data set.
Reinforcement	Learning to take actions in an environment.

Although not all learning problems can be neatly classified into one of these three categories, it is a useful way to categorize many different problems and algorithms. In the following, we will go through each type of learning problem and mention some of the most important methods within each category, also outlined in Figure 6.1.

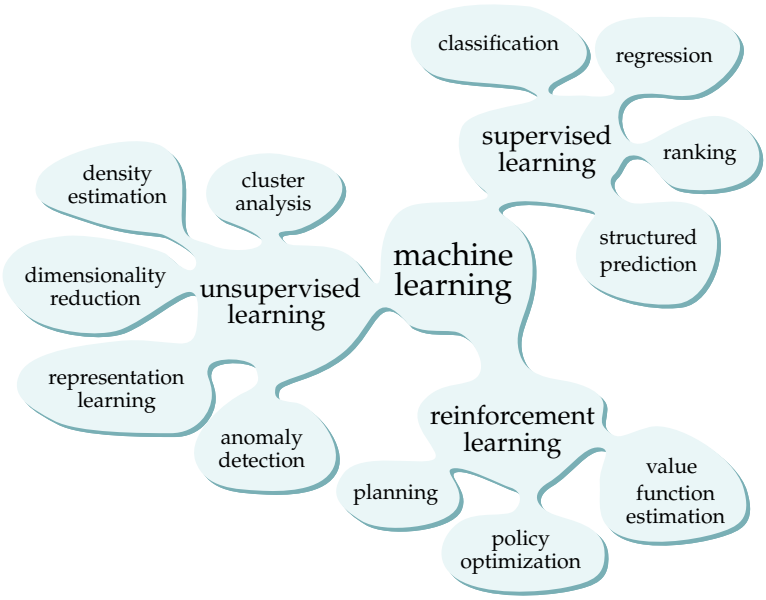


Figure 6.1: Mind map of different learning problems within machine learning.

6.1.1 *Supervised learning*

In supervised learning our data is a set of features x and targets y. The goal is to learn a mapping between x and y in the form of a function from the features to the target, $y \approx f(x)$.

The training data $\mathcal{D}$ often comes in the form of N *exchangeable* pairs of x- and y-values:

$$\mathcal{D}_{\text{s}} = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_N, y_N)\}.$$

By exchangeable, we mean that we do not care about the order in which the data occur in our collection. In other words, the numbering of the observations is arbitrary. All we care about is that we have N examples of an x and a y that go together.

The features x can be almost anything, including continuous or discrete numbers, vectors, strings, etc. Each target y is usually single continuous or discrete value.

Based on the training data, we want to optimally determine a function $f(x)$, so that when we get a new observation x^* we can simply map it through our function to obtain our prediction y^*,

$$y^* = f(x^*).$$

Regression In the case where the target y that we want to predict is a *continuous value* the learning problem is called *regression*. Some examples of regression could be to predict a person's weight, the fuel consumption of a vehicle, or the price of a stock.

Classification When the quantity y that we want to predict takes a value in a discrete set, the learning problem is called *classification*. Examples of classification include predicting whether or not it will rain tomorrow, determining the value 0–9 of a hand-written digit, or identifying an object in a photograph from a fixed set of possibilities.

Structured prediction When the target y is not a scalar, but a more complicated, structured object, the supervised learning problem is called *structured prediction*.

Ranking The supervising target could also be in the form of an ordering of the observations. In that case we could be interested in learning a function that can rank our observations, so that if we come with a new observation or a new set of observations, they can be ranked in the context of the existing data set or be ranked in similar manner as the existing data.

6.1.2 *Unsupervised learning*

In unsupervised learning our observed data is a set of features x. As opposed to the supervised setting, here we have no

targets y to predict. Rather, the goal is to learn a description of the patterns and structure in the data observations.

The data $\mathcal{D}_{\text{u}}$ often comes in the form of N exchangeable observations:

$$\mathcal{D}_{\text{u}} = \{x_1, x_2, x_3, \ldots, x_N\}.$$

As before, these observations could be in the form of numbers, vectors, strings, or anything else imaginable.

Cluster analysis One way to characterize structure in such a data set is to divide the observations into different groups in such a way that observations that are similar go in the same group. For example, we might have a data set where each observation corresponds to a customer in a supermarket and represents the customer's purchase habits. A cluster analysis could then be used to identify groups of customers with similar behavior, which then could be targeted more precisely in a marketing campaign.

Anomaly detection Sometime our goal is to detect observations which fall outside the typical pattern. For example, a bank might be interested in analyzing the patterns of its customers' credit card transactions to find unexpected events. Transactions carried out in a unusual currency, at an unusual time of day, or with an unusual amount might be an indication of fraud, which could be looked further into. This is similar to cluster analysis, where we group observations into just two categories: normal and anomalous.

Dimensionality reduction The goal of dimensionality reduction is to represent each of the observations in a compact form such that we loose as little information about the observations as possible. We could learn a (low-dimensional) vector representation z_n for each observation x_n so that $x_n \approx f(z_n)$ for some fixed function $f(z)$. In a way, cluster analysis can be seen as a special case of dimensionality reduction, where each observations is represented only by which cluster it belongs to.

Representation learning As opposed to dimensionality reduction, where we learn a compact representation optimized for reconstructing our data, in *representation learning* we learn a representation that is optimized for some secondary supervised task. In a sense, representation learning is supervised,

but the focus is to learn a useful representation of the data and not to do well on the supervised task. The idea is that a representaton that is useful in some taks might also be useful in other related tasks, and that learning a representation can help transfer knowledge to other domains.

Density estimation Fitting a probability density to a data set can also be seen as an unsupervised learning task. The probability distribution could possibly be very complicated with lots of parameters to fit, so while it might seem like a simple problem to fit a distribution to data, it might actually be a very difficult learning problem. Once the data has been fitted, the probability distribution can be used for many purposes: For example, we can simulate new fake data from the distribution, or we can use it to measure how likely some new observation is to see if it fits in with the remaining data.

6.1.3 Reinforcement learning

In reinforcement learning we have a dynamic setting where our machine learning system acts as an agent in an environment. The data is a sequence of environment states s, actions a, and rewards r,

$$\mathcal{D}_{\mathrm{r}} = (s_1, a_1, r_1), (s_2, a_2, r_2), \ldots.$$

At first we observe the initial state s_1, and then we must decide which action a_1 to take, after which we observe the associated reward r_1 and the new state s_2. Now we must decide which action a_2 to take, after which we observed the new reward r_2 and the new state s_3 and so on. Both the states and the actions can be in the form of continuous or discrete numbers or vectors or anything else imaginable, whereas the reward is typically a scalar quantity. The objective is to take actions to maximize the expected cumulative reward, which is the reward our agent will gather in the long run.

There is an inherent conflict between *exploration* and *exploitation* in reinforcement learning. After having observed a number of state-action-reward triplets, we might be able to come up with a reasonable policy that we can exploit to give a high reward. But maybe it would actually be better to take actions that giver a lower reward in the short term, but explores a larger area of the state space: Maybe that leads us to discover a different policy with an even better long term reward.

Policy optimization The most general reinforcement problem is to learn an optimal policy. A policy is a function $\pi(s) \to a$ that maps a state to an action. Since we can only learn about the system by interacting with it, learning an optimal policy is an iterative process which explores the system, learns about it, and eventually exploits what it has learned.

Value estimation If we knew the *value* (i.e. the expected reward) of each environmental state, an optimal policy would simply be to take the action which leads to the most valuable state. Some reinforcement learning algorithms focus on learning a value function $v(s)$ or a state-action value function $q(s, a)$ which then implicitly defines the optimal policy.

Planning In the general case, the only way to learn something about the environment is to interact with it. However, in some cases we have access to a simulation of the environment that allows us to run experiments that explore the state space. The problem of learning a policy (or value function) based on such a simulation model is known as *planning*.

6.2 Learning as optimization

Once we have a well defined machine learning problem with a given task, data, and loss, we can view what remains as an *optimization* problem: We need to find a solution that solves the task by minimizing the loss measured on the observed data.

We note that the optimization view is a very clear and direct way to think about machine learning, it is not the only way. Not all machine learning methods can be viewed as optimization problems, but some can better be understood from a statistical inference view or from a purely algorithmic view etc. However, in the following we will describe the learning problem from an optimization perspective.

6.2.1 Supervised learning as optimization

In supervised learning, where we want to find a function that maps from features to targets, we can set up a loss function that measures how well our function fits the true targets. We can then optimize (minimize) that loss to give us the best fitting function. In practice, the functions we consider is often a parametric family: In other words, we consider all functions with

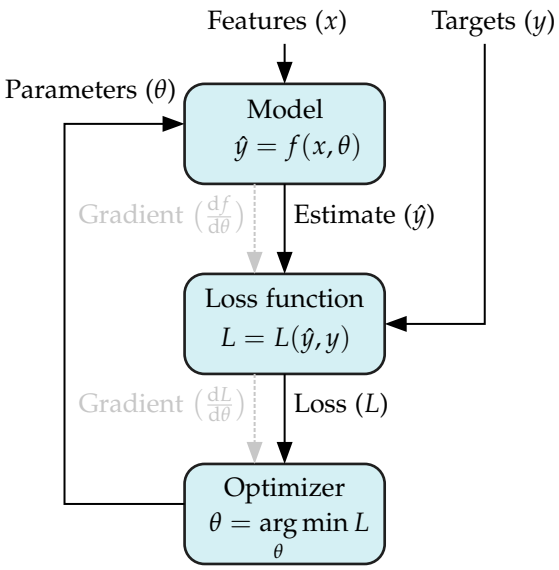


Figure 6.2: Supervised learning flow diagram. Input to the learning algorithm are features x and targets y. The goal is to learn the parameters θ of a function $f_\theta(x)$ that maps from features to targets. A loss function $L(y, \hat{y})$ defines the loss of predicting $\hat{y}$ when the true target is y. The parameters are found by minimizing the loss. In some algorithms, the optimizer uses the gradient of the loss to adjust the parameters.

a particular mathematical formulation with some unknown parameters. Then the problem becomes that of selecting the optimal value for those parameters. A flow chart of supervised learning from the perspective of optimization is given in Figure 6.2.

Example 6.2 Projectile motion

Let us say that we have a cannon that we can fire at different angles on a flat field (illustrated in Figure 6.3). We assume that the projectile always leaves the cannon with the same initial velocity. From physics we know that the laws of motion say that the height and distance of the projectle as a function of time are given by

$$y(t) = v_0 \sin(\phi) t - \tfrac{1}{2} g t^2,$$
$$x(t) = v_0 \cos(\phi) t,$$

where v_0 is the initial veolcity, ϕ is the angle of the cannon and $g = 9.81$ is the gravitational acceleration. We can then solve for the range R (the distance where the projectile hits

the ground)

$$R = \frac{v_0^2 \sin(2\phi)}{g}$$

Now let us say that we have a data set that consists of measured ranges for different angles (the measurements do not exactly follow the theory because of measurement noise):

ϕ	10°	20°	30°	40°	50°	60°
R	87	127	173	193	212	156

We would like to use supervised learning to find a function that allows us to compute the range for any angle. Our data set is $\mathcal{D}_s = \{(\phi_1, R_1), (\phi_2, R_2), \ldots, (\phi_6, R_6)\}$. Assuming that the formula for the range above holds, we choose the parameterized family of functions given by

$$\hat{R}(\phi, v_0) = \frac{v_0^2 \sin(2\phi)}{g}.$$

As ϕ is our input feature, the only unknown parameter is the initial velocity v_0.

We can now define our loss function: We decide to use the mean absolute difference between the measured and modelled range:

$$L(R, \hat{R}) = \frac{1}{6} \sum_{i=1}^{6} |R_i - \hat{R}_i|$$

Finally, to select the optimal value of v_0 we can simply plot the loss function and locate its minimum. In Figure 6.3 we can see that the minimum is located around $\hat{v}_0 = 45$.

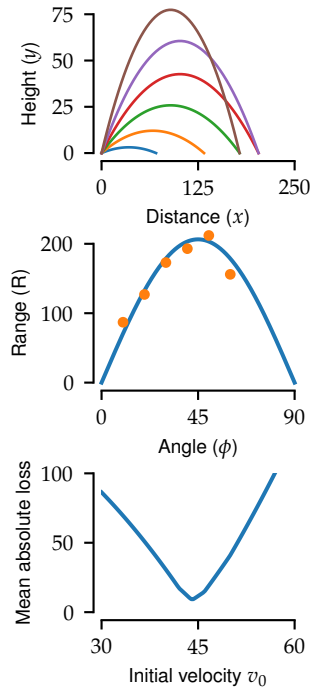


Figure 6.3: Projectile motion. Top: Six trajectories of projectiles fired with same initial velocity at different angles. Middle: The observed range of each of the six projectiles; solid line is the theoretical range as a function of the angle. Bottom: Mean absolute error between the model and the observations for different values of the parameter v_0.

