MATHEMATICS of MACHINE LEARNING

by

MARTIN LOTZ
Mathematics Institute
The University of Warwick

March 2023

Contents

Preface

This book grew out of a course of lectures delivered at Warwick University from 2019 to 2023. The main target audience were third year Mathematics undergraduates, but the lectures were also attended by fourth year and postgraduate students, as well as by students on Statistics, Computer Science, and Physics degree programmes. The focus on the course has been on developing the underlying mathematical ideas rigorously, rather than on applications and implementation, the latter being already covered elsewhere.

One major challenge in writing a book on such a vast and rapidly evolving subject is selecting an appropriate range of topics. In this book, I decided to cover three broad areas: statistical learning theory, optimization and deep learning. Statistical learning theory provides the theoretical foundation on which machine learning is built. Optimization has traditionally been closely related to engineering and operations research, but it has become a crucial part of the machine learning canon. Motivated by the demands of large and high-dimensional data sets, the focus has shifted towards stochastic algorithms. The third part of the book, Deep Learning, introduces artificial neural networks, or multilayer perceptrons, and discusses a selection of topics in that context. Needless to say, many important topics have been omitted. The hope is that the material presented will at least leave the reader well prepared to explore other topics that were not covered, as well as further applications of the material, on her or his own.

In terms of prerequisites, the book assumes a solid background in undergraduate mathematics, such as that taught at Warwick University in the first two years of the Mathematics degree programmes. This includes linear algebra and matrix calculus, analysis (both univariate and multivariate), and basic probability and statistics. A summary of the material that the reader should be familiar with is given in the appendix, and the reader is encouraged to browse through this section. Beyond this, the book does not assume any more advanced mathematical concepts. Additional mathematical topics such as concentration of measure and information theory are introduced as needed. Some thought has been given as to whether to include code. While this approach would be justified in more established fields such as numerical analysis, the feeling is that any examples that use a particular computing framework might be dated by the time you are reading this. I trust that the reader has the intellectual ability to complement this course with more practically oriented resources that use state-of-the-art technology.

Acknowledgments

I would like to thank all the students on the programme for contributing to the success of this project and for their valuable feedback and suggestions. In particular, I would like to thank Marion Dugue, Peter Fazekas, Sanjif Shanmugavelu, Ziad Fakhoury, Sam Goymer, Dan LaBraca, Paul Lezeau, Connor Mattison, Peter Job, Ilona Holostova, Rafał Szlendak, James Palmer and Thomas Poole. I would also like to thank my teaching assistants Haoran Ni, Tsz Fung Yu and Yijie Zhou for very valuable feedback.

Related literature

This book is indebted to a large variety of sources, including textbooks, research papers, and online lecture notes. Each chapter ends with some brief bibliographic notes. These are not intended to be comprehensive, but will give the reader enough pointers to get deeper into the subject on their own.

A note on software

This book deliberately does not put an emphasis on code. Even though at the time of writing Python is the de-facto programming language for machine learning, there is no standard set of packages. With the notorious backward incompatibility associated with Python packages, any specific choice will carry the danger that the code will quickly become dated, and possibly obsolete, by the time you are reading this. Finally, the focus of this book is really to outline the main underlying principles, and including specific code would not only make this book longer but also distract from its mission.

All that being said, I consider it **very important** to complement this course with some practical programming project in order to get the most out of it. The reader will find no shortage of online tutorials on implementing many of the ideas presented in this book, and in particular those from Part III.

Notation

We will use bold letters $\boldsymbol{x}, \boldsymbol{y}, \dots$ to denote vectors in some $\mathbb{R}^d$. Upper indices such as $\boldsymbol{x}^i$ are denoted to enumerate vectors, while x_i denotes the i-th entry of $\boldsymbol{x}$. We will follow the common convention in statistics of using $\boldsymbol{x}$ and $\boldsymbol{y}$ for inputs and outputs, respectively. This leads to the situation that optimization methods will usually operate on the parameters, or weights $\boldsymbol{w}$ of a function. We will use both the notation $\boldsymbol{x}^\intercal \boldsymbol{y}$ and $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ for the Euclidean inner product of two vectors, depending on whether we emphasize a numerical point of view (as in the description of a method or an algorithm) or a more conceptual point of view (where the Euclidean inner product could, in principle, be replaced by any other scalar product). Random variables will be denoted by upper case letters $X, Y, \dots$, probabilities using $\mathbb{P}$ and the expectation is written as $\mathbb{E}$. In most cases it will be clear with respect to which random quantity the expectation is taken, and it will by indicated by a subscript if this is not the case. We will differentiate between column and row vectors only when relevant for matrix computations. We use the shorthand notation $[n] := \{1, \dots, n\}$.

How to read this book

The three parts of this book are, to a large extent, independent (except for the fact that neural networks, introduced in Chapter 21, are trained using stochastic gradient descent, introduced in Chapter 20, but the detailed analysis of this algorithm is not necessary for the rest of Part III). The chapters in Part I and II depend on each other incrementally, while the chapters in Part III all depend on Chapter 21, but are otherwise independent of each other. Look at the table of contents and pick out what interests you.

1

Introduction

"No computer has ever been designed that is ever aware of what it's doing; but most of the time, we aren't either."
— Marvin Minsky, 1927-2016

Learning is the process of transforming information and experience into knowledge and understanding. Knowledge and understanding are measured by the ability to perform certain tasks independently. **Machine Learning** is therefore the study of algorithms and models for computer systems to carry out certain tasks independently, based on the results of a learning process. Learning tasks can range from solving simple classification problems, such as handwritten digit recognition, to more complex tasks, such as medical diagnosis or driving a car.

Machine learning is part of the broader field of **Artificial Intelligence**[1], but distinguishes itself from more traditional approaches to problem solving, in which machines follow a strict set of rules they are provided with. As such, it is most useful for tasks such as pattern recognition, that may be simple for humans but where precise rules are hard to come by with, or for tasks that allow for simple rules, but where the complexity of the problem makes any rule-based approach computationally infeasible. An illustrative example of the latter is the success of DeepMind's AlphaGo and AlphaZero at achieving super-human performance at the board games Chess and Go. Even though the rules of Go are extremely simple, the task of beating the best human players seemed impossible not long ago due to the daunting complexity that ensues from the number of possible positions.

A General Framework for Learning

Machine learning lies at the intersection of approximation theory, probability theory, statistics, and optimization theory. We illustrate the interplay of these fields with a few examples.

A common problem in machine learning is to come up with (**learn**) a function

$$h\colon \mathcal{X} \to \mathcal{Y},$$

where $\mathcal{X}$ is a space of **inputs** or **features**, and $\mathcal{Y}$ consists of **outputs** or **responses**. The input space $\mathcal{X}$ is modelled as a metric space (such as $\mathbb{R}^d$), and the inputs could represent images, texts, emails, amino acid sequences, networks, financial time series, or demographic data. The output could consist of **quantitative** values, such as a temperature or the amount of a certain substance in the body, or of

[1] The term AI is somewhat overused, see [42] for a critical discussion.

qualitative or **categorical** values, such as $\{\text{YES}, \text{NO}\}$ or $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. The first type of problem is usually called **regression**, while the latter is called **classification**. The function h is sometimes called a **hypothesis**, a **predictor**, or a **classifier**. A classifier h that takes only two values (typically 0 and 1, or -1 and $+1$) is called a **binary classifier**. In a machine learning scenario, a function h is chosen from a predetermined set of functions $\mathcal{H}$, called the **hypothesis space**.

Machine learning problems can be subdivided into supervised and unsupervised learning problems. In **supervised learning**, we have at our disposal a collection of input-output data pairs

$$\{(\boldsymbol{x}^i, \boldsymbol{y}^i)\}_{i=1}^n \subset \mathcal{X} \times \mathcal{Y},$$

and the goal is to learn a function $h\colon \mathcal{X} \to \mathcal{Y}$ from this data. The collection of pairs $\{(\boldsymbol{x}^i, \boldsymbol{y}^i)\}_{i=1}^n$ is called the **training set**. In **unsupervised learning**, one does not have access to a training set. The prototypical example of an unsupervised learning task is **clustering**, where the tasks is to subdivide a given data set into groups based on similarities.

Example 1.1 (Digit recognition)**.** Given a dataset of pixel matrices, each representing a grey-scale image, with associated **labels** telling us for each image the number it represents, the task is to use this data to train a computer program to recognise new numbers (see Figure 1.1). Such classification tasks are often carried out using **deep neural networks**, which constitute a powerful class of non-linear functions.

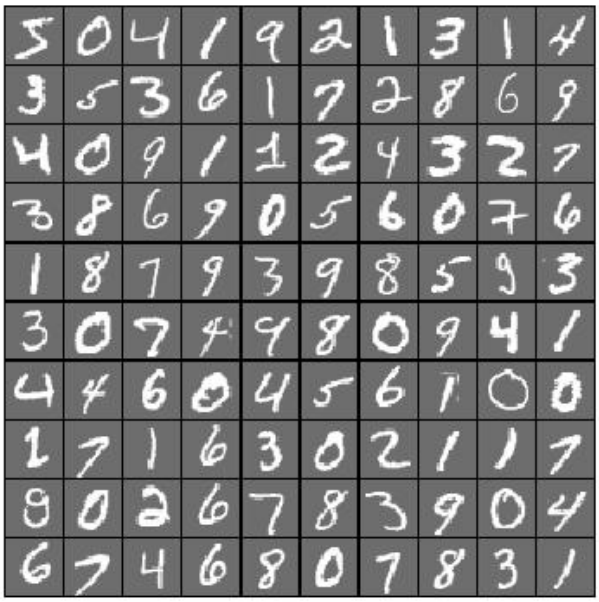


Figure 1.1: The MNIST (Modified National Institute of Standards and Technology) dataset is a large collection of images of hand-written digits, and is a frequently used benchmark in machine learning.

Example 1.2. (Clustering) In clustering applications, one observes data $\{\boldsymbol{x}^i\}_{i=1}^n$, and the goal is to subdivide the data into a number of distinct groups based on similarity, where similarity is measured using a distance function. Figure 1.2 shows an example of an artificial clustering problem and a possible solution. The notion of distance used depends on the application. For example, for binary sequences or DNA sequences one can use the *Hamming metric*, which simply counts the positions at which two

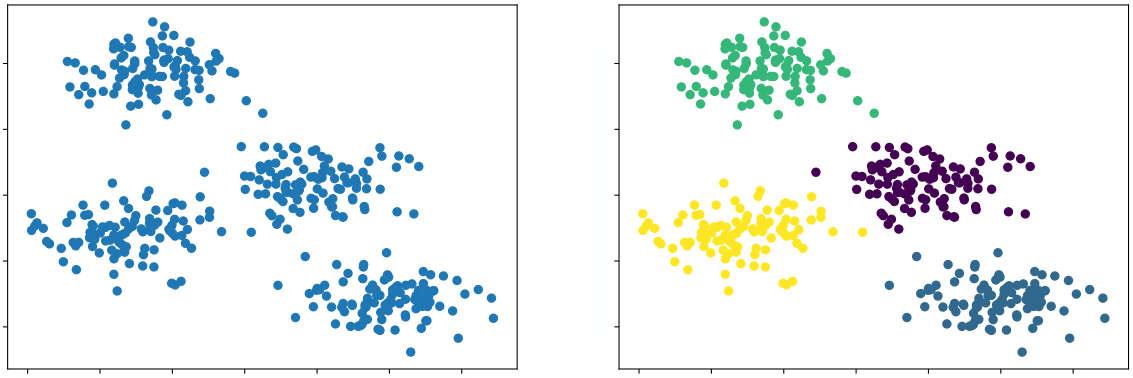


Figure 1.2: A collection of random points on the plane. The image on the right shows the four clusters as determined by the k-means algorithm.

sequences differ. Clustering is used extensively in genetics, biology and medicine. For example, clustering can be used to identify groups of genes (segments of DNA with a function) that encode proteins which are part of a common biological pathway. Other uses of clustering are market segmentation and community detection in networks.

Approximation Theory

One often makes the simplified assumption that the observed training data comes from an unknown function $f\colon \mathcal{X} \to \mathcal{Y}$. The goal is to *approximate* the function f with a function h from a hypothesis class $\mathcal{H}$, based only on the knowledge a finite set of samples $\{(\boldsymbol{x}^i, \boldsymbol{y}^i)\}_{i=1}^n$, where we assume $\boldsymbol{y}^i \approx f(\boldsymbol{x}^i)$ for $i \in [n] := \{1, \ldots, n\}$. Which class of functions is adequate depends on the application at hand, as well as on computational and statistical considerations. In many cases a linear function will do well, while in other situations polynomials or more complex functions, like neural networks, are better suited.

Example 1.3. (Linear regression) Linear Regression is the problem of finding a relationship of the form

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p,$$

where the $X_1, \ldots, X_p$ are **covariates** that describe certain characteristics of a system and Y is the **response**. Given a set of input-output pairs $(\boldsymbol{x}^i, y^i)$, arranged in a matrix $\boldsymbol{X}$ and a vector $\boldsymbol{y}$, we can *guess* the correct $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^\intercal$ by solving the *least-squares* optimization problem

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \, \|\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}\|_2^2.$$

Figure 1.3 shows an example of linear regression.

Example 1.4. (Text classification) In text classification, the task is to decide to which of a given set of categories a given text belongs. The training data consists of a *bag of words*: this is a large sparse matrix, whose columns represent words and the rows represent articles, with the (i, j)-th entry containing the number of times word j is contained in text i.

	Goal	Soup
Article 1	5	0
Article 2	1	7

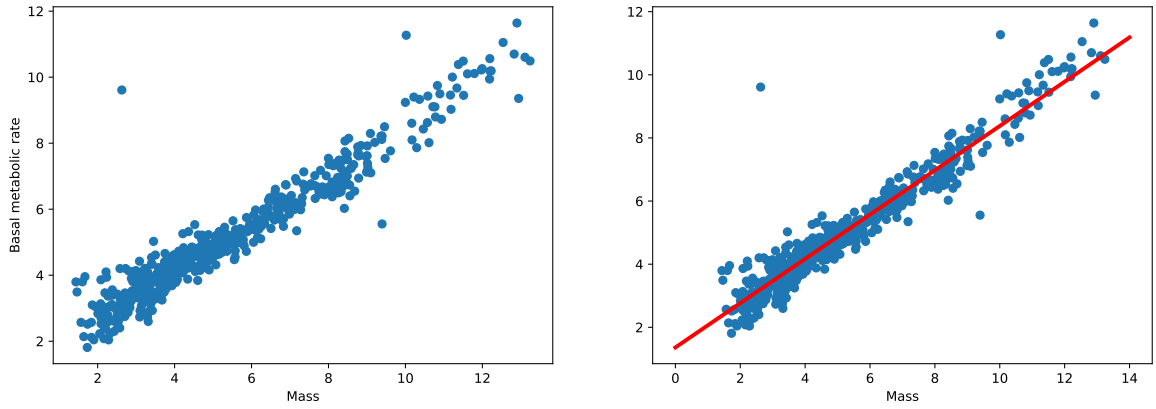


Figure 1.3: The relationship of mass to the logarithm of the basal metabolic rate in mammals. The data consists of 573 samples taken from the PanTHERA database, and the example featured in the episode Size Matters of the BBC series Wonders of Life. The right images shows the regression line determined using linear least squares.

For example, in the above set we would classify the first article as "Sports" and the second one as "Food". One such training dataset is the Reuters Corpus Volume I (RCV1), an archive of over 800,000 categorised newswire stories. A typical binary classifier for such a problem would be a **linear classifier** of the form

$$h(\boldsymbol{x}) = \boldsymbol{w}^\intercal \boldsymbol{x} + b,$$

with $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$. Given a text, represented as a row of the dataset $\boldsymbol{x}$, it is classified into one of two classes $\{+1, -1\}$, depending on whether $h(\boldsymbol{x}) > 0$ or $h(\boldsymbol{x}) < 0$.

Example 1.5. (Deep Neural Networks) Artificial neural networks[2] are functions of the form

$$f_\ell \circ f_{\ell-1} \circ \cdots \circ f_1,$$

where each f_k is the component-wise composition of an **activation function** σ with a linear map $\mathbb{R}^{d_{k-1}} \to \mathbb{R}^{d_k}, \boldsymbol{x} \mapsto \boldsymbol{W}^k \boldsymbol{x} + \boldsymbol{b}^k$:

$$f_k(\boldsymbol{x}) = \sigma(\boldsymbol{W}^k \boldsymbol{x} + \boldsymbol{b}^k).$$

An activation function could be the **sigmoid** $\sigma(x) = 1/(1 + \mathrm{e}^{-x})$, which takes values in $(0, 1)$, and which can be interpreted as "selecting" certain coordinates of a vector depending on whether they are positive or negative. The coefficients w_{ij}^k of the matrix $\boldsymbol{W}^k$ in each layer are the **weights**, while the entries of $\boldsymbol{b}^k$ are called the **bias** terms. The weights and bias terms are to be adapted in order to fit observed input-output pairs. A neural network is usually represented as a graph, see Figure 1.4. Neural networks have been extremely successful in pattern recognition, and are widely used in applications ranging from natural language processing to machine translation and medical diagnostics.

One of the earliest theoretical results in approximation theory is a theorem by Weierstrass that shows that we can approximate any continuous function on an interval to arbitrary precision by polynomials.

Theorem 1.6 (Weierstrass)**.** *Let f be a continuous function on $[a, b]$. Then for any $\epsilon > 0$ there exists a polynomial $p(x)$ such that*

$$\|f - p\|_\infty = \max_{x \in [a,b]} |f(x) - p(x)| \leq \epsilon.$$

[2] In what follows, we will drop the 'artificial'. An alternative designation is multilayer perceptron.

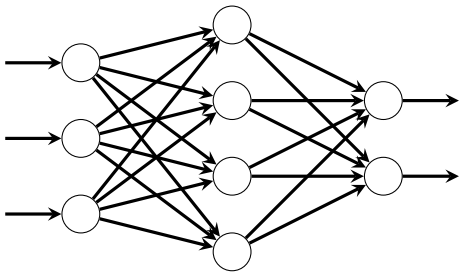


Figure 1.4: A neural network. Each layer correspond to applying a linear map to the outputs of the previous layer, followed by an activation function. Each arrow represents a *weight*. For example, the transition from the first layer to the second is a map $\mathbb{R}^3 \to \mathbb{R}^4$, and the weight associated with the arrow from the second node in layer 1 to the first node in layer 2 is the $(1, 2)$-entry in the matrix defining the corresponding linear map.

This theorem is remarkable because it shows that we can approximate any continuous function on a compact interval using only a *finite* number of parameters, the coefficients of a polynomial. The problem with this theorem is that it gives no bound on the size of the polynomial, which can be rather large. It also does not give a procedure of actually computing such an approximation, let alone finding one efficiently. We will see that neural networks have the same approximation properties, i.e., every continuous function on an interval can be approximated to arbitrary accuracy by a neural network. There are many variations on such results for approximating a class of functions through a simpler class, and we will be interested in cases where such approximations can be efficiently computed. One way of finding good approximating functions is by using optimization methods.

Optimization

The notion of *best fit* is formalized by using a **loss function**. A loss function $L\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ measures the mismatch between a prediction on a given input $\boldsymbol{x} \in \mathcal{X}$ and an element $\boldsymbol{y} \in \mathcal{Y}$. The **empirical risk** of a function $h\colon \mathcal{X} \to \mathcal{Y}$ is the average loss $L(h(\boldsymbol{x}^i), \boldsymbol{y}^i)$ over the training data,

$$\hat{R}(h) := \frac{1}{n} \sum_{i=1}^{n} L(h(\boldsymbol{x}^i), \boldsymbol{y}^i)$$

One would then aim to find a function h among a set $\mathcal{H}$ of candidates that minimizes the loss when applying the function to the training data:

$$\underset{h \in \mathcal{H}}{\text{minimize}}\, \hat{R}(h). \tag{1.1}$$

Problem (1.1) is an **optimization problem**. Minimizing over a set of functions my look abstract, but functions in $\mathcal{H}$ are typically parametrized by few parameters. For example, when the class $\mathcal{H}$ consists of linear functions of the form $\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$, as in Example 1.3, then the optimization problem (1.1) amounts to minimizing a function over $\mathbb{R}^{p+1}$. In the case of neural networks, Example 1.5, one optimizes over the weights w_{ij}^k and bias terms b_i^k.