6.2.2 *Unsupervised learning as optimization*

Many unsupervised learning methods can also be viewed as optimization problems. Here, the goal is to learn a representation of our data set. As before, we need to choose a loss function: Here, the loss measures how well our representation matches our data set. The representation depends on a set of parameters, which again we would like to optimize. Sometimes we have fixed number of parameters, that does not change with the

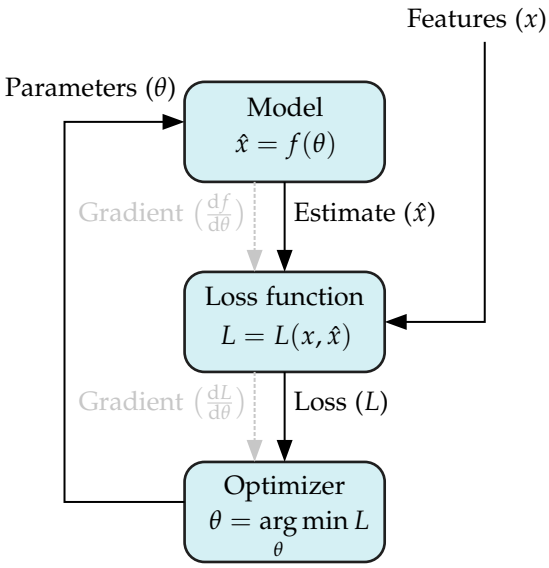


Figure 6.4: Unsupervised learning flow diagram. Input to the learning algorithm is a data set with features x. The goal is to learn a representation $f(\theta)$ of x by optimizing a set of parameters θ. A loss function measures the discrepancy between the representation $\hat{x}$ and the real data set x.

number of data points; however, very often we have specific parameters associated with each observation, such that the number of parameters increases with the size of the data. A flow chart of unsupervised learning form the perspective of optimization is given in Figure 6.4.

Example 6.3 Two cannons

Imagine we have measured the range of 10 shots from two different cannons, illustrated in Figure 6.5. We do not know how many shots were fired from each cannon, but we expect that one of the cannons is more powerful than the other. Based on this data, we would like to determine which shot came from which cannon. To do this, we take an unsupervised learning approach, where we assume that each cannon can be represented by a fixed range, and that deviations from this range is measurement noise. Thus, we can formulate the following model:

$$\hat{R}_i = \begin{cases} R_\mathrm{A} & \text{if shot } i \text{ came from cannon A} \\ R_\mathrm{B} & \text{if shot } i \text{ came from cannon B} \end{cases}$$

The parameters of our model are the two values R_A and

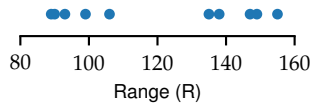


Figure 6.5: Measured ranges of ten cannon shots from two cannons with different power.

R_B as well as the assignment of each shot to one of the cannons. Mathematically, the assignment can be formulated as an indicator variable

$$c_i = \begin{cases} 0 & \text{if shot } i \text{ came from cannon A,} \\ 1 & \text{if shot } i \text{ came from cannon B.} \end{cases}$$

Based on this, we can write our model as

$$\hat{R}_i = (1 - c_i) \cdot R_A + c_i \cdot R_B$$

Next, we must formulate a loss function. Here, we will use the mean squared difference between the measured and hypothesized range:

$$L(R, \hat{R}) = \frac{1}{10} \sum_{i=1}^{10} (R_i - \hat{R}_i)^2.$$

Minimizing this loss is perhaps not so easy. With ten shots from two cannons, the number of different ways we can assign each shot to a cannon is $2^{10} = 1024$. We could try out each combination, and then find the optimal values for R_A and R_B for each, and finally select the values that gives the lowest loss.

Doing this, we find out that there are actually two solutions that minimize the loss. In the first solution, the five shortest range shots are assigned to cannon A, and $\hat{R}_A = 95.4$, $\hat{R}_B = 144.8$. In the other solution, the five shortest range shots are assigned to cannon B and the estimated ranges for the two cannons are the same as before but switched between the two cannons.

6.3 Evaluating performance

So far we have said that in order for a machine learning problem to be well defined, we must specify a loss function that tells us how well the task is solved, and which we can attempt to minimize to get the to the best solution. In supervised learning, the loss would be a measure of how well we can use the features to predict the targets. In unsupervised learning, the loss would be a measure of how well the data is represented. In reinforcement learning, the loss is inversely related to the cumulative reward.

However, in all these cases, the loss we should *really* minimize is not the loss on the particular data we have available, but rather the loss on the data we will see in the future when we put our machine learning system into action. It is not so interesing how well we can characterize the data that we already have available: Of course, if our method is a poor fit for our available data, we cannot expect it to perform well on future data. But even if our method fits our data well, we are not guaranteed to have good performance on future data. In fact, it is trivial to construct a method that perfectly fits any data set, simply my memorizing all the data. The challenge is to solve the problem in a way that will *generalize* to new data.

Example 6.4 Perfect fit?

Let us say we have a supervised learning task, where we want to find a function that maps from x to y, and say we are given the following observations:

x	0	1	2	3	4	5	6
y	True	False	True	False	True	False	True

Now I come up with a proposed solution:

$$f(x) = \begin{cases} \text{False} & \text{if } x \in \{1,3,5\} \\ \text{True} & \text{otherwise.} \end{cases}$$

In other words, my function simply outputs True if x is equal to 1, 3, or 5, and outputs False in all other cases. This function fits the observed data perfectly.

My optimistic colleague proposes the following solution, which fits only $\frac{4}{7} \approx 57\%$ of the observed data,

$$g(x) = \text{True}.$$

Clearly, my solution f is better than my colleague's g: My solution fits the observed data better, and both solutions will give the same result, True, for any other x not included in the set of observations.

Next, my student comes up with another proposal,

based on an observation about a pattern in the data,

$$h(x) = \begin{cases} \text{True} & \text{if } x \text{ is even} \\ \text{False} & \text{otherwise.} \end{cases}$$

Her solution h fits the observed data perfectly just like my solution: The only difference between f and h is what they predict for values of x outside the observed data.

Now which solution is best? We cannot really tell without any further information or assumptions. The point is, that what really matters is how well the solution will generalize to new, unseen data.

6.3.1 *Training and test*

What we need is a way to measure how well our methods work on future data. One of the most simple and popular ways to measure this is *cross validation*. There are many different cross validation schemes, but the most simple way is to divide the observed data set in two parts: One part will be used for *training* the machine learning model, and the other part will be used for *testing* the learned solution. This approach is called the *hold out* method. Since we only use the training data to fit our model, we can use the test data to measure how well the method will generalize: The performance of the test data is an estimate of the generalization error.

Using cross validation, we can then compare different models: A too simple model can not be expected to fit the training data very well, and it will also not generalize well to new data. A too complex model will fit the training data very well, maybe even perfectly, but it will also not generalize well to new data. Somewhere in between these extremes, we can hope to find a model with suitable complexity: One that not only fits the training data reasonably, but which also will perform well on data in the future (see Figure 6.6 for an illustration).

It can be very beneficial to think about machine learning as a type of *inferential statistics*. As you may recall, in inferential statistics we observe a sample from a population, and we the use sample statistics to estimate parameters of the population.

In machine learning, we can think of the training data as a sample from a population. When we fit a machine learning model we are not just interested in finding a model that per-

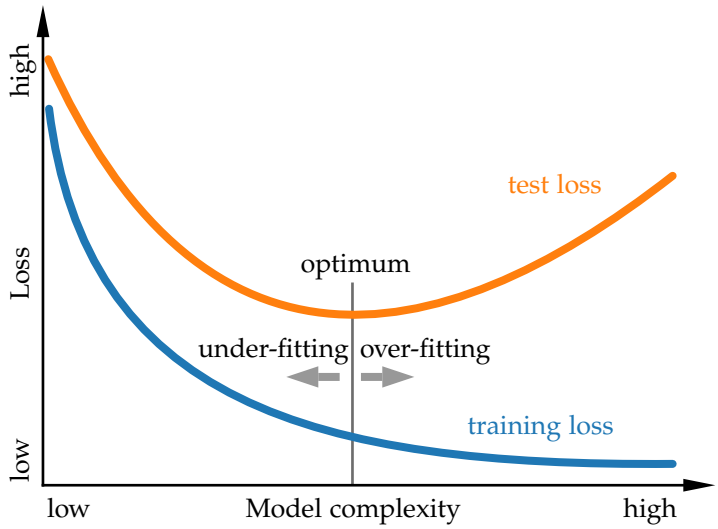


Figure 6.6: Overfitting and underfitting. A very simple model cannot be expected to fit the training data well, and does not have good generalization. This is known as *underfitting*. By increasing the complexity of the model, we can fit the training data arbitrarily well; however, while fitting the training data very well, a too complex model will also not have good generalization. This is known as *overfitting*. The optimal model complexity can be found somewhere between these extremes.

forms well on this particular training data sample. Ideally we would like a model that performs well on *any* sample from the population. A good way to measure how well our model's performance generalizes is to test it on a new, independent sample from the population, which is exactly the role of the test set.

6.4 Linear regression

One of the most simple and effective methods for supervised learning is linear least squares regression. The objective is to predict a continuous target variable y based on a set of observed features x. In linear regression, we use a linear function to map between the features and the target. If we have just a single scalar feature x, the linear regression formula can be written as

$$\hat{y} = f(x) = w_0 + w_1 \cdot x.$$

In other words, we fit the target variable with a straight line with intercept w_0 and slope w_1. In *least squares* regression, the loss function is the mean squared error between the true and the predicted target:

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

To fit the linear regression model, we choose the parameters w_0 and w_1 that minimize the loss. A nice property of the linear

model in combination with the squared error loss is that we can find the optimal parameters very efficiently by solving a system of equations. To do this, we can compute the derivative of the loss with respect to each of the parameters, set the derivative equal to zero, and solve for the parameters.

$$\begin{aligned}
\frac{\partial L}{\partial w_0} &= \frac{1}{N}\sum_{i=0}^{N} -2\big(y_i - (w_0 + w_1 \cdot x_i)\big) = 0 \\
\Rightarrow & w_0 + w_1 \cdot \frac{1}{N}\sum_{i=1}^{N} x_i = \frac{1}{N}\sum_{i=1}^{N} y_i \\
\frac{\partial L}{\partial w_1} &= \frac{1}{N}\sum_{i=0}^{N} -2\big(y_i - (w_0 + w_1 \cdot x_i)\big)x_i = 0 \\
\Rightarrow & w_0 \frac{1}{N}\sum_{i=1}^{N} x_i + w_1 \cdot \frac{1}{N}\sum_{i=1}^{N} x_i^2 = \frac{1}{N}\sum_{i=1}^{N} y_i x_i
\end{aligned}$$

Thus, we have two equations in two unknowns:

$$\begin{aligned}
a \cdot w_0 + b \cdot w_1 &= c \\
d \cdot w_0 + e \cdot w_1 &= f
\end{aligned} \tag{6.1}$$

where $a = 1$, $b = d = \frac{1}{N}\sum_{i=1}^{N} x_i$, $c = \frac{1}{N}\sum_{i=1}^{N} y_i$, $e = \frac{1}{N}\sum_{i=1}^{N} x_i^2$, and $f = \frac{1}{N}\sum_{i=1}^{N} y_i x_i$.

This means that we can find the optimal intercept and slope by solving two equations in two unknowns, where the coefficients of the equations are computed from the data.

Example 6.5 Linear regression

Consider the dataset with $N = 10$ observations shown in Figure 6.7. Let us say we would like to fit a linear regression model to predict y from x. First, let us see what we can say about the parameters just from visually inspecting the figure. Since we have

$$y = w_0 + w_1 \cdot x,$$

the intercept w_0 is equal to the y-value for $x = 0$. From the figure approximatly read off the value

$$w_0 \approx -2.$$

The slope, w_1, can be approximated by reading off two

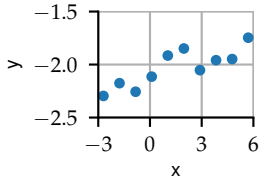


Figure 6.7: A simple linear regression example.

(x, y)-pairs from the grap: Since we have

$$y_A = w_0 + w_1 \cdot x_A, \qquad y_B = w_0 + w_1 \cdot x_B,$$

we may write

$$y_A - y_B = (w_0 + w_1 \cdot x_A) - (w_0 + w_1 \cdot x_B) = w_1(x_A - x_B),$$

and solve for the slope

$$w_1 = \frac{y_A - y_B}{x_A - x_B}.$$

From the figure we can imagine a straight line fit to the data and get two approximate (x, y)-pairs: $(x_A, y_A) \approx (-3, -2.3)$ and $(x_A, y_A) \approx (6, -1.7)$ which gives

$$w_1 \approx \frac{-2.3 - (-1.7)}{-3 - 6} \approx 0.07.$$

Next, let us use the data to compute the coefficients for the two equations in two unknowns from Equation 6.1. To do this, we of course need to have access to the actual data set, so let us just assume that we were able to find the following coefficients,

$$\begin{aligned} 1 \cdot w_0 + 1.50 \cdot w_1 &= -2.03 \\ 1.50 \cdot w_0 + 9.44 \cdot w_1 &= -2.66. \end{aligned}$$

Solving this system of equations gives us the exact solution,

$$w_0 \approx -2.11, \qquad w_1 \approx 0.054,$$

which is not too far of from our rough estimate.

Problems

1. In an image recognition system, what could the task, data, and loss be?

2. Classify the following problems as unsupervised, supervised, or reinforcement learning.

   (a) Learn to control a car by recording what human drives do and training a system to do the same.

   (b) Learn to control a helicopter by trial and error while encouraging it not to crash.

   (c) Learn how human cognitive ability can best be described by a single number based on a test questionnaire.

3. Say I want to predict how much fuel my car will use on a trip based on data from previous trips. I have data from 20 previous trips where I recorded the length of the trip, whether it was in the city, on the highway, or on the motorway, and the number of people in the car, along with a lot of other features related to the trips. Now I fit two least squares linear regression models to predict the fuel consumption: The first model uses only the length of the trip as an input feature and gives a mean squared error of 1.32 on the 20 trips used as training data. The second model uses all of my recorded features as input and gives a mean squared error of 0.68. Which of the two regression models is best?