The form of the loss function depends on the problem at hand and is usually derived from statistical considerations. Two common candidates are the **square error** for regression problems, which applied to a function $h\colon \mathbb{R}^d \to \mathbb{R}$ takes the form

$$L(h(\boldsymbol{x}), y) = (h(\boldsymbol{x}) - y)^2,$$

and the **indicator loss function**, which takes the general form

$$L(h(\boldsymbol{x}), y) = \mathbf{1}\{h(\boldsymbol{x}) \neq y\} = \begin{cases} 1 & \text{if } h(\boldsymbol{x}) = y \\ 0 & \text{if } h(\boldsymbol{x}) \neq y \end{cases}.$$

As this function is not continuous, in practice one often encounters continuous approximations. A binary classifier is often implemented by a function $h\colon \mathcal{X} \to [0,1]$ that provides a *probability* of an input belonging to a class. If $h(\boldsymbol{x}) > 1/2$, then $\boldsymbol{x}$ is assigned to class 1, while if $h(\boldsymbol{x}) \leq 1/2$, then $\boldsymbol{x}$ is assigned to class 0. A common loss function for this setting is the **log-loss** function, or **cross-entropy**,

$$\begin{aligned} L(h(\boldsymbol{x}), y) &= -y \log(h(\boldsymbol{x})) - (1-y)\log(1-h(\boldsymbol{x})) \\ &= \begin{cases} -\log(h(\boldsymbol{x})) & \text{if } y = 1 \\ -\log(1-h(\boldsymbol{x})) & \text{if } y = 0 \end{cases}. \end{aligned} \tag{1.2}$$

The function is designed to take on large values if the class predicted by $h(\boldsymbol{x})$ does not match y, and can be interpreted in the context of maximum-likelihood estimation.

Finding or approximating a minimizer of a function falls into the realm of **numerical optimization**. While for linear regression we can solve the relevant optimization problem (least-squares minimization) in closed form, for more involved problems such as neural networks we use optimization algorithms such as **gradient descent**: we start with an initial guess and try to minimize our function by taking steps in direction of *steepest descent*, that is, along the negative gradient of the function. In the case of composite functions such as neural networks, computing the gradient requires the chain rule, which leads to the famous **backpropagation** algorithm for training a neural network that will be discussed in detail.

There are many challenges related to optimization models and algorithms. The function to be minimized may have many *local* minima or saddle points, and algorithms that look for minimizers may find any one of these, instead of a global minimizer. The functions to be minimized may not be differentiable, and methods from the field of **non-smooth optimization** come into play. One of the biggest challenges for optimization algorithms in the context of machine learning, however, lies in the particular form of the objective function: it is given as a sum of many terms, one for each data point. Evaluating such a function and computing its gradient can be time and memory consuming. An old class of algorithms that includes **stochastic gradient descent** circumvents this issue by not computing the gradient of the whole function at each step, but only of a small random subset of the terms. These algorithms work surprisingly well, considering that they do not make use of all the information available at every step.

Statistics

Suppose we have a binary classification task at hand, with $\mathcal{Y} = \{0, 1\}$. We could *learn* the following function from our training data $\{(\boldsymbol{x}^i, y^i)\}_{i=1}^n$:

$$h(\boldsymbol{x}) = \begin{cases} y^i & \text{if } \boldsymbol{x} = \boldsymbol{x}^i, \\ 1 & \text{otherwise.} \end{cases}$$

This is called **learning by memorization**, since the function simply memorizes the value y^i for every seen example $\boldsymbol{x}^i$. The empirical risk with respect to the unit loss function for this problem is

$$\hat{R}(h) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{h(\boldsymbol{x}^i) \neq y^i\} = 0.$$

Nevertheless, this is not a good classifier: it will not perform very well outside of the training set. This is an example of **overfitting**: when the function is adapted too closely to the seen data, it does not generalize to unseen data. The problem of **generalization** is the problem of finding a classifier that works well on unseen data.

To make the notion of generalization more precise, we assume that the training data points $(\boldsymbol{x}^i, \boldsymbol{y}^i)$ are realizations of a pair of random variables (X, Y), sampled from an (unknown) probability distribution on the product $\mathcal{X} \times \mathcal{Y}$. The variables X and Y are in general not independent (otherwise there would be nothing to learn), but are related by a relationship of the form $Y = f(X) + \epsilon$, where ϵ is a random perturbation with expected value $\mathbb{E}[\epsilon] = \mathbf{0}$. One could interpret the presence of the random noise ϵ as indicative of uncertainty or missing information. For example, when trying to predict a certain disease based on genetic markers, the genetic data might simply not carry enough information to always make a correct prediction. The function f is called the **regression function**. It is the conditional expectation of Y given a value of X,

$$f(\boldsymbol{x}) = \mathbb{E}[Y \mid X = \boldsymbol{x}].$$

Given a classifier $h \in \mathcal{H}$ and a loss function L, the **generalization risk** is the expected value

$$R(h) := \mathbb{E}[L(h(X), Y)].$$

If $L(h(\boldsymbol{x}), \boldsymbol{y}) = \mathbf{1}\{h(\boldsymbol{x}) \neq \boldsymbol{y}\}$ is the unit loss, then this is simply $\mathbb{P}(h(X) \neq Y)$, i.e., the probability of misclassifying a randomly chosen input-output pair. The training data can be modelled as sampling form n pairs of random variables

$$(X_1, Y_1), \ldots, (X_n, Y_n)$$

that are identically distributed and independent copies of (X, Y). Given a classifier h, the empirical risk becomes a random variable

$$\hat{R}(h) = \frac{1}{n} \sum_{i=1}^{n} L(h(X_i), Y_i).$$

The expected value of this random variable is

$$\mathbb{E}[\hat{R}(h)] = \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}[L(h(X_i), Y_i)] = \mathbb{E}[L(h(X), Y)] = R(h),$$

where we used the linearity of expectation and the fact that the (X_i, Y_i) are independent and identically distributed (i.i.d). The empirical risk $\hat{R}(h)$ is thus an **unbiased estimator** of the generalization risk $R(h)$.

Example 1.7. The loss function is often chosen so that the problem of empirical risk minimization becomes a **maximum likelihood** estimation problem. Consider the example where Y takes values in $\{0, 1\}$ with probability $\mathbb{P}(Y = 1 \mid X = \boldsymbol{x}) = f(\boldsymbol{x})$. Conditioned on $X = \boldsymbol{x}$, Y is a Bernoulli random variable with parameter $p = f(\boldsymbol{x})$, and the log-loss function (1.2) is precisely the negative **log-likelihood** function for the problem of estimating the parameter p.

When looking for a good hypothesis h, all we have at our disposal is the empirical risk function constructed from the training data. It turns out that the quality of an empirical risk minimizer $\hat{h}$ from a hypothesis class $\mathcal{H}$ can be measured by the **estimation error**, which compares the generalization risk of $\hat{h}$ to the smallest possible generalization risk in $\mathcal{H}$, and the **approximation error**, which measures how small the generalization risk can become within $\mathcal{H}$. There is usually a trade-off between these to errors: if the class of functions $\mathcal{H}$ is large, then it is likely to contain functions with small generalization risk and thus have small approximation error, but the empirical risk minimizer $\hat{h}$ is likely to "overfit" the data and not generalize well. On the other hand, if $\mathcal{H}$ is small (in the extreme case, consisting of only one

function), then the empirical risk minimizer is likely to be close to the best possible hypothesis in $\mathcal{H}$, but the approximation error will be large. Figure 1.5 shows an example in which data from a function with noise is approximated by polynomials of different degrees.

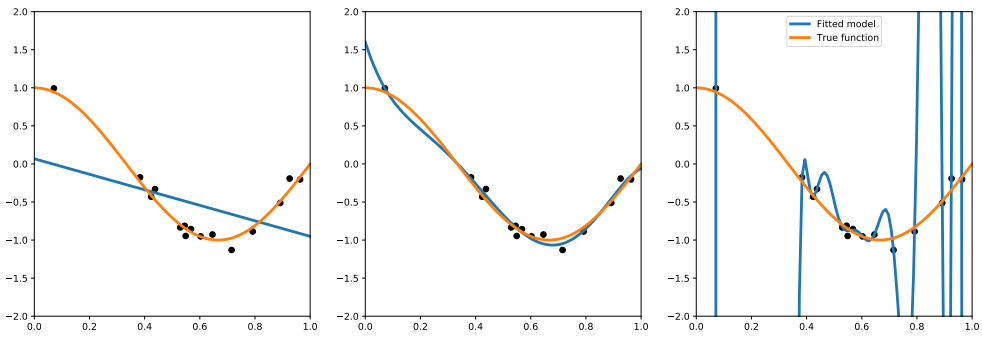


Figure 1.5: The data consists of 15 samples from the graph of a cosine function with added noise. The three displays show an approximation with a linear function, with a polynomial of degree 5, and with a polynomial of degree 15. The linear function has a large error on both the training set and in relation to the true function. The polynomial of degree 15, on the other hand, has zero error on the training data (a polynomial of degree d can fit $d+1$ points with distinct x-values exactly), but it will likely perform poorly on new data. This is an example of **overfitting**: more parameters in the model will not necessarily lead to a better performance on unseen data.

The field of **Statistical Learning Theory** aims to understand the relation between the generalization risk, the empirical risk, the capacity of a hypothesis class $\mathcal{H}$, and the number of samples n. In particular, notions such as the capacity of a hypothesis class are given a precise meaning through concepts such as VC dimension, Rademacher complexity, and covering numbers.