4. Say we are given the following values for the fuel consumption and the trip length

Trip length (x)	10	15	12	40	13
Fuel consumption (y)	0.4	0.7	0.6	1.9	0.6

   (a) Plot the data

   (b) Using least squares linear regression, fit the following model

$$\hat{y} = f(x) = w \cdot x.$$

   Hint: Write down the loss function. Compute the derivative of the loss with respect to the parameter w. Set the derivative equal to zero, and solve for w.

Solutions

1. The task is to recognize an object in an image. The data is typically a large database of images with known object labels. The loss is related to the accuracy of the classifier: Most often classifiers output a probability for each class, and the loss is defined so that the probability of the correct class is maximized.

2. (a) Supervised learning. (b) Reinforcement learning. (c) Unsupervised learning.

3. We cannot tell from the performance on the training data: Even though the second model gives a lower training loss, it might not generalize to new data. We would need to use cross validation and test the models on a set of trips that are not used for training the model.

4.(a) See Figure 6.8

(b) $w \approx 0.047$.

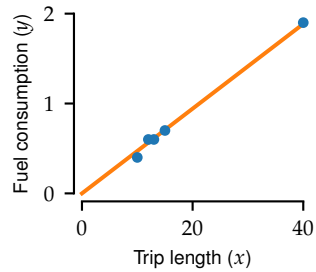


Figure 6.8: Plot of fuel consumption vsersus trip length

7 Features and clustering

7.1 Features

In macine learning, where we want learn from patterns and relations in data, it is obviously very important which data we base our models on. If the features we have available are not very informative about the problems we are addressing, we do not have much chance of solving them. "Garbage in, garbage out" as the proverb says. Thus, it is of great importance that we carefully consider which features to base our models on. If the features we have available are not enough, we should consider gathering more data.

Once we have decided which features to base our analysis on, we need to consider if we need to preprocess the features. Here we can distinguish between the following tasks:

Feature selection If we have a large number of potential features that we can consider, we must choose which to include in our model.

Feature transformation We must decide how to present the features to our learning algorithm. For example, we might consider scaling or normalizing the features, converting between numerical and categorical representations etc.

Feature synthesis We can also consider if it is relevant to create new features by combining some of our existing features. For example, if we have *length* and *width* as features, we might include their product as a new *area* feature.

Feature extraction Some features might be very complex such an image or an audio waveform. For these types of inputs

it could be relevant to extract a small number of features to characterize the raw data more compactly.

It is important to use our background knowledge about the problem to choose features that are likely to be informative. We can also carry out quantitative evaluation of different features or combinations of features to determine which features to include. Some machine learning methods can be somewhat sensitive and perform poorly if we include bad (noisy, uninformative) features, whereas other methods are more robust and learn to ignore the bad features: Thus, how we preprocess the features will also depend on which subseqeuent analysis we plan to conduct. The primary reason to consider various feature transformations is to make the learning problem easier: To learn faster and generalize better.

7.1.1 Feature scaling

A typical way to preprocess numerical features is to scale them so that their range is on a comparable scale.

Min-max normalization A simple way to scale a numerical feature is to map it onto the range [0,1]: If x is our original feature, we can create a new normalized feature x' by subtracting the minimum and dividing by the range,

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}.$$

Here, the maximum and minimum is taken over all observed values in the training set. This ensures that all observations in the training set will be in the range [0,1].

Standardization Another common approach is to rescale numerical features to have zero mean and unit standard deviation by subtracting the mean and dividing by the standard deviation.

$$x' = \frac{x - \mu_x}{\sigma_x}.$$

As before, the mean and standard deviation are typically computed on the training set.

Example 7.1 Standardized truck features

Say we have a data set that consists of information about different trucks: The features we have available are the weight of the truck in kilograms and the length of the truck in meters. We have the following data about five trucks:

Weight [kg] (w)	4906	5091	5192	5593	4775
Length [m] (ℓ)	5.0	4.2	7.9	1.6	5.9

We decide to standardize the features to put them on a comparable scale. For the weight feature we compute the mean and standard deviation

$$\mu_w = \frac{1}{5}\sum_{n=1}^{5} w_n \approx 5111$$

$$\sigma_w = \sqrt{\frac{1}{5}\sum_{n=1}^{5}(w_n - \mu_w)^2} \approx 281$$

Based on this we can compute the standardized weights as

$$w' = \frac{w - 5111}{281}.$$

Similarly for the length features, we subtract the mean and divide by the standard deviation. This gives us the following standardized features

Std. weight (w')	-0.732	-0.073	0.287	1.715	-1.198
Std. length (ℓ')	0.039	-0.348	1.441	-1.705	0.474

7.1.2 *Numerical and categorical features*

It can be important to distinguish between *numerical* and *categorical* features.

Numerical features Features that are numerical correspond to counts or measurements. For example, the number of weight or number of pages in a book are numerical features. Numerical data is also often referred to as *quantitative*.

Categorical features Other features represent a characteristic which is absent or present, and cannot be counted or measured. For example, whether a book is a novel, a biography, or a comic book is a categorical feature. Categorical data are also often referred to as *qualitative*.

Often categorical features are represented by whole numbers, for example novel=0, biography=1, comic=2, etc. It is important that we do not treat such features as numerical: The numbers used to represent categorical feature have no meaning whatsoever. When we have categorical features, we must be careful not to use them in a way that is not appropriate, as machine learning methods implicitly assume that the features are numerical. To avoid this situation, it can often be of practical use to transform a categorical feature into several new binary features which correspond to each of the possible categories. This is known as *one-hot-coding* or as creating a set of *dummy features*.

Example 7.2 One-hot-coding the book-type

Consider a feature *book-type* which can take the values 0, 1, or 2, which are interpreted as:

Book-type	Description
0	Novel
1	Biography
2	Comic

and let us say we have the following data set:

Title	Book-type
Gone with the wind	0
Long walk to freedom	1
Astonishing X-Men	2
Amazing Spider-Man	2
The catcher in the rye	0

We can then transform the book-type feature into three new binary dummy features, which indicate the type of book.

Title	Novel	Biography	Comic
Gone with the wind	1	0	0
Long walk to freedom	0	1	0
Astonishing X-Men	0	0	1
Amazing Spider-Man	0	0	1
The catcher in the rye	1	0	0

7.2 Clustering

In cluster analysis, the task is to divide the observations into a number of clusters in such a way that similar observations go in the same group. K-means clustering is a particular algorithm that can give a fast but approximate solution to the cluster analysis problem under certain assumptions which we will go through here.

7.2.1 Centroid-based methods

In the following we will assume that the data we have available is a set of N numerical feature vectors $\mathcal{D}_{\text{u}} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$. We assume that we know the number of clusters K in advance, and that that each cluster can be described by a single representative vector that resides in the same space as the features. These vectors are referred to as cluster centroids and we denote them by $\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_K$. The clustering problem consists of assigning each observation to a cluster while at the same time determining the optimal values for the centroids.

Let c_i denote the cluster to which the ith observation is assigned (for example $c_3 = 5$ if observation number 3 is assigned to cluster number 5.) To ensure that each observation is close to its cluster centroid, we can use the squared distance between the observation and its corresponding cluster centroid as a loss function.

$$L = \sum_{i=1}^{N} \|\boldsymbol{x}_i - \boldsymbol{z}_{c_i}\|^2.$$

If the observations and centroids are scalar, this would simply be the squared difference. In the case where observations and centroids are vectors, we use the squared Euclidean distance.

Now, the optimization problem consists of determining the cluster centroids as well as the cluster assignments. This is not a simple optimization problem, because we need to optimize two

things that are strongly dependent: The cluster assignments and the cluster centroids.

7.2.2 *The K-means algorithm*

The K-means algorithm is a fairly simple procedure that aims at finding optimal values in the centroid-based clustering problem. It is based on the idea of alternating between estimating the optimal cluster centroids for a given cluster assignment, and estimating the optimal cluster assignment for a given set of centroids. By going back and forth between estimating the two, after a while the algorithm will converge on a good (albeit not necessarily optimal) solution.

Definition 7.1 K-means clustering

K-means clustering alternates between the following two steps until convergence:

1. Estimate optimal centroids, based on current cluster assignment.
2. Estimate optimal cluster assignment, based on current centroids.

To get the algorithm going, we can start with a random cluster assignment or start with a random set of centroids.

First, we assume that the cluster assignments c_i are known and fixed. Now the problem of estimating the optimal centroids not too difficult: We can simply differentiate the loss with respect to the jth centroid and set it equal to zero to solve for each centroid. In the sum over the N data points in the loss, only some of the terms include the jth centroid $\boldsymbol{z}_j$. It can be useful to define the number of observations assigned to the jth cluster as N_j and define the set of observations in cluster j as $\{i : c_i = j\}$ (the set of all indices i for which the cluster assignment c_i is equal to j). With this notation, we can write sum over the N_j observations in cluster j as a sum over $\{i : c_i = j\}$. Thus, we get the following optimal cluster centroids

$$\frac{\partial L}{\partial \boldsymbol{z}_j} = -2 \sum_{i:c_i=j} (\boldsymbol{x}_i - \boldsymbol{z}_j) = 0 \Rightarrow \boldsymbol{z}_j = \frac{1}{N_j} \sum_{i:c_i=j} \boldsymbol{x}_i. \tag{7.1}$$

This means, that the least squares optimal centroid for a cluster is simply the average of the observations in the cluster.

Example 7.3 Computing centroids

Assume we have $N = 5$ observations

Observation (i)	1	2	3	4	5
Features ($\boldsymbol{x}_i$)	$\begin{bmatrix} 4 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 3 \end{bmatrix}$	$\begin{bmatrix} 2 \\ 4 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 5 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 4 \end{bmatrix}$
Cluster (c_i)	1	2	1	2	2

We can now compute the least squares optimal cluster centroids as

$$\boldsymbol{z}_1 = \frac{1}{2}\left(\begin{bmatrix} 4 \\ 2 \end{bmatrix} + \begin{bmatrix} 2 \\ 4 \end{bmatrix}\right) = \begin{bmatrix} 3 \\ 3 \end{bmatrix},$$
$$\boldsymbol{z}_2 = \frac{1}{3}\left(\begin{bmatrix} 1 \\ 3 \end{bmatrix} + \begin{bmatrix} 1 \\ 5 \end{bmatrix} + \begin{bmatrix} 1 \\ 4 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 4 \end{bmatrix}.$$

Next, we assume that the cluster centroids $\boldsymbol{z}_i$ are known and fixed. Now, the problem of estimating the optimal cluster assignment is easy: Since the loss function measures the sum of squared distances between the observations and the cluster centroids, the optimal solution is clearly to assign each observation to the closest centroid. Thus, the optimal cluster assignment can be found for each observation by computing the distance to each centroid and finding the minimum distance:

$$c_i = \arg\min_j \|\boldsymbol{x}_i - \boldsymbol{c}_j\|^2.$$

Example 7.4 K-means

Consider again the data set in Example 7.3. Now, we would like to use the K-means algorithm to cluster the five data observations. Say we initialize the algorithm with the following cluster assignment:

i	1	2	3	4	5
c_i	1	1	2	2	2

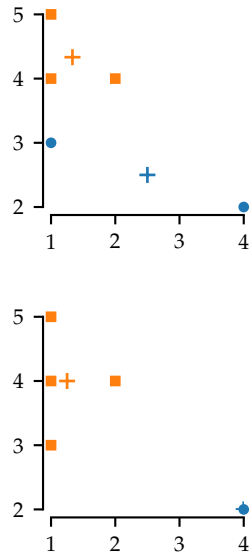


Figure 7.1: Example of K-means clustering of five observations (two-dimensional vectors). Cluster centroids are shown as + symbols and cluster assignment is indicated by color. The top panel shows the initialization, and the bottom panel shows the final result. In this example, the algorithm converges after just one update.

We can then compute the optimal cluster centroids as

$$z_1 = \frac{1}{2}\left(\begin{bmatrix} 4 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 3 \end{bmatrix}\right) = \begin{bmatrix} 2.5 \\ 2.5 \end{bmatrix},$$
$$z_2 = \frac{1}{3}\left(\begin{bmatrix} 2 \\ 4 \end{bmatrix} + \begin{bmatrix} 1 \\ 5 \end{bmatrix} + \begin{bmatrix} 1 \\ 4 \end{bmatrix}\right) \approx \begin{bmatrix} 1.33 \\ 4.33 \end{bmatrix}.$$

Now, we can compute the distances from each of the five observations to the two centroids

i	1	2	3	4	5
$\|x_i - z_1\|^2$	2.5	2.5	2.5	8.5	4.5
$\|x_i - z_2\|^2$	12.6	1.89	0.56	0.56	0.22

Based on these distances, we see that the new optimal cluster assignment is given by

i	1	2	3	4	5
c_i	1	2	2	2	2

We now update the cluster centroids:

$$z_1 = \begin{bmatrix} 4 \\ 2 \end{bmatrix},$$
$$z_2 = \frac{1}{4}\left(\begin{bmatrix} 1 \\ 3 \end{bmatrix} + \begin{bmatrix} 2 \\ 4 \end{bmatrix} + \begin{bmatrix} 1 \\ 5 \end{bmatrix} + \begin{bmatrix} 1 \\ 4 \end{bmatrix}\right) = \begin{bmatrix} 1.25 \\ 4 \end{bmatrix}.$$

Again we compute the distances from each of the five observations to the two centroids

i	1	2	3	4	5
$\|x_i - z_1\|^2$	0	10	8	18	13
$\|x_i - z_2\|^2$	11.56	1.06	0.56	1.06	0.06

Based on these distances, the optimal cluster assignment does not change and the algorithm has converged. The steps in the algorithm are illustrated in Figure 7.1.