Notes

The ideas from approximation theory, optimization and statistics that underlie modern machine learning are old. Linear regression was known to Legendre and Gauss. The Weierstrass Approximation Theorem was published by Weierstrass in [85], see [77, Chapter 6] for an account and more history on approximation theory. Neural networks go back to the seminal work by McCulloch and Pitts from 1943 [54], followed by Rosenblatt's Perceptron [64]. The term "Machine Learning" was first coined by Arthur Samuel in 1959 [68]; at the time, "Cybernetics" was still widely used. Gradient descent was known to Cauchy, and the most important algorithm for deep learning today, Stochastic Gradient Descent, was introduced by Robbins and Monro in 1951 [63]. The field of Statistical Learning Theory arose in the 1960s through seminal work by Vapnik and Chervonenkis, see [80] for an overview. For an account of mathematical learning theory, see [22].

Research in machine learning exploded in the 1990s, with striking new results and applications appearing at breathtaking pace. Still, apart from some of the more theoretical developments in learning theory and high-dimensional probability, these breakthroughs rarely relied on mathematics that was not available 50 years ago. So what has changed since the early days of cybernetics? The main reason for the sudden surge in popularity is the availability of vast amounts of data, and equally important, the computational resources to process the data. New applications have in turn led to new mathematical problems, and to new connections between various fields.

Part I

Statistical Learning Theory

2

Binary Classification

In this chapter we begin the study of statistical learning theory in the case of binary classification. Binary classifiers assign data to one of two classes. Even though we are often interested in more than two classes (for example, in digit identification), the insights gained from the binary case generalize easily.

Binary Classification

A **binary classifier** is a function

$$h\colon \mathcal{X} \to \mathcal{Y} = \{0, 1\},$$

where $\mathcal{X}$ is a space of features. The fact that we use $\{0, 1\}$ it not very important, and in many cases it will be convenient to consider classifiers taking values in $\{-1, 1\}$. Binary classifiers arise in a variety of applications. In medical diagnostics, for example, a classifier could take an image of a skin mole and determine if it is benign or if it is melanoma. A classifier can arise from a function $\mathcal{X} \to [0, 1]$ that assigns to every input $\boldsymbol{x}$ a probability p. If $p > 1/2$, then $\boldsymbol{x}$ is assigned to class 1 and otherwise to 0.

In the context of binary classification, we usually use the **unit loss function**

$$L(h(\boldsymbol{x}), y) = \mathbf{1}\{h(\boldsymbol{x}) \neq y\} = \begin{cases} 1 & h(\boldsymbol{x}) \neq y \\ 0 & h(\boldsymbol{x}) = y. \end{cases}$$

The unit loss does not distinguish between **false positives** and **false negatives**. A false positive is a pair $(\boldsymbol{x}, y)$ with $h(\boldsymbol{x}) = 1$ but $y = 0$, and a false negative is a pair for which $h(\boldsymbol{x}) = 0$ but $y = 1$. We would like to *learn* a classifier from observations

$$\{(\boldsymbol{x}^i, y^i)\}_{i=1}^n \subset \mathcal{X} \times \mathcal{Y}. \tag{2.1}$$

The classifier should not only match the data, but **generalize** in order to be able to classify unseen data. For this, we assume that the points in (2.1) are drawn from a probability distribution on $\mathcal{X} \times \mathcal{Y}$, and replace each data point $(\boldsymbol{x}^i, y^i)$ in (2.1) with a copy (X_i, Y_i) of a pair of random variables (X, Y) on $\mathcal{X} \times \mathcal{Y}$. We are after a classifier h such that the expected value of the **empirical risk**

$$\hat{R}(h) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{h(X_i) \neq Y_i\} \tag{2.2}$$

is small. We can write this expectation as

$$\begin{aligned}\mathbb{E}[\hat{R}(h)] &\overset{(1)}{=} \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[\mathbf{1}\{h(X_i)\neq Y_i\}]\\ &\overset{(2)}{=} \frac{1}{n}\sum_{i=1}^{n}\mathbb{P}(h(X_i)\neq Y_i)\\ &\overset{(3)}{=} \mathbb{P}(h(X)\neq Y) =: R(h),\end{aligned}$$

where (1) uses the linearity of expectation, (2) expresses the expectation of an indicator function as probability, and (3) uses the fact that all the (X_i, Y_i) are identically distributed. The function $R(h)$ is called the **generalization risk** or simply **risk**: it is the probability that the classifier gets something wrong.

Example 2.1. Assume that the distribution is such that Y is completely determined by X, that is, $Y = f(X)$. Then

$$R(h) = \mathbb{P}(h(X)\neq f(X)),$$

and $R(h) = 0$ if $h = f$ almost everywhere. If $\mathcal{X}$ is a finite or compact set with the uniform distribution, then $R(h)$ simply measures the proportion of the input space on which h fails to classify inputs correctly.

While for certain tasks such as image classification there may be a unique label to each input, in general this need not be the case. In many applications, the input does not carry enough information to completely determine the output. Consider, for example, the case where $\mathcal{X}$ consists of whole genome sequences and the task is to predict hypertension (or any other condition) from it. The genome clearly does not carry enough information to make an accurate prediction, as other factors also play a role. To account for this lack of information, define the **regression function**

$$f(X) = \mathbb{E}[Y|X] = 1\cdot\mathbb{P}(Y=1|X) + 0\cdot\mathbb{P}(Y=0|X) = \mathbb{P}(Y=1|X).$$

It is sometimes convenient to separate the dependency of Y on X into a deterministic part and random noise,

$$Y = f(X) + E.$$

In this cases we have $\mathbb{E}[E|X] = 0$, because

$$f(X) = \mathbb{E}[Y|X] = \underbrace{\mathbb{E}[f(X)|X]}_{=f(X)} + \mathbb{E}[E|X].$$

The Bayes classifier

While in Example 2.1 we could choose (at least in principle) $h(\boldsymbol{x}) = f(\boldsymbol{x})$ and get $R(h) = 0$, in the presence of noise this is not possible. However, we can define a classifier by setting

$$h^*(\boldsymbol{x}) = \begin{cases} 1 & f(\boldsymbol{x}) > \frac{1}{2}\\ 0 & f(\boldsymbol{x}) \leq \frac{1}{2},\end{cases}$$

We call this the **Bayes classifier**. Note that the Bayes classifier can be interpreted as a maximum a posteriori (MAP) estimator:

$$h^*(\boldsymbol{x}) = \arg\max_y \mathbb{P}(Y = y|X = \boldsymbol{x}).$$

The following result shows that this is the best possible classifier.

Theorem 2.2. *Let $h\colon \mathcal{X} \to \mathcal{Y}$ be any binary classifier. Then*

$$R(h) - R(h^*) = \mathbb{E}[|2f(X) - 1| \cdot \mathbf{1}\{h(X) \neq h^*(X)\}].$$

In particular, the Bayes classifier h^* satisfies

$$R(h^*) = \inf_h R(h),$$

where the infimum is over all measurable h. Moreover, $R(h^*) \leq 1/2$.

The difference $\mathcal{E}(h) = R(h) - R(h^*) \geq 0$ is called the **excess risk** of h.

Proof. Let h be any classifier. To compute the risk $R(h)$, we first condition on X and then average over X:

$$R(h) = \mathbb{E}[\mathbf{1}\{h(X) \neq Y\}] = \mathbb{E}\left[\mathbb{E}[\mathbf{1}\{h(X) \neq Y\}|X]\right]. \tag{2.3}$$

For the inner expectation, we have

$$\begin{aligned}
\mathbb{E}[\mathbf{1}\{h(X) \neq Y\}|X] &= \mathbb{E}[\mathbf{1}\{h(X) = 1, Y = 0\} + \mathbf{1}\{h(X) = 0, Y = 1\}|X] \\
&= \mathbb{E}[(1 - Y)\mathbf{1}\{h(X) = 1\}|X] + \mathbb{E}[Y\mathbf{1}\{h(X) = 0\}|X] \\
&\overset{(1)}{=} \mathbf{1}\{h(X) = 1\}\,\mathbb{E}[(1 - Y)|X] + \mathbf{1}\{h(X) = 0\}\,\mathbb{E}[Y|X] \\
&= \mathbf{1}\{h(X) = 1\}(1 - f(X)) + \mathbf{1}\{h(X) = 0\}f(X).
\end{aligned}$$

To see why (1) holds, recall that the random variable $\mathbb{E}[Y\mathbf{1}\{h(X) = 0\}|X]$ takes values $\mathbb{E}[Y\mathbf{1}\{h(x) = 0\}|X = \boldsymbol{x}]$, and will therefore only be non-zero if $h(x) = 0$. We can therefore pull the indicator function out of the expectation. Hence, using (2.3),

$$\begin{aligned}
R(h) &= \mathbb{E}[\mathbf{1}\{h(X) = 1\}(1 - f(X)) + \mathbf{1}\{h(X) = 0\}f(X)] \\
&= \mathbb{E}[1 - f(X) + (2f(X) - 1) \cdot \mathbf{1}\{h(X) = 0\}].
\end{aligned} \tag{2.4}$$

Therefore, for the difference $R(h) - R(h^*)$ we get

$$R(h) - R(h^*) = \mathbb{E}[(2f(X) - 1) \cdot (\mathbf{1}\{h(X) = 0\} - \mathbf{1}\{h^*(X) = 0\})]$$

Going through all the possible combinations of values $(h(\boldsymbol{x}), h^*(\boldsymbol{x})) \in \{0, 1\}^2$, we arrive at the case distinction

$$\mathbf{1}\{h(\boldsymbol{x}) = 0\} - \mathbf{1}\{h^*(\boldsymbol{x}) = 0\} = \begin{cases} 1 & \text{if } h(\boldsymbol{x}) = 0, h^*(\boldsymbol{x}) = 1 \\ -1 & \text{if } h(\boldsymbol{x}) = 1, h^*(\boldsymbol{x}) = 0 \\ 0 & \text{if } h(\boldsymbol{x}) = h^*(\boldsymbol{x}) \end{cases}$$

Moreover, $h^*(\boldsymbol{x}) = 1$ if and only if $2f(\boldsymbol{x}) - 1 > 0$, and $h^*(\boldsymbol{x}) = 0$ if and only if $2f(\boldsymbol{x}) - 1 \leq 0$. Hence,

$$(2f(\boldsymbol{x}) - 1) \cdot (\mathbf{1}\{h(\boldsymbol{x}) = 0\} - \mathbf{1}\{h^*(\boldsymbol{x}) = 0\}) = |2f(\boldsymbol{x}) - 1| \cdot \mathbf{1}\{h(\boldsymbol{x}) \neq h^*(\boldsymbol{x})\},$$

and taking the expectation gives the desired characterization. From (2.4) we also get

$$\begin{aligned}
R(h^*) &= \mathbb{E}[\mathbf{1}\{f(X) > 1/2\}(1 - f(X))] + \mathbf{1}\{f(X) \leq 1/2\}f(X)] \\
&= \mathbb{E}\left[\min\{f(X), 1 - f(X)\}\right] \leq \frac{1}{2},
\end{aligned}$$

which completes the proof. □

We have seen in Example 2.1 that the Bayes risk is 0 if Y is completely determined by X. At the other extreme, if the response Y does not depend on X at all, then the Bayes risk is $1/2$. This means that for every input, the best possible classifier consists of "guessing" without any prior information!