Problems

1. We have a numerical feature x that takes values in the range -17 to 25. Using min-max normalization, how would a value of $x = 0$ be represented?

2. We have 10 observations of a numerical feature x that takes the following values $1, 2, 3, 4, 5, 6, 7, 8, 9, 10$. If we standardize this feature, how would the value $x = 5$ be represented?

3. In Example 7.2 we created three dummy variables *Novel*, *Biography*, and *Comic* to represent the categorical feature *Book-type*. If we decide to throw away the feature *Comic* (so we only have *Novel* and *Biography* left) would we loose any information?

4. What is the value of the loss function in Example 7.3?

Solutions

1. $x' = 0.4048$.
2. $x' \approx -0.174$
3. No, because the feature *Comic* can be inferred from the two other features (if *Novel* and *Biography* are both 0, then *Comic* must be 1). In fact, we can always represent a categorical variable with K categories using $K - 1$ dummy features.
4. $L = 6$

8 Gradient based optimization

8.1 Gradient descent

In calculus, the derivative of a function $y = f(x)$ is defined as a the limit,

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

For a differentiable function, the derivative tells us how much the function $y = f(x)$ will change in proportion to an infinitesimal change in x. Geometrically, the derivative is the slope of the tangent line of the function at the point x. The derivative of a simple function at different locations is visualized in figure 8.1.

In optimization problems, we are interested in finding an optimal value x for a given function $f(x)$. Depending on the problem formulation, the optimum can either be defined as a maximum or a minimum value. In the following, we will focus on the minimization problem, but we note that a maximization problem can easily be converted to a minimization problem, for example by taking the negative or inverse of the objective.

In some cases we can compute the derivative of our objective function and set it equal to zero to solve the minimization problem. However, in many practical cases the objective function is too complicated to allow us to easily solve the minimization problem analytically. In those cases, we can instead use a simple *gradient descent* method. Gradient descent is an interative procedure that finds a minimum of a function by taking steps in the x variable proportional to the negative of the derivative of the objective function.

Intuitively, if the gradient of a function $f(x)$ is positive at a

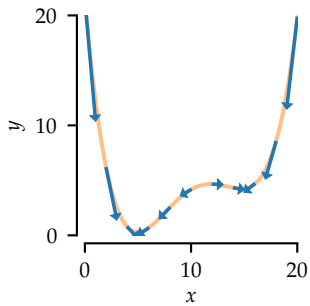


Figure 8.1: Visualization of the derivatives of a fourth order polynomial at different points. Here the derivative of y with respect to x is a scalar that tells us how much y will change for a given change in x. The derivative is shown for different values of x as an arrow from a point (x, y) to a point $(x + \Delta x, y + \Delta y)$ where $\frac{\Delta y}{\Delta x} \propto \frac{dy}{dx}$.

point x, then the function will have a positive slope. Thus, a minimum of the function must be located somewhere in the negative x-direction, and we subtract from x a value proportional to the derivative.

8.1.1 *Gradients of multivariable functions*

If the function we wish to minimize depends on several variables $f(x_1, x_2, \ldots, x_M)$ we need to compute the derivative with respect to each of the input variables. To simplify the notation, we can collect the input variables in a vector (of length M), which we will denote by $\mathbf{x} = [x_1, x_2, \ldots, x_M]^\top$.

We can compute the derivative of the multivariable function $f(\mathbf{x})$ with respect to any single one of its inputs (say x_m) and keep all other inputs fixed. This is known as the *partial derivative* and is most often denoted by $\frac{\partial f}{\partial x_m}$. Based on this, we define the *gradient* of the function as the vector containing all the partial derivatives with respect to each of the inputs,

$$\nabla f(\mathbf{x}) = \frac{df}{d\mathbf{x}} = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_M}\right].$$

Example 8.1 Gradient of a simple neural network

Consider a non-linear regression problem where we want to minimize the squared error between a chosen function $f(x)$ and our observations y. Say our function is given by

$$f(x) = \tanh(\theta_0 + \theta_1 x)$$

and the cost function, which is our objective for minimization, is given by

$$C(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\big(y_i - f(x_i)\big)^2.$$

Note that now x are our (fixed) inputs and $\boldsymbol{\theta}$ are the parameters of the function we wish to optimize. The partial derivatives with respect to the parameters θ_0 and θ_1 are

given by

$$\frac{\partial C}{\partial \theta_0} = -\frac{2}{N}\sum_{i=1}^{N}\big(y_i - f(x_i)\big)\big(1 - f^2(x_i)\big),$$
$$\frac{\partial C}{\partial \theta_1} = -\frac{2}{N}\sum_{i=1}^{N}\big(y_i - f(x_i)\big)\big(1 - f^2(x_i)\big)x_i,$$

where we have used the fact that $\frac{\mathrm{d}}{\mathrm{d}x}\tanh(x) = 1 - \tanh^2(x)$. Finally, the gradient is given as the vector containing these two partial derivatives,

$$\nabla C(\boldsymbol{\theta}) = \left[\frac{\partial C}{\partial \theta_0}, \frac{\partial C}{\partial \theta_1}\right].$$

8.1.2 *The gradient descent algorithm*

When applied to the minimization of a multivariable function, the gradient descent algorithms works by taking steps in the input argument $\mathbf{x}$ proportional to the negative of the gradient. The size of the steps is controlled by a parameter α.

Definition 8.1 Gradient descent

The method of gradient descent is used to find a minimum of a function $f(\mathbf{x})$ by iterating the following update until convergence.

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \cdot \nabla f(\mathbf{x}^{(t)})$$

Here, $\mathbf{x}^{(t)}$ denotes the value of $\mathbf{x}$ at iteration t, $\mathbf{x}^{(0)}$ is a suitably chosen initial value, and α is a step size parameter.

It is very important to choose the step-size parameter carefully. If the step-size is too large, the updates will overshoot and diverge. If, on the other hand, the step-size is too small, the updates will only make very small changes to $\mathbf{x}$ and the algorithm will take a very long time to converge. An example of the gradient descent algorithm is illustrated in Figure 8.2.

If the objective function has more than one minimum, the gradient descent algorithm not not necessarily converge to the *global* minimum (the value for which the function takes is minimum across its entire domain). Rather, we say that the algorithm finds a *local minimum*, and which one it finds depends on the initial conditions and the step size. Furthermore, if the

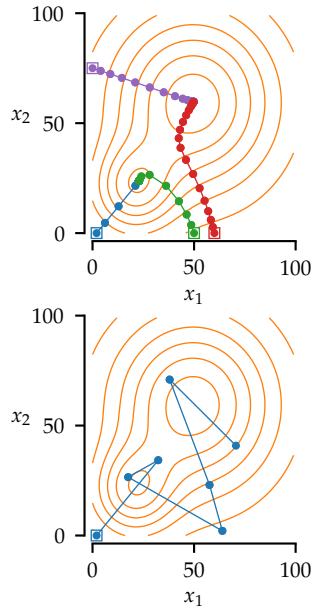


Figure 8.2: Gradient descent on a funtion in two variables. The objective function is illustrated with contour lines (levels sets where the function value is constant). The top plot shows the convergence of gradient descent with a suitably small step size. The algorithm is run from four different initial points marked with squares, and different local minima are found depending on initial conditions. The bottom plot shows six iterations of gradient descent with a fairly large step size.

objective function contains a *saddle-point* (a point where the gradient is zero which is not a local minimum) the gradient descent algorithm might also converge to this.

Under certain mild assumptions about the objective function, the gradient descent algorithm is guaranteed to converge if the step-size is small enough. Since the objective function will decrease if we take a sufficiently small step in the direction of the negative gradient, the sequence of function values we visit will be guaranteed to decrease. As we approach a minimum, the function will be more and more flat, and thus the magnitude of the gradient will decrease. This means that as we approach a minimum the algorithm will take smaller and smaller steps, eventually converging at the minimum.

8.2 Stochastic gradient descent

A disadvantage of the gradient descent algorithm is that it requires us to evaluate the gradient of the objective in each iteration. If we want to estimate parameters in a model based on a large dataset, we need to do computations on the whole data in each iteration, which might not be feasible.

Stochastic gradient descent (SGD) is a simple method that avoids this problem. Instead of computing the exact gradient, in SGD the gradient is estimated based on a small subset of the data. By using a different subset of the data in each iteration, we obtain a "noisy" estimate of the gradient, and while each update of the algorithm does not necessarily reduce the objective function, on average the updates will take us in the direction of a local minimum. To ensure that the SGD algorithm converges, it is common practice to start with a relatively large step-size, and reduce it gradually as the algorithm runs. If the step-size is not reduced, the algorithm will never converge, and will continue to jump around following the noisy gradient estimates.

In addition to reducing the computational burden, SGD also has another advantage. If the objective function has multiple local minima, SGD is not as likely to be trapped in a sub-optimal local minimum as a standard gradient descent algorithm. Intuitively, SGD with its noisy gradient estimates have a chance of escaping a bad local optimum.

Problems

1. What is the gradient of the following function

$$f(x_1, x_2) = x_1^2 \cdot \log(x_1 \cdot x_2)$$

2. If we start at $\mathbf{x}^{(0)} = [2, 2]^\top$ and run the gradient descent algorithm with $\alpha = 0.5$, to minimize the function

$$f(\mathbf{x}) = x_1^2 + 5 \cdot x_2^2$$

   what will the value $\mathbf{x}^{(1)}$ be (after the first iteration)?

3. Is the gradient descent algorithm guaranteed to converge?

4. If you were to run the following gradient descent algorithm

$$x^{(t+1)} = x^{(t)} - \alpha \cdot (2x - 6)$$

   with a suitably small value of α, to which value of x would it converge?

Solutions

1. $\nabla f = \left[2x_1 \log(x_1 \cdot x_2) + x_1, \frac{x_1^2}{x_2}\right]$
2. $\mathbf{x}^{(1)} = [0, -8]$
3. No, if the step-size is too high, the algorithm might diverge.
4. x=3

9 Neural networks and automatic differentiation

9.1 Neural networks

An artificial neural network are made up of simple building blocks, that are put together to form a complex model. To understand how neural networks work, we will start by examining the most simple, basic building block that is used to construct these networks. Such a building block called a *neuron* in analogy to how the human brain consists of a network of connected neuronal cells.

9.1.1 The neuron

Each neuron in an artificial neural network is a linear model that is passed through a non-linear activation function. The weights in the linear model inside the neuron are adjustable parameters of the neuron. Similar to the weights in linear regression, the weights of the neuron can be adjusted by to minimize a cost function that measures the discrepancy between the output of the neuron and the desired output.

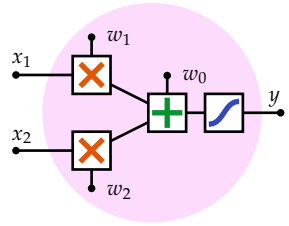


Figure 9.1: Sketch of a single neuron in an artificial neural network. Inside the neuron, each input feature is first multiplied by a weight, and the weighted inputs are then summed together with a bias weight. Finally, the sum is passed through a non-linear activation function to produce the output.

> Example 9.1 Neuron with two inputs
>
> Let us consider a neuron that takes in two input features x_1 and x_2 (see Figure 9.1.) Mathematically, we can the define the output of the neuron as
>
> $$y = f(w_0 + w_1 \cdot x_1 + w_2 \cdot x_2)$$
>
> where w_0, w_1 and w_2 are the weights (parameters inside

the neuron) and f is the non-linear activation function. An example of an often used activation function is the hyperbolic tangent, $f(x) = \tanh(x)$.

If we have the parameters $w_0 = 0$ and $w_1 = w_2 = 1$ and get the inputs $x_1 = 0$ and $x_2 = 2$, our neuron will produce the ouput

$$y = \tanh(0 + 1 \cdot 0 + 1 \cdot 2) = \tanh(2) \approx 0.964.$$

The output of the neuron for different inputs with three different settings of the parameters are illustrated in Figure 9.2. Examining the figure, can you approximately read off the output value we just computed?

In general, we can write the output of a single neuron with a K-dimensional input as

$$y = f\left(w_0 + \sum_{k=1}^{K} w_k x_k\right),$$

where f is a non-linear activation function and w_0, w_1, ..., w_K are the weights in the neuron. If we collect the inputs in a vector $\mathbf{x}$ and collect the weights except w_0 in a vector $\mathbf{w}$, we can write this more compactly using a dot product

$$y = f(w_0 + \mathbf{w}^\top \mathbf{x}).$$

Example 9.2 Implementing a neuron

With the inputs given as a vector `x` and the weights w_1, w_2,...,w_K stored in a vector `w` and the bias weight w_0 stored as a scalar variable `w0`, we can implement a neuron with a hyperbolic tangent activation using the following computer code

```
y = np.tanh(np.dot(x, w) + w0)
```

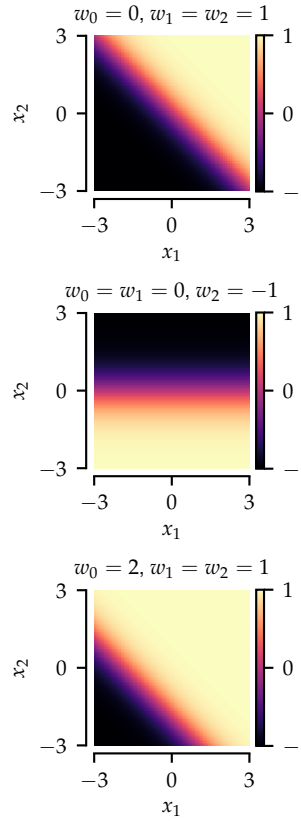


Figure 9.2: Image plots of the output of a single neuron with a two-dimensional input for different values of the weights.