Example 2.3 (Gaussian Mixture)**.** Assume we have a list of house prices from two different neighbourhoods, let's call them 0 and 1 (or Coventry and Leamington Spa), and we would like to devise a method to "guess" the neighbourhood from the price. We assume that the prices in each neighbourhood are, after normalizing to account for intrinsic factors such as size, approximately normally distributed. There may also be more available houses in one neighbourhood than in the other, leading to an intrinsically higher probability of picking a house from one neighbourhood. Under these assumptions, we can model the distribution house prices as a distribution on $\mathcal{X} \times \mathcal{Y} = \mathbb{R} \times \{0, 1\}$ as follows. Let $Y \sim \mathrm{Ber}(p)$ be a Bernoulli random variable with parameter p, that is, $\mathbb{P}(Y = 1) = p$ and $\mathbb{P}(Y = 0) = 1 - p$. Let X be a real-valued random variable with density ρ, such that the conditional density of X given $Y = 1$ is a Gaussian density $\rho_1(x) := \rho_{Y=1}(x)$, and the conditional density of X given $Y = 0$ is a Gaussian density $\rho_0(x) := \rho_{Y=0}(x)$, see Figure 2.1 for an illustration.

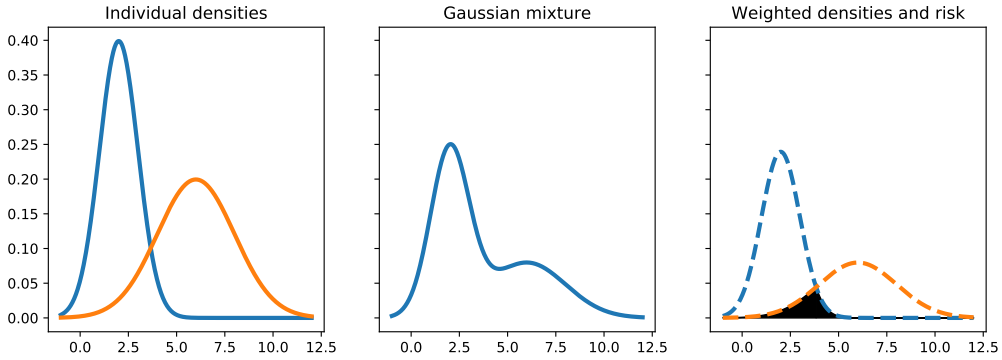


Figure 2.1: A mixture of Gaussians. If $p = 1/2$, the Bayes classifier assigns each sample to the density that is larger at this sample. The intersection of the area below the curves is the risk $R(h^*)$. If $p \neq 1/2$, then the densities are weighted.

.

We can think of the data generating process as first sampling a value $y \in \{0, 1\}$ from Y, and then sampling x from a Gaussian distribution with density ρ_y. The goal is to find a binary classifier $h \colon \mathbb{R} \to \{0, 1\}$ that assigns x to one of the two distributions that generated the data. To get the Bayes classifier, we use Bayes' rule to compute the regression function:

$$f(x) = \mathbb{P}(Y = 1 \mid X = x) = \frac{\rho_1(x)\mathbb{P}(Y = 1)}{\rho(x)} = \frac{p \cdot \rho_1(x)}{(1 - p)\rho_0(x) + p\rho_1(x)}.$$

We can rearrange:

$$\frac{p \cdot \rho_1(x)}{(1 - p)\rho_0(x) + p\rho_1(x)} > \frac{1}{2} \quad \Leftrightarrow \quad \frac{\rho_1(x)}{\rho_0(x)} > \frac{1 - p}{p}.$$

Therefore, the Bayes classifier for the problem above is given by

$$h^*(x) = \begin{cases} 1 & \text{if } \frac{\rho_1(x)}{\rho_0(x)} > \frac{1-p}{p} \\ 0 & \text{else} \end{cases}$$

If $p = 1/2$, then this amounts to assigning x to the class y for which $\rho_y(x)$ is larger. The risk of this classifier is clearly non-zero.

No Free Lunch

In practice, a classifier $\hat{h}_n$ is constructed from **training data** $\{(\boldsymbol{x}^i, y^i)\}_{i=1}^n$. This can be formalized by setting

$$\hat{h}_n(\boldsymbol{x}) = \hat{g}_n(\boldsymbol{x}; (\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^n, y^n))$$

for a function $\hat{g}_n \colon \mathcal{X} \times (\mathcal{X} \times \mathcal{Y})^n \to \mathcal{Y}$. For example, we could construct $\hat{h}_n$ by minimizing the empirical risk $\hat{R}(h)$ over $\mathcal{H}$, that is

$$\hat{h}_n = \operatorname*{arg\,min}_{h \in \mathcal{H}} \hat{R}(h) = \operatorname*{arg\,min}_{h \in \mathcal{H}} \sum_{i=1}^{n} \mathbf{1}\{h(\boldsymbol{x}^i) \neq y^i\}.$$

Intuitively, we would expect that more samples allow us to construct classifiers that better approximate the Bayes classifier. To study this formally, we consider the classifier $\hat{h}_n$ as constructed from random samples, i.e.,

$$\hat{h}_n(\boldsymbol{x}) = \hat{g}_n(\boldsymbol{x}; (X_1, Y_1), \ldots, (X_n, Y_n)), \tag{2.5}$$

where the (X_i, Y_i) are i.i.d. samples from the given distribution on $\mathcal{X} \times \mathcal{Y}$. A sequence of classifiers $\{\hat{h}_n\}_{n \geq 0}$ as in (2.5) is called **consistent**, if $\mathcal{E}(\hat{h}_n) \to 0$ in probability and **strongly consistent** if $\mathcal{E}(\hat{h}_n) \to 0$ almost surely. A naive approach to construct an estimator is to construct an unbiased estimator of the regression function $f(\boldsymbol{x}) = \mathbb{E}[Y|X = \boldsymbol{x}]$, and then define $\hat{h}_n$ from this estimator just like we defined the Bayes classifier using f. It can be shown that such an estimator is **universally strongly consistent**, which means that it is strongly consistent for any data distribution.

Example 2.4. In Example 2.3 we studied the Bayes classifier for the problem of assigning data to one of two components of a Gaussian mixtures. We now look at how to arrive at an estimator based solely on the observation of training data $\{\boldsymbol{x}^i, y^i\}$. If we assume that a Gaussian mixture is a good model for the data, then we may use maximum likelihood or MAP to produce an estimate of the relevant parameters of the mixture density (in practice, one would employ standard algorithms such as Expectation-Maximization (EM) to estimate the parameters of the distribution). This would then provide us with an estimate of the regression function $f(\boldsymbol{x})$ as in Example 2.3, which in turn leads to a classifier $\hat{h}_n$ that allows us to classify new, unseen data (see Figure 2.2).

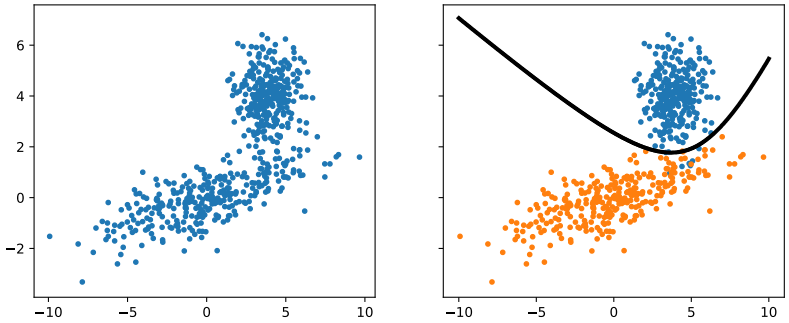


Figure 2.2: A Gaussian mixture model in two dimensions.

A subtle variation of the problem of consistency is whether it is possible to construct a series of classifiers $\hat{h}_n$ such that the excess risk $\mathcal{E}(\hat{h}_n)$ converges to 0 *uniformly* for all distributions. The following theorem shows that this is not the case. We denote by $\mathcal{D}^n$ the product distribution on $(\mathcal{X} \times \mathcal{Y})^n$ induced by a distribution $\mathbb{P}$ on the data space $\mathcal{X} \times \mathcal{Y}$.

Theorem 2.5 (No Free Lunch)**.** *Assume $|\mathcal{X}|$ is infinite. For a fixed n, consider a binary classifier $\hat{h}_n$ constructed from random data as in* (2.5)*. Let $\epsilon > 0$. Then there exists a probability distribution on $\mathcal{X} \times \mathcal{Y}$ such that*

$$\mathbb{E}_{\mathcal{D}^n}[\mathcal{E}(\hat{h}_n)] \geq \frac{1}{2} - \epsilon.$$

Proof. Choose m elements $S = \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^m\} \subset \mathcal{X}$ and consider a random variable X on $\mathcal{X}$ defined as

$$\mathbb{P}(X = \boldsymbol{x}) = \frac{1}{m}\mathbf{1}\{\boldsymbol{x} \in S\}.$$

Fix a binary (0-1) sequence $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_m)$, and consider the random variable Y on $\mathcal{Y} = \{0, 1\}$ with conditional distribution

$$\mathbb{P}(Y = y \mid X = \boldsymbol{x}_j) = \mathbf{1}\{y = \sigma_j\}$$

and $\mathbb{P}(Y = y \mid X = \boldsymbol{x}) = 0$ if $\boldsymbol{x} \notin S$. The random variable Y is thus completely determined by X, and the regression function is

$$f_{\boldsymbol{\sigma}}(\boldsymbol{x}_j) := \mathbb{E}[Y|X = \boldsymbol{x}_j] = \sigma_j$$

on S. In particular, the Bayes risk is $R(h^*) = 0$ and the excess risk is

$$\mathcal{E}(\hat{h}_n) = R(\hat{h}_n) = \mathbb{E}_X[\mathbf{1}\{\hat{h}_n(X) \neq f_{\boldsymbol{\sigma}}(X)\}].$$

We now consider all possible such distribution, as $\boldsymbol{\sigma}$ ranges over all 2^m possible sign vectors in $\{0, 1\}^m$. Specifically, we consider $\boldsymbol{\sigma}$ itself to be uniformly distributed over $\{0, 1\}^m$. Given X, the corresponding random variable $Y = f_{\boldsymbol{\sigma}}(X)$ thus depends on random X *and* random $\boldsymbol{\sigma}$. Taking the expectation with respect to this random sign vector and with respect to random data, we get:

$$\mathbb{E}_{\boldsymbol{\sigma}}\,\mathbb{E}_{\mathcal{D}^n}[\mathcal{E}(\hat{h}_n)] = \mathbb{E}[\mathbb{P}_{\boldsymbol{\sigma}}(\hat{h}_n(X; (X_1, f_{\boldsymbol{\sigma}}(X_1)), \ldots, (X_n, f_{\boldsymbol{\sigma}}(X_n))) \neq f_{\boldsymbol{\sigma}}(X))],$$

where we exchanged the order of the expectations, replaced the expectation of an indicator function with a probability, and the outer expectation is over all X and $X_1, \ldots, X_n$. If X is different from $X_1, \ldots, X_n$, then $f_{\boldsymbol{\sigma}}(X)$ is independent of the signs $f_{\boldsymbol{\sigma}}(X_i)$ for $i \in \{1, \ldots, n\}$, and takes the values 0 and 1 with equal probability. Conditioning on $X \neq X_j$ for $j \in \{1, \ldots, n\}$, we therefore get the bound

$$\begin{aligned}\mathbb{E}_{\boldsymbol{\sigma}}\,\mathbb{E}_{\mathcal{D}^n}[\mathcal{E}(\hat{h}_n)] &\geq \frac{1}{2}\mathbb{P}(X \neq X_j \colon j \in \{1, \ldots, n\}) \\ &= \frac{1}{2}\mathbb{P}(X \neq X_1)^n = \frac{1}{2}\left(1 - \frac{1}{m}\right)^n\end{aligned}$$

Given $\varepsilon > 0$, we can therefore find m such that the lower bound is great than $1/2 - \varepsilon$. Now since the *average* expected value $\mathbb{E}_{\mathcal{D}^n}[\mathcal{E}(\hat{h}_n)]$ over all distributions defined by a sign vector $\boldsymbol{\sigma} \in \{0, 1\}^m$ is greater than $1/2 - \varepsilon$, there surely has to be at least one sign vector for which which this is the case. This was to be shown. □

Notes

The classification setting described in this chapter is classic, see for example [80]. The binary classification problem may appear to be excessively simple, but once the problem of binary classification is understood, the underlying ideas generalize easily to more general situations. Our treatment of the Bayes classifier is based on [25, Chapter 2], and the version of the No Free Lunch Theorem presented here is based on [25, Section 7.1].

3

Concentration of Measure

The empirical risk is an average of random variables. By the law of large numbers, the average of independent and identically distributed random variables converges to the mean, represented by the generalization risk. In statistical learning, however, we are interested in *how well* the empirical risk (sample mean) approximates the generalization risk (mean) for a *fixed* number n of samples. Concentration of measure inequalities are bounds on the deviation of a random variable from its mean:

$$\mathbb{P}(|S_n - \mathbb{E}[S_n]| \geq t) \leq C(n,t),$$

where $S_n = (1/n)\sum_{i=1}^n X_i$ is the average of random variables and $C(n,t)$ is a bound that depends on t and the number of samples n. If each of the X_i has bounded variance, then Chebyshev's inequality yields a bound of order $O(1/nt^2)$, but this is often not good enough. When dealing with a set of classifiers $\mathcal{H}$, we have one random variable $S_n(h)$ for each classifier in $\mathcal{H}$, and we would like to bound the deviation over *all* $S_n(h)$. Assuming $\mathcal{H}$ is finite, this amounts to using the union bound:

$$\mathbb{P}\left(\sup_{h\in\mathcal{H}} |S_n(h) - \mathbb{E}[S_n(h)]| \geq t\right) \leq \sum_{h\in\mathcal{H}} \mathbb{P}(|S_n(h) - \mathbb{E}[S_n(h)]| \geq t) \leq |\mathcal{H}| \cdot C(n,t).$$

For this to be useful, we would like ta bound $C(n,t)$ that is *small*, preferably exponentially so in n and t. We present two inequalities that achieve this and operate under varying conditions: Hoeffding's inequality for sums of independent, bounded random variables, and McDiarmid's Inequality for functions with bounded differences.

The Cramér-Chernoff method

We begin with a very general deviation bound for random variables. Let X be a real-valued random variable and $\lambda \geq 0$. Then for any t,

$$\mathbb{P}(X \geq t) \leq \mathrm{e}^{-\lambda t}\,\mathbb{E}[\mathrm{e}^{\lambda X}]. \tag{3.1}$$

This inequality is a direct consequence of Markov's inequality, applied to the exponential random variable $\exp(\lambda X)$, and is left as an exercise. The best possible bound of type (3.1) can be formalized using the Cramér transform

$$\psi_X^*(t) = \sup_{\lambda\geq 0}(\lambda t - \psi_X(\lambda)), \tag{3.2}$$

where $\psi_X(\lambda)$ is the cumulant generating function, $\psi_X(\lambda) = \log \mathbb{E}[\mathrm{e}^{\lambda X}]$. With this terminology, we get the **Chernoff bound**

$$\mathbb{P}(X \geq t) \leq \mathrm{e}^{-\psi_X^*(t)}. \tag{3.3}$$

Note that $\psi_X(0) = 0$ implies that $\psi_X^*(t) \geq 0$. Moreover, if $t \leq \mathbb{E}[X] < \infty$, then one can show that $\psi_X^*(t) = 0$ and the Chernoff bound becomes trivial. We will typically apply the bound (3.3) to a **centred** random variable $\overline{X} = X - \mathbb{E}[X]$,

$$\mathbb{P}(X \geq \mathbb{E}[X] + t) \leq \mathrm{e}^{-\psi_X^*(t)}.$$

This is a one-sided bound: it bounds the probability of exceeding the expected value $\mathbb{E}[X]$ by more than t. One can similarly get a bound on $\mathbb{P}(X \leq \mathbb{E}[X] - t)$ by considering the normalized random variable $-\overline{X} = \mathbb{E}[X] - X$.

The most important case for us is when X is the average S_n of n independent and identically distributed (i.i.d.) random variables:

$$S_n = \frac{1}{n} \sum_{i=1}^{n} X_i.$$

In this case, the Cramér transform satisfies

$$\psi_{S_n}^*(t) = n \cdot \psi_X^*(t) \quad \Rightarrow \quad \mathbb{P}(S_n \geq t) \leq \mathrm{e}^{-n\psi_X^*(t)},$$

and, contingent on $\psi_X^*(t)$, we automatically have exponential decay in n.

Example 3.1. Let $X \sim \mathcal{N}(\mu, \sigma^2)$ be a normal distributed random variable with mean μ and variance σ^2. The moment generating function has the form

$$\mathbb{E}[\mathrm{e}^{\lambda(X-\mu)}] = \mathrm{e}^{\frac{\sigma^2\lambda^2}{2}}. \tag{3.4}$$

Hence,

$$\psi_{(X-\mu)}^*(t) = \sup_{\lambda \geq 0} \left(\lambda t - \frac{\sigma^2\lambda^2}{2} \right) = \frac{t^2}{2\sigma^2}. \tag{3.5}$$

It follows that

$$\mathbb{P}(X \geq \mu + t) \leq \mathrm{e}^{-\frac{t^2}{2\sigma^2}}.$$

In the case of a normal distribution, we can get the more precise bounds

$$\left(\frac{1}{t} - \frac{1}{t^3} \right) \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-\frac{t^2}{2}} \leq \mathbb{P}((X - \mu)/\sigma \geq t) \leq \frac{1}{t} \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-\frac{t^2}{2}}$$

by a direct calculation, which shows that the inequality derived from the Chernoff bound is of the right order of magnitude.

Note that in (3.4), a mere inequality would have sufficed to derive the subsequent bound. Random variables for which the moment generating function satisfies such an inequality are called subgaussian.

Definition 3.2. A centred random variable X is called **subgaussian** with variance factor σ^2 if for every $\lambda \in \mathbb{R}$,

$$\psi_X(\lambda) \leq \frac{\sigma^2\lambda^2}{2}.$$

The following simple observation states that subgaussian random variables satisfy the same concentration bounds as the one derived in Example 3.1.