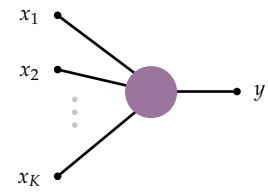


Figure 9.3: In general, a neuron takes K inputs and produces a single output.

Interpretation of the weights In linear regression, we have a nice intuitive interpretation of the weights: 1) The sign of each weight determines if the output tends to increase or decrease with the input and 2) the magnitude of each weight determines how much the output will change with one unit change of the input. This interpretation carries over to the neural network

to some extent: Usually the non-linear function is chosen as an increasing function, which means that the first intuition above holds for the neural network as well. But because of the non-linearity, the second intuition does not hold.

9.1.2 *The activation function*

If each neuron did not contain a non-linear function, it would not make much sense to put together many neurons in a big neural network: Since if each neuron were linear, the total network of neurons would also just be a linear model. The non-linear activation function in each neuron ensures that when we put together many neurons in a big neural network, the full network will be a (possibly very complex) non-linear function.

Many different non-linear functions can be used as activation functions in neural networks. Two of the most popular functions are the hyperbolic tangent (tanh) and the rectified linear unit (ReLU). These two functions are illustrated in Figure 9.4. The hyperbolic tangent can be defined as

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{2}{1 + e^{-2x}} - 1$$

It is a smooth function that maps the real numbers (from minus infinity to plus infinity) onto the range from -1 to 1. The rectified linear unit is a function that truncates negative inputs to zero and lets positive inputs pass directly through,

$$\text{ReLU}(x) = \max(0, x) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Many other activation function have been used, and each have their different pros and cons, but here we restrict our discussion to the hyperbolic tangent and rectified linear unit. Some of the considerations that go in to selecting the activation function include

Computational speed The ReLU requires almost no computation, whereas the tanh requires the computation of the exponential function.

Dead neurons When the signal just before the activation function is negative, the ReLU simply provides zero output. This means that many of the neurons in the neural network could possibly be inactive. With the tanh this problem does not occur.

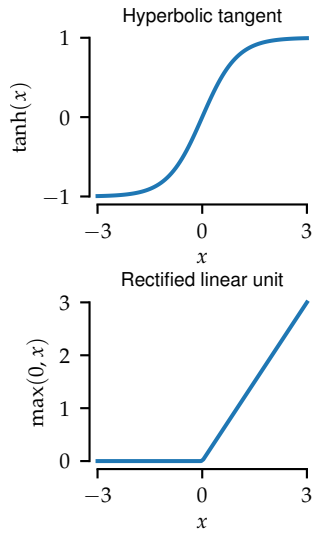


Figure 9.4: Two popular activation functions: The hyperbolic tangent and the rectified linear unit.

Exploding or vanishing gradients When training a deep neural network using gradient descent, we need the gradients to be well behaved. When using the tanh activation the gradients can sometimes tend to be smaller and smaller (vanishing gradients) as we move through the layers. With the ReLU the gradients sometimes tend to get bigger and bigger (exploding gradients).

Unfortunately it is not easy to give any general recommendations for how to choose the best activation function. The best recommendation is usually to try different ones and see what works best in practice.

9.1.3 *Connecting neurons*

To create a neural network, we can simple connect a number of neurons together. One way to do this is to first create a layer of neurons that each takes the features x as their input and produce some outputs. Then we can create a second layer of neurons that take the outputs from the first layer as inputs to produce new outputs. We can continue this process until we finally at some stage create a single neuron that takes all the outputs at the penultimate layer and produces the final output. This approach is called a multilayer perceptron and is illustrated in Figure 9.5.

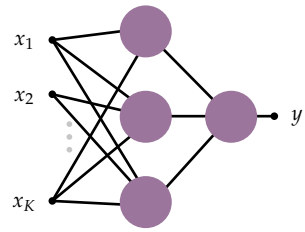


Figure 9.5: An example of a multilayer perceptron neural network with two layers. The first layer contains three neurons and the second layer contains a single neuron, that uses the output of the three neurons on the previous layer as input.

Example 9.3 Implementing multiple neurons

We have seen that a single neuron can be implemented using a dot product between the inputs and the weight vector. When we want to implement multiple neurons that all take the same input but have different weights, we can use a matrix product. Let us again say that we have the inputs in a vector `x` and now we store the weights of each neuron as rows in a matrix `W`. The bias weights w_0 for each neuron are stored in a vector `w0`. Then we can implement all the neurons in a single line of code:

```
y = np.tanh(np.dot(x, W) + w0)
```

Notice that this code is basically identical to the code for the single neuron, as the `np.dot` function can both be used for dot products and matrix multiplication.

To implement a multilayer perceptron, we can just re-

peat the code above for each layer of the network. In each line of code, the output from the previous layer is used as input in the next layer. For example, to implement a three-layer perceptron, we could write

```
h1 = np.tanh(np.dot(x, W1) + b1)
h2 = np.tanh(np.dot(h1, W2) + b2)
y = np.tanh(np.dot(h2, W3) + b3)
```

Here, we have three weight matrices `W1`, `W3`, and `W3` and three bias vectors which we have now named `b1`, `b2`, and `b3`.

9.2 Automatic differentiation

When we use the gradient descent algorithm to optimize the parameters in a model, we of course need to compute the gradient. As you may recall, the gradient is simply the vector of partial derivatives of the objective function with respect to each of the parameters in the model. In a simple model, it might be okay to just derive the formulas required to compute the gradient and implement them manually. But when we work with more complicated models such as deep neural network, it quickly becomes too cumbersome to derive everything by hand. Luckily, we can get the computer to figure out what the gradient is automatically, using an approach called automatic differentiation. In fact, automatic differentiation allows us to compute the gradient of almost anything that we can implement as a computer program.

9.2.1 *Computations as a graph*

To see how this works, let us start by considering how a mathematical expression is evaluated on a computer. Any mathematical expression can be evaluated by computing a sequence of elementary operations.

Example 9.4 Computation of a simple expression

Consider the following mathematical expression

$$z = x \cdot y + y \cdot \cos(\pi x).$$

When we want to evaluate this expression (that is com-

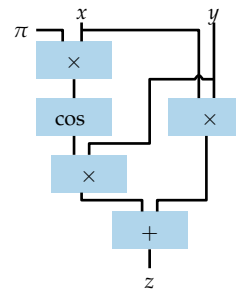


Figure 9.6: Computation graph for the mathematical expression in Example 9.4

pute the value of z for some given values of x and y) we usually work our way from the inside out and from left to right. First, we compute the product of x and y and save the result for later. Then we multipy π with x, take the cosine and save the result. Finaly we add together the two intermediate results we have computed. We can think of this step-by-step computation as a sequence of elementary operations,

$$
\begin{aligned}
a &= x \cdot y \\
b &= \pi \cdot x \\
c &= \cos(b) \\
d &= y \cdot c \\
z &= a + d
\end{aligned}
$$

Alternatively, we can think of the operations as a *computation graph* which links the input values to the output through a sequence of operations as shown in Figure 9.6.

9.2.2 *Computing a partial derivative*

Once we have formulated a computation as a graph, we can compute the partial derivative of the output with respect to any parameter we are interested in, simply by working our way backwards through the sequence of operations. Assuming that we know the derivative of each of the elementary functions that are used, at each step we simply apply this knowledge and chain together each result using the chain rule.

As you may recall, the chain rule says that we can evaluate the derivative of a function composite[1] by multiplying the derivative of the outer function with the derivative of the inner function.

[1] A function composite is when you take the output of a function and use it as input in another function, such as $f(g(x))$.

Definition 9.1 The chain rule

The derivative of a function composite

$$z(x) = f\big(g(x)\big)$$

is the product of the derivative of the outer function f evaluated at $g(x)$ and the derivative of the inner function g evaluated at x. With f' and g' denoting the derivatives of the function, this can be stated as

$$z'(x) = f'\big(g(x)\big) \cdot g'(x).$$

In Leibniz's notation, the chain rule is written as

$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}f}{\mathrm{d}g} \cdot \frac{\mathrm{d}g}{\mathrm{d}x}.$$

The chain rule makes it explicit how we can compute the derivative of a sequence of operations, as we have represented in a computation graph. We simply need to take the derivative of each computation and chain the results together by multiplying the derivatives.

When dealing with multivariable functions (such as functions that operate on vectors), an extension of the chain rule tells us that we simply need to apply the chain rule to each variable and sum the results.

Definition 9.2 The multivaiable chain rule

The derivative of a multivariable composite function

$$z(x) = f\big(g_1(x), g_2(x), \ldots, g_K(x)\big)$$

is the sum of the derivative of the outer function with respect to each of its arguments multiplied with the derivative of the respective inner function

$$\frac{\mathrm{d}z}{\mathrm{d}x} = \sum_{k=1}^{K} \frac{\partial f}{\partial g_k} \cdot \frac{\mathrm{d}g_k}{\mathrm{d}x}.$$

Example 9.5 Derivative of a simple expression

Continuing Example 9.4, let us compute the derivative of z with respect to one of our variables. We can compute the

derivative with respect to either x or y, but for now let us not decide which one we are interested in, and just take the derivative with respect to some arbitrary parameter that we call t. We start at the final computation, which is a sum so the derivative is the sum of the individual derivatives

$$\frac{\partial z}{\partial t} = \frac{\partial a}{\partial t} + \frac{\partial d}{\partial t}. \tag{9.1}$$

Then we go to the second to last computation, which is a product so we need to use the product rule,

$$\frac{\partial d}{\partial t} = y\frac{\partial c}{\partial t} + c\frac{\partial y}{\partial t}. \tag{9.2}$$

Continuing this process, we end up with the following sequence of derivatives, which we show here along side with the original computations

$$\begin{aligned}
a &= x \cdot y & \frac{\partial a}{\partial t} &= x\frac{\partial y}{\partial t} + y\frac{\partial x}{\partial t} \\
b &= \pi \cdot x & \frac{\partial b}{\partial t} &= \pi\frac{\partial x}{\partial t} \\
c &= \cos(b) & \frac{\partial c}{\partial t} &= -\sin(b)\frac{\partial b}{\partial t} \\
d &= y \cdot c & \frac{\partial d}{\partial t} &= y\frac{\partial c}{\partial t} + c\frac{\partial y}{\partial t} \\
z &= a + d & \frac{\partial z}{\partial t} &= \frac{\partial a}{\partial t} + \frac{\partial d}{\partial t}
\end{aligned}$$

The new sequence of computations can then simply be run forward to compute the partial derivative. For example, to compute $\frac{\partial z}{\partial x}$ we can simply insert the initial values $\frac{\partial x}{\partial t} = 1$ and $\frac{\partial y}{\partial t} = 0$.

We could manually find the partial derivative of our expression with respect to x to be

$$\frac{\partial z}{\partial x} = y - \pi y \cdot \sin(\pi x)$$

Evaluating the derivative $\frac{\partial z}{\partial x}$ at $x = 0.5$, $y = 2$ we get

$$\left.\frac{\partial z}{\partial x}\right|_{x=0.5,y=2} = 2 - \pi \cdot 2 \cdot \sin(\pi \cdot 0.5) = \underline{2(1-\pi)}$$

We can get to the same result by inserting numbers in the

sequential computations above, which gives us

$$
\begin{aligned}
a &= 0.5 \cdot 2 = 1 & \frac{\partial a}{\partial t} &= 0.5 \cdot 0 + 2 \cdot 1 = 2 \\
b &= \pi \cdot 0.5 = \tfrac{\pi}{2} & \frac{\partial b}{\partial t} &= \pi \cdot 1 = \pi \\
c &= \cos(\tfrac{\pi}{2}) = 0 & \frac{\partial c}{\partial t} &= -\sin(\tfrac{\pi}{2}) \cdot \pi = -\pi \\
d &= 2 \cdot 0 = 0 & \frac{\partial d}{\partial t} &= 2 \cdot (-\pi) + 0 \cdot 0 = -2\pi \\
z &= 1 + 0 = 1 & \frac{\partial z}{\partial t} &= 2 + (-2\pi) = \underline{2(1 - \pi)}
\end{aligned}
$$

Problems

1. Repeat the calculations at the end of Example 9.5 to evaluate the derivative of z with respect to y at $x = 0.5$, $y = 2$. Check your result by inserting in $\frac{\partial z}{\partial y} = x + \cos(\pi x)$.

Solutions

1. Computation of $\left.\frac{\partial z}{\partial y}\right|_{x=0.5,y=2}$

$$
\begin{aligned}
a &= 0.5 \cdot 2 = 1 & \frac{\partial a}{\partial t} &= 0.5 \cdot 1 + 2 \cdot 0 = 0.5 \\
b &= \pi \cdot 0.5 = \tfrac{\pi}{2} & \frac{\partial b}{\partial t} &= \pi \cdot 0 = 0 \\
c &= \cos(\tfrac{\pi}{2}) = 0 & \frac{\partial c}{\partial t} &= -\sin(\tfrac{\pi}{2}) \cdot 0 = 0 \\
d &= 2 \cdot 0 = 0 & \frac{\partial d}{\partial t} &= 2 \cdot 0 + 0 \cdot 1 = 0 \\
z &= 1 + 0 = 1 & \frac{\partial z}{\partial t} &= 0.5 + 0 = \underline{0.5}
\end{aligned}
$$

Check:

$$\frac{\partial z}{\partial x} = x + \cos(\pi x) = 0.5 + \cos(\pi \cdot 0.5) = \underline{0.5}$$

11 Reinforcement learning

In reinforcement learing, the objective is to learn a policy that can control an agent in an environment in such a way that the agent achieves a high reward in the long run. Reinforcement learning is a very general framework that can be used to solve many different tasks.

11.0.1 Markov decision process

Typically, a reinforcement learning problem is formulated as a Markov decision process in the following way: We assume that the agent interacts with the environment in a sequence of steps. At each step, the agent must decide upon an action a to perform. Based on the action, the environment changes its state s, and after each step the agent recieves some reward r.

Environment The agent observes the environment in form of a *state* s. The environment can either be fully observed, in which case the agent has access to all information about the environment, or it can be partially observed such that the agent can only access a smaller part of the total information. The dynamic behavior of the environment is governed by a set of rules that controls what happens to the environment state when the agent takes a particular action. These rules can be deterministic or stochastic: In the deterministic case, the rules define what the next state s' of the environment will be if the current state is s and the agent takes action a, which can be formulated as a function $s' = f(s, a)$. In the stochastic case, the rules define a probability distribution over possible next states, $p(s'|s, a)$.

Action The agent interacts with the environment by taking actions a. The set of possible actions can be discrete (such as

go north, east, south, or west in a maze) or continuous (such as apply a voltage between 0V and 5V to a motor in a robot arm). Actions can also be multivariate, for example when controlling a complicated robot with many joints. The set of possible actions can depend on the state of the environment: For example, if we have reached a dead end in a maze, we cannot continue to walk forward.

Reward After taking an action, the agents recieves a reward r. The reward is a single number, and the goal of the agent is usually to optimize the total expected cumulative reward. In the general setting, the reward can depend on the current state, the chosen action, and the next state, and it can either be defined as a deterministic function $r = r(s, a, s')$ or in the stochastic setting as a probability distribution $p(r|s, a, s')$. When the environment is deterministic (the next state can be determined uniqely from the current state and action), the reward is typically written as a function of the current state and action, $r(s, a)$.

In settings where the reinforcement learning problem can potentially continue indefinitely, the expected reward is often weighted (discounted) such that the agent will focus on achieving high reward sooner rather than later. The discounting is also used to ensure that the expected reward does not diverge; if an agent is allowed to continue collecting reward forever, the expected cumulative reward of any reasonable policy is infinite and can thus not be used as an optimality criterion.

11.1 Deterministic value iteration

One way to approach the reinforcement learning problem is to estimate the *value* of each state denoted by $v(s)$. We refer to this as the *value function*. In the following, we consider a deterministic environment, $s' = f(s, a)$, with a finite number of states, and a deterministic reward associated with each state-action pair, $r(s, a)$. In this setting we have the following definition of the value function.

Definition 11.1 Value function (deterministic setting)

The value function is defined recursively as

$$v(s) = \max_a \left(r(s,a) + \gamma \cdot v(s') \right).$$

Let us understand what this equation means: We are in state s and we can now take some action a. Depending on which action we take, we end up in a new state $s' = f(s,a)$. The value of our current state s is equal to: $r(s,a)$ (the reward we get for taking the action a from our current state s) plus $\gamma \cdot v(s')$ (the value of the next state s' discounted by γ) for the action a that leads to the highest value.

The discount factor γ is a number between 0 and 1: A typical value might be 0.9. The discount makes us give higher priority to immediate rewards, rather than rewards further out in the future. Furthermore, the discounting ensures that the value of the cumulative reward is never infinite.

Once we have estimated the value function, the optimal policy is to take the action that leads to the highest discounted future reward. Mathematically, we write this as

$$a^* = \arg\max_a \left(r(s,a) + \gamma \cdot v(s') \right),$$

where as usual $s' = f(s,a)$ is the next state and s is the current state. To find the optimal action, we need to know which next state each action will take us to.

But how can we in practice compute the value function? Since the value $v(s)$ depends on the value of the next state $v(s')$, we have a recursively defined optimality criterion: A set of equations that must be satisfied for all states. One way to estimate the value function is to start by initializing the the value of all states (perhaps randomly or all zeros) and then loop through all states and update their value according to the definition over and over again, until the value function converges. This approach is called *value iteration*.

Example 11.1 Up and down the ladder

Consider the reinforcement learning problem illustrated in Fig. 11.1. We have 10 states named 0–9, and in each state we have two possible actions, *up* and *down*. The reward associated with all state-action pairs is zero except for the

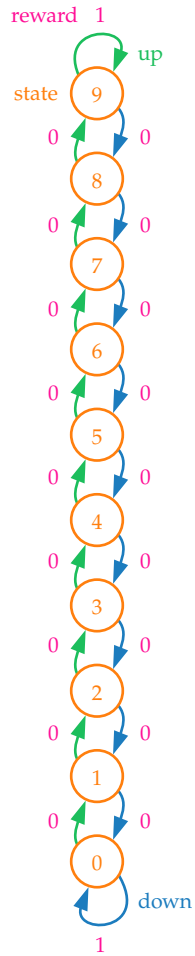


Figure 11.1: An agent can move *up* or *down* through ten states 0–9. Taking the action *up* in state 9 returns the agent to state 9, and similar, taking action *down* in state 0 returns the agent to state 0. The reward is 1 for transitioning into state 0 or 9, and zero for all other state-action pairs.

actions that self-transition in the top and bottom states (0 and 9) where the reward is one. The goal is thus to reach one of these two states as quickly as possible and stay there.

Since the only non-zero rewards are associated with state 0 and 9, the optimal policy is obviously to move *down* in states 0–4 and *up* in states 5–9. This will most quickly get the agent to one of the rewarding states and stay there forever.

Now, what is the *value* of the different states? This depends on the value we choose for the discount factor. Here, let us use $\gamma = 0.9$, and let us first consider state 0. Following the optimal policy (*down*) will keep us in state 0 forever, so we get reward 1 at each time step. The value of state 0, $v(0)$, can thus be computed as

$$v(0) = 1 + \gamma v(0) \tag{11.1}$$

$$= 1 + \gamma + \gamma^2 + \cdots \tag{11.2}$$

$$= \frac{1}{1-\gamma} = 10 \tag{11.3}$$

We can continue in a similar manner to compute the value of all other states. But note, that this approach required us to guess the optimal policy.

Instead, let us use *value iteration* to compute the value of all states. We can do this using a small computer program where we initialize the value of each state to zero and then repeatedly loop over all states and update their value.

```
# Initial values
v = [0,0,0,0,0,0,0,0,0,0]
# Discount
gamma = 0.9
# Actions
actions = ['up', 'down']

# Next state function
def f(s, a):
    if a=='up':
        return min(s+1, 9)
    if a=='down':
        return max(s-1, 0)
# Reward function
def r(s, a):
```

```
    if (s==0 and a=='down') or (s==9 and a=='up'):
        return 1
    else:
        return 0

# 1000 value iterations
for t in range(1000):
    # Loop over all states
    for s in range(10):
        # Update value
        v[s] = max([r(s,a) + gamma*v[f(s,a)] for a in
    actions])
```

After running this code, the state values converge to:

s	0	1	2	3	4	5	6	7	8	9
$v(s)$	10	9	8.1	7.29	6.561	6.561	7.29	8.1	9	10

11.2 Deterministic q-learning

Q-learning is a *model free* reinforcement learning methods. In contrast to value iteration, q-learning does not require that we have knowledge about the state transition mechanism, but learns directly by interacting with the environment.

The "q" in the name stands for *quality*, and it is named so because the central aspect of the method is to estimate a quality function $q(s,a)$. Analogous to the value function discussed previously, the quality function is also called the *state-action value* function, and it measures the value of taking action a in state s. Similar to value iteration, the quality is measured in terms of discounted cumulative reward.

In the setting where state transitions and rewards are deterministic, we have the following recursive definition of the quality function.

Definition 11.2 Quality function (deterministic setting)

The quality function is defined recursively as

$$q(s,a) = r(s,a) + \gamma \max_{a'} \left(q(s',a') \right). \tag{11.4}$$

Similar to value iteration, we can use this formula to iteratively update the quality function until it converges. However, there is an important difference: In the value iteration algorithm,

we updated the value of a state based on the reward and the highest valued next state we could reach. This required us to know exactly which states we could reach from each state. In other words, we needed a model of the environment. Here, in q-learning, we update the quality function based on the reward and the quality of the next state. Thus, we do not need to know which states we can possibly reach from the current state: All we need to do is to take an action in the envirionment, which will bring us to some new state, and we can then update the quality function based on this interaction.

11.2.1 *Epsilon-greedy action selection*

With q-learning, we can thus update the quality function by interacting with the environment and learning about the state transitions as we go along. However, in order for the algorithm to converge, we must interact with the environment in such a way that we are guaranteed to visit all states and try all actions. Obviously, if there are states or actions that are never explored, we would have no way to learn about them. In value iteration, this issue was solved by simply looping over all states to do the updates.

In q-learning, the typical approach is to use the *epsilon greedy* strategy. The epsilon greedy strategy consists two options:

1. With probability $1 - \epsilon$ we take the best action according to the current estimate of the quality function,

$$a^* = \arg\max_a q(s, a).$$

2. With probability ϵ we take a random action.

The constant *epsilon* definies a trade-off between *exploration* and *exploitation*. In the one extreme $\epsilon = 1$, the algorithm takes completely random actions and thus explores the envirionment through a random walk. In the other extreme $\epsilon = 0$ the algorithm exploits its current knowledge by taking actions that are optimal under the current (possibly flawed) estimate of the quality function. In practice, a selecting a good value for *epsilon* can be highly problem dependent, but a typical reasonable value could be around $\epsilon = 0.1$ such that 10% of the actions are random.

Problems

1. Show that the value of state 5 in the "up and down the ladder" problem is $v(5) = 6.561$. Use the optimal strategy (always go *up*) with $\gamma = 0.9$ and compute

$$v(5) = 0 + \gamma v(6) = 0 + \gamma\left((0 + \gamma v(7)\right) = \ldots$$

Solutions

1. We start by expanding $v(5)$ until we have it expressed in terms of the final state $v(9)$

$$\begin{aligned} v(5) &= 0 + \gamma v(6) = \gamma v(6) \\ &= \gamma\left(0 + \gamma v(7)\right) = \gamma^2 v(7) \\ &= \gamma^2\left(0 + \gamma v(8)\right) = \gamma^3 v(8) \\ &= \gamma^3\left(0 + \gamma v(9)\right) = \gamma^4 v(9). \end{aligned}$$

We can then evaluate the value of the final state as

$$v(9) = 1 + \gamma v(9) \Leftrightarrow v(9) = \frac{1}{1-\gamma} = 10.$$

Inserting this, we arrive at

$$v(5) = \gamma^4 v(9) = 0.9^4 \cdot 10 = 6.561$$

12 Causality

In causal estimation one of the primary goals is to estimate the strength of the causal effect of one variable on another. In other words: We are interested in finding out how much the value of one variable influences the value of another variable. This will allow us to answer causal questions such as: "If I were to modify variable x, how much would I expect variable y to change?". This is not always an easy problem, and the answer most often depends on additional knowledge that is not evident in the statistical data alone.

Example 12.1 Bullet holes in air planes[1]

During the second world it was noticed that air planes returning from battle had more bullet holes in the fuselage than in the engine. At first, this finding suggested that more armour should be added to the fuselage to protect them where they seemed to get hit the most. However, a statistician who examined the case came to the opposite conclusion: They should add more armour to the engine. While at first this might seem backward, the argument is clear: If we assume that the bombers are equally likely to get hit everywhere, the fact that the air planes that return have more bullet holes in the fuselage suggests that planes that are hit in the engine are more likely to crash and therefore not return. The planes that do return are the ones that are hit where it does not matter so much.

[1] This example is based on a true story as told by Jordan Ellenberg in the book "How Not to Be Wrong: The Power of Mathematical Thinking" regarding the statistician Abraham Wald who published his results in the 1943 report "A Method of Estimating Plane Vulnerability Based on Damage of Survivors"

Often causal estimation is discussed in the language of experimental design, where we imagine a hypothetical study where some units are given a treatment x and we measure the outcome y. The treatment could be a binary variable (some units recieve the treatment, others get no treatment) or it could be a

continuous variable (for example the size of a dose of medicine). Similarly, the outcome could be binary or continuous.

12.1 Statistical dependence

Since causal estimation is concerned with measuring the causal dependence between variables, we begin by discusion how to measure statistical dependence. Here, it is easier to begin by defining when two variables are *not* associated, i.e. there is no statistical dependence.

Definition 12.1 Statistical independence

Two variables x and y are said to be statistically independent if there is no association between the variables: Knowing something about x tells us absolutely nothing about y and vice versa. Technically we say that the joint probability of x and y factorizes as the product of the individual distributions $p(x)$ and $p(y)$,

$$p(x, y) = p(x)p(y).$$

Example 12.2 Independent coin flips

If we flip a fair coin we get a random outcome of either heads or tails, and each outcome occurs with 50% probability. If we flip the coin twice and call the two outcomes x and y, we will observe one of the following sequences, each of which occur with probability 25%:

x	y	$p(x, y)$
heads	heads	0.25
heads	tails	0.25
tails	heads	0.25
tails	tails	0.25

In this case x and y are statistically independent: Knowledge about the outcome of one of the coin flips tells us nothing about the outcome of the other. We can verify this using the definition of statistical independence. Using the

individual probabilities of the outcomes

$$p(x = \text{heads}) = p(x = \text{tails}) = 0.5$$
$$p(y = \text{heads}) = p(y = \text{tails}) = 0.5$$

we can verify that $p(x, y) = p(x)p(y) = 0.25$ for any value of x and y.