Lemma 3.3. *If $X - \mathbb{E}[X]$ is subgaussian, then for $t \geq 0$ we have*

$$\mathbb{P}(X \geq \mathbb{E}[X] + t) \leq \mathrm{e}^{-\frac{t^2}{2\sigma^2}}, \qquad \mathbb{P}(X \leq \mathbb{E}[X] - t) \leq \mathrm{e}^{-\frac{t^2}{2\sigma^2}}.$$

Proof. The proof of the first inequality consists of simply replacing the first equality in (3.5) with an inequality. For the second inequality, take the random variable $\mathbb{E}[X] - X$, which is clearly also subgaussian. □

Using the union bound, we can combine the two bounds in Lemma 3.3 into a single bound

$$\mathbb{P}(|X - \mathbb{E}[X]| \geq t) \leq 2\mathrm{e}^{-\frac{t^2}{2\sigma^2}},$$

which measures the deviation from the mean in either direction.

Hoeffding's Inequality

Hoeffding's inequality bounds the sum of independent random variables with a bounded domain.

Theorem 3.4 (Hoeffding's Bound)**.** *Let $X_1, \ldots, X_n$ be independent random variables, with X_i taking values in $[a_i, b_i]$, and set $c_i = b_i - a_i$, $\boldsymbol{c} = (c_1, \ldots, c_n)^\intercal$. Let $S_n = \frac{1}{n}\sum_{i=1}^n X_i$ and $\mu = \mathbb{E}[S_n]$. Then for any $t \geq 0$,*

$$\mathbb{P}\left(S_n \geq \mu + t\right) \leq \exp\left(-\frac{2n^2t^2}{\|\boldsymbol{c}\|^2}\right), \qquad \mathbb{P}\left(S_n \leq \mu - t\right) \leq \exp\left(-\frac{2n^2t^2}{\|\boldsymbol{c}\|^2}\right).$$

Note the implication of this: if we have a sequence of random variables $\{X_i\}$ bounded in $[0, 1]$, such as from a Bernoulli distribution, then $\|\boldsymbol{c}\|^2 = n$ and

$$\mathbb{P}(|S_n - \mu| \geq t) \leq 2\mathrm{e}^{-2nt^2}.$$

As n increases, the probability that the *average* of the random variables deviates from its mean decreases exponentially with n, which is exactly the type of behaviour that we are looking for. In particular, if the random variables all have the same expectation μ, then (by linearity of expectation) we have $\mathbb{E}[S_n] = \mu$, and the probability of straying from this value becomes very small very quickly!

The proof of Theorem 3.4 relies on bounding the cumulant generating function of a bounded random variable, a result known as Hoeffding's Lemma.

Lemma 3.5 (Hoeffding's Lemma)**.** *Let X be a random variable with $\mathbb{E}[X] = 0$ and taking values in $[a, b]$. Then for any $\lambda \in \mathbb{R}$,*

$$\psi_X(\lambda) \leq \frac{(b-a)^2\lambda^2}{8}.$$

That is, X is subgaussian with variance factor $(b-a)^2/4$.

The proof makes use of a simple variance bound for a bounded random variable. The proof is left as an exercise.

Lemma 3.6 (Popoviciu's Inequality)**.** *Let X be a random variable taking values in $[a, b]$. Then*

$$\mathrm{Var}(X) \leq \frac{(b-a)^2}{4}.$$

Proof of Hoeffding's Lemma 3.5. Assume $\lambda \neq 0$ (otherwise the bound is trivial). The bound is derived by considering the Taylor expansion of $\psi_X(\lambda)$ around $\lambda = 0$ up to second order:

$$\psi_X(\lambda) = \underbrace{\psi_X(0)}_{=0} + \lambda \underbrace{\psi_X'(0)}_{=0} + \frac{\lambda^2}{2}\psi_X''(\theta) = \frac{\lambda^2}{2}\psi_X''(\theta)$$

for some $\theta \in [0, \lambda]$ (or $[\lambda, 0]$ if $\lambda < 0$). We are therefore led to investigate the second derivative:

$$\psi_X''(\theta) = \frac{\mathbb{E}[Y^2 \mathrm{e}^{\theta X}]}{\mathbb{E}[\mathrm{e}^{\theta X}]} - \left(\frac{\mathbb{E}[Y \mathrm{e}^{\theta X}]}{\mathbb{E}[\mathrm{e}^{\theta X}]} \right)^2. \tag{3.6}$$

If X has density $\rho(x)$, consider the tilted random variable X_θ with density

$$\frac{\mathrm{e}^{\theta x} \rho(x)}{\mathbb{E}[\mathrm{e}^{\theta X}]}.$$

If X is discrete, then take $\rho(x)$ to be the probability of x. With this, we see that (3.6) is simply the variance $\mathrm{Var}(X_\theta)$. Since $X_\theta \in [a, b]$ (note that only the density has changed, not the support), the claim follows from Lemma 3.6. □

Proof of Hoeffding's Bound 3.4. Consider without loss of generality the normalized random variables $Y_i = X_i - \mathbb{E}[X_i]$ and set $\overline{S}_n = (1/n) \sum_{i=1}^n Y_i = S_n - \mu$. Then the cumulant generating function satisfies

$$\psi_{\overline{S}_n}(\lambda) = \sum_{i=1}^n \psi_{Y_i}(\lambda/n), \tag{3.7}$$

because Ψ_X is linear in X for independent X. Since each of the X_i lies in $[a_i, b_i]$, the Y_i live in intervals of the same length, and applying Hoeffding's Lemma 3.5 to (3.7) we get

$$\psi_{\overline{S}_n}(\lambda) \leq \frac{\lambda^2}{8n^2} \sum_{i=1}^n (b_i - a_i)^2 = \frac{\lambda^2 \|\boldsymbol{c}\|^2}{8n^2}.$$

Using Lemma 3.3, we conclude that

$$\mathbb{P}(S_n \geq \mu + t) \leq \exp\left(-\frac{2n^2 t^2}{\|\boldsymbol{c}\|^2} \right) \text{ and } \mathbb{P}(S_n \leq \mu - t) \leq \exp\left(-\frac{2n^2 t^2}{\|\boldsymbol{c}\|^2} \right).$$

This completes the proof. □

McDiarmid's Inequality

McDiarmid's inequality is a generalization of Hoeffding's bound, where instead of considering sums of independent random variables, one considers functions with bounded differences.

Theorem 3.7 (McDiarmid's Inequality)**.** *Let $X_1, \ldots, X_n$ be independent random variables and let $c_1, \ldots, c_n$ be positive constants. Set $\mathbf{c} = (c_1, \ldots, c_n)^\intercal$. Let f be a real valued function such that for each $i \in \{1, \ldots, n\}$ and values $x_i \neq x_i'$ we have*

$$|f(x_1, \ldots, x_i, \ldots, x_n) - f(x_1, \ldots, x_i', \ldots, x_n)| \leq c_i.$$

Set $\mu = \mathbb{E}[f(X_1, \ldots, X_n)]$. Then for any $t \geq 0$,

$$\mathbb{P}\left(f(X_1, \ldots, X_n) \geq \mu + t \right) \leq \exp\left(-\frac{2t^2}{\|\boldsymbol{c}\|^2} \right),$$
$$\mathbb{P}\left(f(X_1, \ldots, X_n) \leq \mu - t \right) \leq \exp\left(-\frac{2t^2}{\|\boldsymbol{c}\|^2} \right).$$

For the proof we recall the notion of conditional expectation. Given random variables (X, Y) defined on $\mathcal{X} \times \mathcal{Y}$, the conditional expectation $\mathbb{E}[X \mid Y]$ is a random variable $Z \colon \mathcal{Y} \to \mathcal{X}$ with $Z(y) = \mathbb{E}[X \mid Y](y) = \mathbb{E}[X \mid Y = y]$. Moreover, this random variable satisfies $\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X|Y]]$

Proof. To ease notation, set $X = (X_1, \ldots, X_n)$ and $X^{(i)} = (X_1, \ldots, X_i)$. For each $i \in \{2, \ldots, n\}$, consider the random variables

$$Z_i := \mathbb{E}[f(X) \mid X^{(i)}] - \mathbb{E}[f(X) \mid X^{(i-1)}].$$

Note that $\mathbb{E}[Z_i \mid X^{(i-1)}] = 0$ and that

$$S_n := \sum_{i=1}^{n} Z_i = f(X_1, \ldots, X_n) - \mathbb{E}[f(X_1, \ldots, X_n)],$$

so that we reduced the problem to bounding the deviation of a sum of random variables. If the Z_i were independent and bounded in some interval, we could apply Hoeffding's inequality, but this is not the case. The situation is salvaged, however, by the fact that they form a *martingale*, and that each Z_i is bounded *conditional on* the previous Z_j in the sequence. To make this precise, introduce the random variables

$$\begin{aligned} A_i &= \inf_x(\mathbb{E}[f(X) \mid X^{(i-1)}, X_i = x] - \mathbb{E}[f(X) \mid X^{(i-1)}], \\ B_i &= \sup_x(\mathbb{E}[f(X) \mid X^{(i-1)}, X_i = x] - \mathbb{E}[f(X) \mid X^{(i-1)}]. \end{aligned}$$