12.1.1 *Linear dependence*

Obviously, if two variables are not statistically independent, we say that they are statistically dependent. In that case, information about one of the variables tells us something about the other variable. The dependence between two variables x and y can either be positive or negative: If observing some particular value of x makes it more likely to observe some particular value of y, we say there is a positive dependence. If some particular value of x makes some particular value of y more unlikely, the dependence is negative.

Example 12.3 Dependent coin flips

Let us consider flipping a coin to get a random outcome of either heads or tails. Each of these possible outcomes occur with probability 50%. Let us call the outcome x, and let us define another variable y to be the *opposite* possible outcome. In other words, if x is tails, the y is heads and vice versa. The possible combined outcomes we can see and their probability would then be given by:

x	y	$p(x, y)$
heads	tails	0.5
tails	heads	0.5

Clearly, x and y are not independent, since they are always the opposite of each other. We can verify this by checking the definition of statistical independence, namely that the joint probabiliy is equal to the product of the individual marginal probabilities, $p(x, y) = p(x)p(y)$, for any values of x and y. As in the previous example we have that the

probability of each individual outcome of x and y is 50%

$$p(x = \text{heads}) = p(x = \text{tails}) = 0.5$$
$$p(y = \text{heads}) = p(y = \text{tails}) = 0.5$$

However, the joint probability $p(x = \text{heads}, y = \text{tails}) = 0.5$

$$p(x = \text{heads}, y = \text{tails}) = 0.5$$
$$\neq$$
$$p(x = \text{heads})p(y = \text{tails}) = 0.5 \cdot 0.5 = 0.25$$

This proves that x and y are not independent. In this case there is a negative dependence between observing $x = \text{heads}$ and $y = \text{heads}$, because if we see $x = \text{heads}$ it makes it less likely (impossible actually) to see $y = \text{heads}$.

One particular way that we can measure the degree of dependence between two variables is by their *covariance* and the related *correlation coefficient*. Both of these are measures of the *linear* dependence between two variables[2]. Covariance is defined in a similar way as variance is defined for a single variable. Where the variance is the average squared deviation from the mean, the covariance is the average product of the difference of the mean for the two variables.

[2] Note that it is possible for two variables to have a non-linear dependence while having zero covariance; thus, if the covariance between two variables is zero, it does not necessarily mean that the variables are independent; however, if they are independent the covariance will be zero.

Definition 12.2 Covariance and correlation

Covariance

The covariance between two variables is a measure of the linear dependence between two variables. It can be computed as:

$$\text{cov}(x, y) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu_x)(y_i - \mu_y).$$

Correlation coefficient

The correlation coefficient between two variables is their covariance normalized by the product of their standard deviations, and is computed as:

$$\rho_{x,y} = \frac{\text{cov}(x, y)}{\sigma_x \cdot \sigma_y}.$$

The correlation coefficient is a number between -1 and 1.

Example 12.4 Covariance and correlation coefficient

Consider two data sets which consist of five numbers each:

$$x = \{1, 8, 4, 10, 2\}, \quad y = \{7, 5, 4, 3, 6\}.$$

Recall from Example **??** that the means and standard deviations of these two data sets are:

$$\mu_x = 5, \quad \mu_y = 5, \quad \sigma_x \approx 3.46, \quad \sigma_y \approx 1.41.$$

Using the means, we can compute the covariance between x and y as

$$\begin{aligned} \text{cov}(x, y) &= \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu_x)(y_i - \mu_y) \\ &= \frac{1}{5} \Big((1-5)(7-5) + (8-5)(5-5) + (4-5)(4-5) \\ &\qquad + (10-5)(3-5) + (2-5)(6-5) \Big) = -4 \end{aligned}$$

The fact that the covariance is negative means that when x tends to be relatively high (greater than its mean) then y tends to be relatively low (smaller than its mean).

Finally, we can compute the correlation coefficient, which measures the degree of correlation as a number between -1 and 1:

$$\rho_{x,y} = \frac{\text{cov}(x, y)}{\sigma_x \cdot \sigma_y} \approx \frac{-4}{3.46 \cdot 1.41} \approx -0.82$$

12.2 Correlation is not causation

If two variables are strongly correlated, we might think that there is a causal link between them; however, we should be very careful when we try to explain *why* the correlation occurs, because there could be many equally good explanations and we might not have enough information to determine the correct one.

Causal relation It could be the case, that there is a causal relation between the variables: This would mean that the changes observed in one of the variables is caused by changes in the other variable. But since the measure of correlation is symmetrical, we cannot say which variable is the cause and

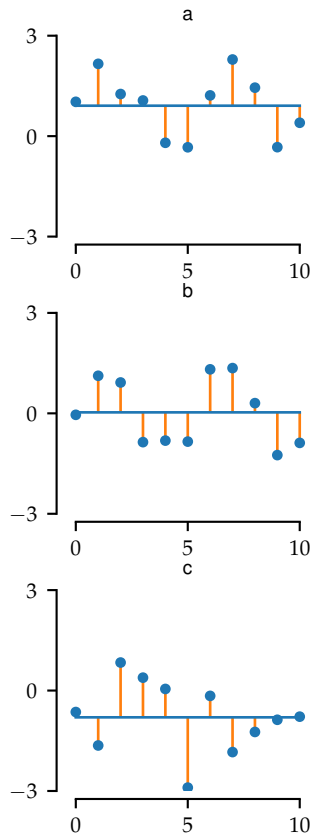


Figure 12.1: Examples of correlations. The plots shows the mean and 11 obervations of three different variables. The data in the two top panes are corelated: When one variable is above its mean the other also tends to be above its mean and vice versa. The bottom data is uncorrelated with the data in the top two panes.

which is the effect. To determine the direction of the causal link requires further information or assumptions.

Bidirectional relation The cause-effect relation might also go both ways, as for example in a predator-prey relation: As the number of predators increases, the number of prey will decrease (as they are eaten), which in turn leads to fewer predators (as they starve) and so on.

Confounding Another possibility could be that there is some third, unobserved variable, (known as a confounder) that is a common cause of both of our observed variables, and the observed correlation between our variables simply occurs because they are both influenced by this common cause.

Spurious correlation Finally, the observed correlation might simply be a random coincidence, a phenomenon known as *spurious* correlation.

Definition 12.3 Correlation and causation

Correlation is a statistical measure that describes the strenght of the linear relationship between two variables.

Causation means that one variable (the effect) is the result of another variable (the cause). If there is a causal relationship between two variables, and we were to manually modify the cause-variable, the effect-variable would be affected as well.

Example 12.5 Income earned and hours worked

Let us say that we have measured two variables "income earned" and "hours worked", and we have estimated a strong positive correlation (people who work more hours earn more income and vice versa). Is it likely that there is a direct causal effect? Well yes, we could imagine that if we increased the number of hours worked the income would increase as well, for example if we are talking about people working on hourly wages. So the "hours worked" could possibly have a causal effect on "income earned".

A causal effect the other way around is perhaps not so easy to imagine: If we increase their income, people would

probably not start working more hours.

But before we make any conclusions we should think things through: Imagine a scenario where everybody have a fixed monthly wage and no overtime pay, and imagine that people with a high wage tend to put in more overtime without compensation. In that case "hours worked" and "income earned" would be correlated. But since no-one gets paid for their overtime, changing the number of hours worked will have no causal effect on the income.

The bottom line is that it requires strong assumptions about how the world works to go from an observed correlation to a causal explanation.

12.2.1 *Confounding variables*

In many situations the direction of the causal relation between two variable is fairly obvious, for example if the cause logically preceeds the effect in time or there is some other well established theory about their causal relation. In that case we often refer to the hypothesized cause as the "treatment" and the effect as the "outcome". But even in that situation we should be very cautious about using a measure of correlation to say something about the strength of the causal relationship between treatment and outcome. One reason is that there could be *confounding* variables that influence the measured correlation by affecting both the treatment and the outcome (see Figure 12.2).

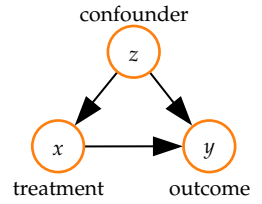


Figure 12.2: Causal diagram: The treatment is a direct cause of the outcome, but the confounder is a cause of both the treatment and the outcome.

Definition 12.4 Confounding variable

A *confounder* is a variable (often unobserved) that influence both the treatment and the outcome. In the presence of unobserved confounding variables, the observed correlation cannot be taken as evidence for the strength of a hypothesized causal relation.

When an unobserved confounder is present, the observed correlation between the treatment and the outcome could be completely different from the causal relation. It might even be the case that there is a positive observed correlation even if the underlying causal relation is negative.

Example 12.6 The best AI teacher

Let us say that a university course in AI runs two times per year. In the spring semester the course is taught by Teacher A, and in the fall it is taught by Teacher B. We now observe the number of students who pass the standardized exam in the two semesters:

- 95 out of 100 students (95%) pass the course with Teacher A in the spring.
- 72 out of 80 students (90%) pass the course with Teacher B in the fall.

We can think of the teacher as the treatment and the proportion of students who pass the exam as the outcome: It seems that Teacher A does a better job than Teacher B, and it would appear that taking the course with Teacher A more likely leads to passing the exam.

Now, assume we asked the teachers about their classes and they gave us the following further information.

- In the spring, 80 of the students were experienced graduate students 78 of whom passed the exam, and the remaining 20 were inexperienced freshmen 17 of whom passed.
- In the fall, 10 of the students were graduate students who all passed and 70 were freshmen of whom 62 passed.

If we examine how well the two types of students do, we might break down the numbers as follows:

	Graduates	Freshmen	Total
Teacher A	78/80≈98%	17/20=85%	95/100=95%
Teacher B	10/10=100%	62/70≈89%	72/80=90%

Now we can see that while Teacher A overall has the best passing rate, the passing rate for Teacher B is better *both for the graduates and the freshmen*.

In this example, the type of student is a confounder. The passing rate with Teacher A in the spring is higher in part because more experienced students take the course in

the spring. While this more detailed analysis might suggest that Teacher B is best after all, there is no guarantee that there are not other unobserved confounders, which might lead us to reverse our conclusion again.

12.2.2 *Randomized trials*

If we want to interpret an observed correlation as a causal relation, we need to make sure that we take into account all possible confounding variables. In an observational study, where we gather data from some "real life" situation, there is no general way to find out which confounders that are present. If we did have access to measurements of all the confounders, we could in principle adjust for them to estimate the strength of the causal effect. One way to do this would be to divide our observations into subgroups in which the confounder is practically constant, and perform our analysis on each subgroup separately, as we did for the different types of students in Example 12.6. However, this approach relies strongly on the assumption that we have measured all relevant confounders.

Another approach, which completely sidesteps the issue with possible confounders is to conduct a *randomized trial*. The idea in a randomize trial is that the decision about who gets the treatment (or how large a treatment) is completely randomized, which by definition guarantees that there are no possible confounders. Since the decision about the treatment is completely specified by a randomized procedure, there can be no other hidden causes that affect the treatment, and thus there is no confounders. In a randomized trial, it is suddenly easy to estimate the strength of a causal relation, because any observed correlation immediately can be interpreted as causal.

Problems

1. What is the covariance and correlation coefficient between x and y in Example 12.2 concerning the dependent coin flips? Hint: Consider a sample that includes every possible outcome.

2. I have observed that my average speed (in my car) is 62 km/h when I use my summer tires, and 58 km/h when I use my winter tires. If the type of tire is the treatment and the average speed of the car is the outcome, what are possible confounders?

3. What do you think are the most probable explanations for the following observed relations

   (a) Dancing at parties is correlated with throwing up.

   (b) Snow is correlated with road accidents.

   (c) Cheese consumption is correlated with the risk of dying by becoming entangled in bedsheets.

   (d) Smoking is correlated with cancer.

   (e) Moderate alcohol consumption is correlated with increased life expectancy.

Solutions

1. $\text{cov}(x, y) = -0.25$ and $\rho_{xy} = -1$.
2. One possible confounder could be the weather (winter tires are used in winter weather, which also might cause me to slow down.)
3. While there is not necessarily any correct answer here, possible explanations could be
   (a) A common cause could be alcohol consumption.
   (b) This is most likely direct causal relation.
   (c) This is probably a spurious correlation.
   (d) This is most surely a direct causal relation.
   (e) Who knows — the jury is still out on this one.

13 Fairness

Artificial intelligence and machine learning are being used more and more for automated decision making. This is also the case for sensitive issues such as whether a bank grants a loan, whether a citizen is enrolled in a social program, or whether a child is forcibly removed from their family.

The hope is that the use of AI will result in better decisions and outcomes because the algorithms can take into account more factors and learn from historical data as well as from its own mistakes. At the same time, there is a risk that learning from previous human decisions will reinforce existing biases and that reliance on quantitative data could introduce new forms of discrimination. As a result, there is a strong need to incorporate *fairness* into algorithms in order to protect disadvantaged groups. With the right approach, we might even be able to design AI algorithms that are optimal not only in terms of prediction and performance, but also in terms of fairness and equal opportunity.

A naïve approach to avoiding discrimination against a *protected attribute*, such as gender, race, or disability, would be to simply exclude that attribute from the model. However, this does not resolve the issue because the protected attribute may be correlated with other attributes that do enter the model: Even if the protected attribute is not directly used, it will influence the decision through its statistical dependence on other attributes. Therefore we require a better way to guarantee that a decision is not influenced by the protected attribute.

13.1 Fairness in binary decisions

Let us consider a setting where we want to make a decision $\hat{Y}$ based on attributes X while ensuring that the decision is fair with respect to a protected attribute G. To keep the discussion simple, we restrict ourselves to the binary setting, where $\hat{Y} \in \{0, 1\}$ is a binary decision, and the protected attribute $G \in \{a, b\}$ is a binary group membership variable, whereas X can be any attributes we include in making our decision $\hat{Y}$. Some fairness criteria also require us to know, at least in part, if the decision we made was correct: We will denote the correct decision by Y.[1]

[1] In some cases, it is not possible to ever know if the decision is correct. For example, in the case of a bank granting a loan, the correct decision could be defined as granting the loan to customers who repay but not to customers who default their payment. If the loan is granted, the bank will eventually learn if the decision was correct, but if the loan is not granted they have no direct way to find out if that was a correct decision.

We assume that we have access to a function

$$\hat{S} = f(X), \tag{13.1}$$

that outputs a score, $\hat{S}$, which we use to make our decision. This function could typically be based on a machine learning model, optimized on training data to predict some measure of success we would like to attain in our decision making.

While the protected group G does not enter directly into the model, it can be used either to evaluate the fairness of the decision procedure, or as a part of fitting the machine learning model to optimize fairness.

Once we have our function, we can use it to make decisions by selecting a threshold τ and deciding $\hat{Y} = 1$ when the score exceeds the threshold,

$$\hat{Y} = \begin{cases} 1 & \text{if } \hat{S} > \tau, \\ 0 & \text{otherwise.} \end{cases} \tag{13.2}$$

In this setting we can summarize the decision procedure by a *decision matrix* for each of the protected groups,

	$G = a$		$G = b$	
	$\hat{Y} = 0$	$\hat{Y} = 1$	$\hat{Y} = 0$	$\hat{Y} = 1$
$Y = 0$	a_{00}	a_{01}	b_{00}	b_{01}
$Y = 1$	a_{10}	a_{11}	b_{10}	b_{11}

The matrix lists the number of times we made the decisions $\hat{Y} = 0$ and $\hat{Y} = 1$ when the correct decision would have been $Y = 0$ and $Y = 1$ for each of the two protected groups, $G = a$ and $G = b$.

To ensure that our decisions are fair, we need two things:

1. A method to adjust our classification procedure to optimize its fairness.

2. A clear, mathematical criterion that defines what it means to be *fair*, which we can either optimize or use to check if our decision procedure is fair.

13.2 Optimizing fairness

There are several methods that can be used to optimize a decision procedure to make it more fair. When we base our decision on a scoring function $f(X)$ learned from data, there are three fundamental approaches:

1. **Modify the training data to ensure that it is representative of a fair and unbiased decision process.** We need to make sure our data is representative for all protected groups: This might actually mean that we need an unbalanced data set, for example if a fair scoring function is more difficult to learn for a particular protected group.

2. **Modify the training method to ensure that the learned scoring function is fair and unbiased.** Several methods have been proposed which either directly optimize a mathematical fairness criterion, or ensure fairness in an implicit manner. An example of an implicit method is shown in Figure 13.1, where the idea is to train a classifier to predict the target Y based on features extracted from X in such a way that the protected group G cannot be predicted from the features. This ensures fairness in the sense that no information reflecting the protected group is used in the classification.

3. **Modify the way the scoring function is used to make decisions, so that they are fair and unbiased.** For example, we can modify or recalibrate the scores for the different protected groups to reduce bias. In the binary decision setting, as simple method is to choose different threshold values for each protected group.

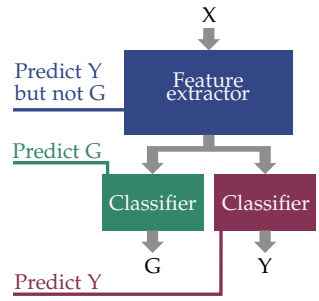


Figure 13.1: A neural network approach to create a fair classifier: The input layers (feature extractor) are optimized to learn feature that predict the target Y but *not* the protected attribute G. The two output blocks are trained to predict Y and G respectively.

13.3 Fairness criteria

13.3.1 Demographic parity

A simple fairness criterion is to require, that we decide $\hat{Y} = 1$ and $\hat{Y} = 0$ in the same fraction of cases in each protected group. Mathematically we may formulate this criterion as[2]

$$P(\hat{Y}=1|G=a) = P(\hat{Y}=1|G=b), \tag{13.3}$$

which we can read as: "The probability of deciding $\hat{Y} = 1$ for a case in protected group a is the same as the probability of deciding $\hat{Y} = 1$ for a case in protected group b." This does not take into account whether the decision was correct or not.

[2] This also implies that we decide $\hat{Y} = 0$ in the same fraction of cases in the two protected groups, which you will be asked to show as an exercise.

Based on the decision matrix, this criterion can be evaluated as

$$\frac{a_{01} + a_{11}}{n_a} = \frac{b_{01} + b_{11}}{n_b}, \tag{13.4}$$

where $n_a = a_{00} + a_{01} + a_{10} + a_{11}$ (and similarly n_b) is the total number of cases in the protected group.

While the demographic parity criterion is simple and intuitive, it has several potential drawbacks. If the target Y has a different distribution in the two protected groups, it might be too much to require that the percentages match in the decisions. Furthermore, if the scoring model is not equally good for the two groups, perhaps because of lack of training data for one of the groups, this will not be reflected in the decision.

> **Example 13.1 Demographic parity: University enrollment**
>
> A university uses a machine learning model to determine which applicants to accept as students. The system is trained on historical data to predict a student success score $\hat{S}$, and it depends on a number of student attributes X including high-school grades etc. This year, the university plans to enroll the 500 highest ranked applicants.
>
> Before making its final decision, the university decides to examine whether the decision process is fair for socially disadvantaged applicants. To do this, they divide the applicants into two groups, a and b, consisting of applicants from socially advantaged and disadvantaged groups respectively.
>
> According to the model, the university plans to accept 450 out of $n_a = 900$ applicants from group a and 50 out of

$n_b = 200$ applicants from group b.

Is this fair, according to the demographic parity criterion? No, because the fraction of students accepted from the two groups are not equal:

$$\text{Group } a: \quad \frac{450}{900} = 50\ \% \tag{13.5}$$

$$\text{Group } b: \quad \frac{50}{200} = 25\ \% \tag{13.6}$$

How could the university achieve demographic parity? Easy: Since the fraction of applicants in the two groups are known,

$$\text{Group } a: \quad \frac{900}{900+200} \approx 82\ \% \tag{13.7}$$

$$\text{Group } b: \quad \frac{200}{900+200} \approx 18\ \% \tag{13.8}$$

they could accept 82 % ($0.82 \cdot 500 = 410$ students) from group a and 18 % ($0.18 \cdot 500 = 90$ students) group b. This corresponds to choosing a different score threshold, τ, for each group.

While this would make the enrollment fair according to demographic parity with respect to the protected group, this necessarily also means that the university will accept some applicants from group b with a lower score than some rejected applicants from group a.

13.3.2 *Equalized odds*

A slightly more elaborate criterion, which avoids some of the downsides of demographic parity is the *equalized odds* criterion. This criterion requires that the probability of making a correct positive decision $\hat{Y} = 1$ among positive cases $Y = 1$ is the same in the two protected groups, and that the probability of making a correct negative decision $\hat{Y} = 0$ among negative cases $Y = 0$ is also the same in the two protected groups. Mathematically we can write this as the following two equations:

$$P(\hat{Y}=0|Y=0,G=a) = P(\hat{Y}=0|Y=0,G=b), \tag{13.9}$$

$$P(\hat{Y}=1|Y=1,G=a) = P(\hat{Y}=1|Y=1,G=b). \tag{13.10}$$

Based on the decision matrix, this can be computed as

$$\frac{a_{00}}{a_{00}+a_{01}} = \frac{b_{00}}{b_{00}+b_{01}}, \qquad \frac{a_{11}}{a_{10}+a_{11}} = \frac{b_{11}}{b_{10}+b_{11}}. \tag{13.11}$$

This criterion ensures that the decision has the same chance of being correct for both protected groups, or equivalently that the decision errors are equal. A potential drawback of this criterion is that because it requires the decision system to be tuned such that the error rates are equal across groups, it necessarily will perform worse for some group than it could.

Example 13.2 Equalized odds: Tiredness detection

A car manufacturer has created a new system that uses a camera to detect if the driver is tired, in order to take measures to avoid accidents. The system outputs a prediction $\hat{Y} = 1$ if it thinks the driver is tired. The system engineers are worried that the system might perform poorly for a protected group, so they examine the data they have gathered in their system test where they have also collected ground truth information, Y, about whether the driver was actually tired or not.

Their available data is summarized in the following table:

	$G = a$		$G = b$	
	$\hat{Y} = 0$	$\hat{Y} = 1$	$\hat{Y} = 0$	$\hat{Y} = 1$
$Y = 0$	70	15	30	10
$Y = 1$	20	130	5	30

First, let us examine the fraction of negative decisions among the negative cases, i.e., how large a fraction of non-tired drives were classified as non-tired:

$$\text{Group } a: \quad \frac{70}{90+15} \approx 74\ \% \tag{13.12}$$

$$\text{Group } b: \quad \frac{30}{30+10} = 75\ \% \tag{13.13}$$

These percentages seem fairly close (pun intended). Next, we look at the fraction of positive decisions among the positive cases, i.e., how large a fraction of tired drives were

classified as tired:

$$\text{Group } a: \quad \frac{130}{20+130} \approx 87\ \% \tag{13.14}$$

$$\text{Group } b: \quad \frac{30}{5+30} \approx 86\ \% \tag{13.15}$$

Again, the percentages match approximately. This indicates that the tiredness detector is fair according to the equalized odds criterion.[3]

[3] In this example we assess the fairness of the decision process by comparing sample proportions, but since we are actually interested in how fair the system is in the general population, we should take the uncertainty associated with the (small) sample size into account, for example by computing confidence intervals for the proportions.

13.3.3 *Equalized opportunity*

A third fairness criterion we will consider here is called the *equalized opportunity* criterion. According to this criterion, the probability that the decision is correct among cases where the decision is positive is required to be the same across protected groups. Mathematicall this can be written as

$$P(Y=1|\hat{Y}=1, G=a) = P(Y=1|\hat{Y}=1, G=b). \tag{13.16}$$

Based on the decision matrix, this criterion can be evaluated as

$$\frac{a_{11}}{a_{01}+a_{11}} = \frac{b_{11}}{b_{01}+b_{11}}. \tag{13.17}$$

The equalized opportunity criterion considers only whether there is a balanced chance of making the correct decision among cases for which the decision is positive. The cases where the decision is negative are not taken into account.

Example 13.3 Equalized opportunity: Granting a loan

A bank wants to examine if their procedure for granting or denying loans is fair according to the equalized opportunity criterion. Let $\hat{Y} = 1$ denote the decision to grant the loan, and let $Y = 1$ denote that the customer was able to repay the loan on time. Among the 10 000 most recent loans granted,

7500 were granted to customers from protected group a of which 7180 repaid on time, and

2500 were granted to customers from protected group b of which 2475 repaid.

The proportion of correct decisions (the loan was re-

paid) among positive decisions (the loan was granted) in the two groups are

$$\text{Group } a: \quad \frac{7180}{7500} \approx 96 \pm 0.5\ \% \tag{13.18}$$

$$\text{Group } b: \quad \frac{2470}{2500} \approx 99 \pm 0.4\ \% \tag{13.19}$$

Here we have included a confidence interval for the two proportions to make the statistical uncertainty clear.

Because the proportions are different for the two groups, the bank does not grant its loans in a fair manner according to the equalized opportunity criterion. To balance the opportunity, the bank could decide to grant more loans to group b, thereby increasing their risk, and/or grant fewer loans to group a (thereby reducing their risk).

Problems

1. Show that the demographic parity criterion in Eq. 13.4 implies $P(\hat{Y}=0|G=a) = P(\hat{Y}=0|G=b)$.
2. Is the tiredness detector in Example 13.2 fair according to the demographic parity criterion?

Solutions

1. We have $P(\hat{Y} = 1|G = a) = 1 - P(\hat{Y} = 0|G = a)$ since the probabilities must sum to one. Thus, we can write

$$\begin{aligned} P(\hat{Y}=1|G=a) &= P(\hat{Y}=1|G=b) && \Leftrightarrow \\ 1 - P(\hat{Y}=1|G=a) &= 1 - P(\hat{Y}=1|G=b) && \Leftrightarrow \\ P(\hat{Y}=0|G=a) &= P(\hat{Y}=0|G=b). \end{aligned}$$

2. Let us compute the fraction of drivers detected as tired in the two groups:

$$\text{Group } a: \quad \frac{130+15}{70+15+20+130} \approx 62\ \% \tag{13.20}$$

$$\text{Group } b: \quad \frac{10+30}{30+10+5+30} \approx 53\ \% \tag{13.21}$$

   This indicates that the tiredness detector is not fair according to demographic parity; although, due to the small sample size, the difference in percentages is not statistically significant. Anyway, demographic parity might not be desirable in this case, since drivers from group a appear to be more tired overall, which the decision system reflects.