Clearly, $Z_i \in [A_i, B_i]$ for all i, but these are not deterministic bounds. We have, however, that

$$\begin{aligned} B_i - A_i &= \sup_{x,x'}(\mathbb{E}[f(X) \mid X^{i-1}, X_i = x] - \mathbb{E}[f(X) \mid X^{(i-1)}, X_i = x']) \\ &= \sup_{x,x'}(\mathbb{E}[f(X_1, \ldots, X_{i-1}, x, X_{i+1}, \ldots, X_n) \\ &\qquad - f(X_1, \ldots, X_{i-1}, x', X_{i+1}, \ldots, X_n) \mid X^{(i-1)}]) \leq c_i, \end{aligned}$$

where the last inequality follows from the bounded difference assumption. This means that for each assignment $\boldsymbol{x}^{(i-1)} = (x_1, \ldots, x_{i-1})$ we can find bounds a_i and b_i with $c_i = b_i - a_i$, such that the random variable

$$Z_i(\boldsymbol{x}^{i-1}, X_i) := \mathbb{E}[f(X) \mid X^{(i-1)} = \boldsymbol{x}^{(i-1)}, X_i] - \mathbb{E}[f(X) \mid X^{(i-1)} = \boldsymbol{x}^{(i-1)}]$$

is bounded in $[a_i, b_i]$. We can therefore bound the exponential moment of this random variable as

$$\mathbb{E}[\mathrm{e}^{\lambda Z_i}] = \mathbb{E}\left[\mathbb{E}[\mathrm{e}^{\lambda Z_i} \mid X^{(i-1)}]\right] \leq \mathrm{e}^{\frac{\lambda^2 c_i^2}{8}},$$

where the last inequality follows from interpreting the outer expectation as first integrating/summing over all values $X^{(i-1)} = \boldsymbol{x}^{(i-1)}$ and then over X_i, and applying Hoeffding's Lemma. For the exponential moment of the sum we have

$$\mathbb{E}[\mathrm{e}^{\lambda S_n}] = \mathbb{E}\left[\mathbb{E}[\mathrm{e}^{\lambda S_n} \mid X^{(n-1)}]\right] = \mathbb{E}\left[\mathbb{E}[\mathrm{e}^{\lambda Z_n}\mathrm{e}^{\lambda S_{n-1}} \mid X^{(n-1)}]\right].$$

where we use the notation $S_k = \sum_{i=1}^{k} Z_i$. Since S_{n-1} is completely determined by $X^{(n-1)}$, we have $\mathbb{E}[\mathrm{e}^{\lambda S_{n-1}} \mid X^{(n-1)}] = \mathrm{e}^{\lambda S_{n-1}}$, and conditioned on $X^{(n-1)}$ this is independent of Z_n. We therefore have

$$\mathbb{E}[\mathrm{e}^{\lambda S_n}] = \mathbb{E}\left[\mathbb{E}[\mathrm{e}^{\lambda Z_n} \mid X^{(n-1)}]\mathrm{e}^{\lambda S_{n-1}}\right] = \mathbb{E}[\mathrm{e}^{\lambda Z_n}]\,\mathbb{E}[\mathrm{e}^{\lambda S_{n-1}}] \leq \mathrm{e}^{\frac{\lambda^2 c_n^2}{8}}\,\mathbb{E}[\mathrm{e}^{\lambda S_{n-1}}].$$

The claim now follows by applying this reduction recursively to S_{n-1}. □

Notes

Excellent sources for the material covered in this chapter are [14] and [83, Chapter 2]. The law of large number for binary random variables was first derived rigorously by Swiss mathematician Jakob Bernoulli in 1713 [9, Chapter 4], a treatise that is widely considered as the beginning of the mathematical study of probability. Concentration inequalities for sums of independent random variables that are derived from the exponential moment inequality are commonly referred to as Chernoff bounds, named after Herman Chernoff [18]. Hoeffding's inequality was first derived by Hoeffding in [40], where he also hinted at a generalization to the setting of martingales. This generalization was carried out by Azuma [1], giving rise to the Azuma-Hoeffding inequality. The excellent survey [55] by Colin McDiarmid contains a discussion of martingale methods and, in particular, his eponymous inequality. For a more general perspective on measure concentration, with connections to geometry, isoperimetry, and functional analysis, a good reference is [49].

4

The Bias-Variance Tradeoff

Given a fixed hypothesis set $\mathcal{H}$ consisting of binary classifiers $h\colon \mathcal{X} \to \mathcal{Y} = \{0, 1\}$, we would like to find a classifier $\hat{h}_n \in \mathcal{H}$, computed from n i.i.d. random samples (X_i, Y_i), that has a small **excess risk**

$$\mathcal{E}(\hat{h}_n) = R(\hat{h}_n) - R(h^*), \tag{A}$$

where h^* denotes the Bayes classifiers and $R(h) = \mathbb{P}(h(X) \neq Y)$. The excess risk may be large because the best achievable risk among classifiers in $\mathcal{H}$ is far from $R(h^*)$. But even if $\mathcal{H}$ is expressive enough to contain h^*, finding the best possible classifier in $\mathcal{H}$ may not be feasible. We will mostly be concerned with the case when $\hat{h}_n$ is constructed by minimizing the empirical risk over the class $\mathcal{H}$:

$$\hat{R}(\hat{h}_n) = \inf_{h \in \mathcal{H}} \hat{R}(h), \quad \text{where} \quad \hat{R}(h) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{h(X_i) \neq Y_i\}.$$

As $\hat{h}_n$ is constructed by minimizing the average over a finite sample set, rather than the expected value over the whole space, the resulting $\hat{h}_n$ will usually not minimize $R(h)$ over $\mathcal{H}$. Less ambitious than trying to minimize the excess risk is therefore the goal of bounding the difference in risk between $\hat{h}_n$ and the best possible $h \in \mathcal{H}$. In this chapter we study the relationship between the error due to the limitations of the class $\mathcal{H}$ to the error due to the limitations of working with finite data samples.

Approximation and Estimation

Given a set of candidate classifiers $\mathcal{H}$, we can decompose the generalization risk $R(\hat{h}_n)$ as follows:

$$R(\hat{h}_n) = \underbrace{R(\hat{h}_n) - \inf_{h \in \mathcal{H}} R(h)}_{\text{Estimation error}} + \underbrace{\inf_{h \in \mathcal{H}} R(h) - R(h^*)}_{\text{Approximation error}} + \underbrace{R(h^*)}_{\text{Irreducible error}}$$

The first part compares the performance of $\hat{h}_n$ against the best possible classifier *within the class* $\mathcal{H}$, while the second part is a statement about the power of the class $\mathcal{H}$ itself: it tells us how close we can get to the Bayes classifier if we stay within $\mathcal{H}$. We can reduce the estimation error by making the class $\mathcal{H}$ smaller, but then the approximation error increases. Note that here we consider $\hat{h}_n$, and hence $R(\hat{h}_n)$, as a random variable, constructed from n i.i.d. random samples $\{(X_i, Y_i)\}_{i=1}^n$.

Example 4.1. There is usually no unique minimizer of $\hat{R}(h)$ within a class $\mathcal{H}$. Figure 4.1 shows data generated by taking $X = (X_1, X_2)$ to be uniformly distributed on a square in $\mathbb{R}^2$, and choosing Y from a Bernoulli distribution with probability p that is $1/2$ the line $x_2 = 4 - x_1$, increases to 1 with the distance

to this line if $X_2 > 4 - X_1$, and decreases to 0 with the distance if $X_2 < 4 - X_1$. Four classifiers $\hat{h}_n$ are chosen from standard sets of classifiers in machine learning (we will study Support Vector Machines (SVM) in more detail later). On the training data, the accuracy (percentage of correctly classified points) of the four methods is 92.5%, 95%, 97.5% and 100%, respectively. On the whole distribution, though, the linear SVM classifier performs best, with an accuracy of 87.7%, while the decision tree classifier, that fitted the training data perfectly, only achieves 82%.

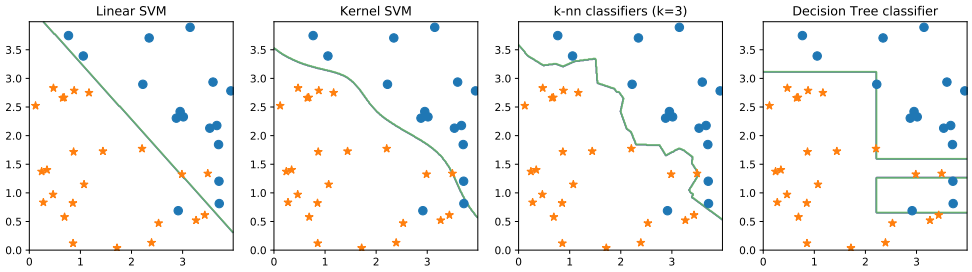


Figure 4.1: For decision boundaries for some standard classifiers.

In this particular example, the Bayes classifier itself is a linear classifier (we assign values $y = 1$ or $y = 0$ to an input x depending on whether that point lies on one side of a straight line or another), and hence the class $\mathcal{H}$ of linear classifiers has approximation error 0. This is not necessarily true for other, possibly more extensive classes of functions that can have a smaller estimation error. A classifier is a deterministic rule: given an input $\boldsymbol{x}$, it assigns to it a label y. Adapting a classifier too closely to observed training data is called **overfitting**, and is generally considered to be the worst possible crime in data science. Informally, one can think of overfitting as trying to model the noise, and not just the relationships in the data.

Example 4.2. We now consider again points in $\mathbb{R}^2$ that are split in two groups, and consider the class $\mathcal{H}_n$ of k-nearest neighbour classifiers. Given a reference data set $\{(\boldsymbol{x}^i, y^i)\}_i = 1^n$, a k-th nearest neighbour classifier $\hat{h}_n$ is constructed by first computing, for each $\boldsymbol{x}$, the average

$$\frac{1}{k} \sum_{i \in N_k(\boldsymbol{x})} y^i,$$

where $N_k(\boldsymbol{x})$ consists of those indices $j \in \{1, \dots, n\}$ that correspond to the closest points to $\boldsymbol{x}$ among the $\boldsymbol{x}^j$. We then assign the label $y = 1$ if this average is greater than $1/2$ and 0 if it is smaller than $1/2$ (hence, the label is assigned based on the label of the majority of the k closest neighbours). The distance between points can be measured in various ways; here, we use the usual Euclidean distance on the plane. Figure 4.2 show the decision boundaries for k-nearest classifiers for various k.

How do we evaluate which of these performs best, in the sense of having the smallest generalization risk? Since we do not (or pretend to not) have access to the distribution that generated the data, the only sensible option is to sample more data and evaluate compute the empirical error on this additional data. Such additional data is called **test data**. In machine learning, one typically splits the available data randomly into training and test data, uses the training data to construct the classifier, and the test data to evaluate the classifier by computing an approximation to the generalization risk. Figure 4.3 compares the classification error on the training set with the generalization risk (computed by generating a very

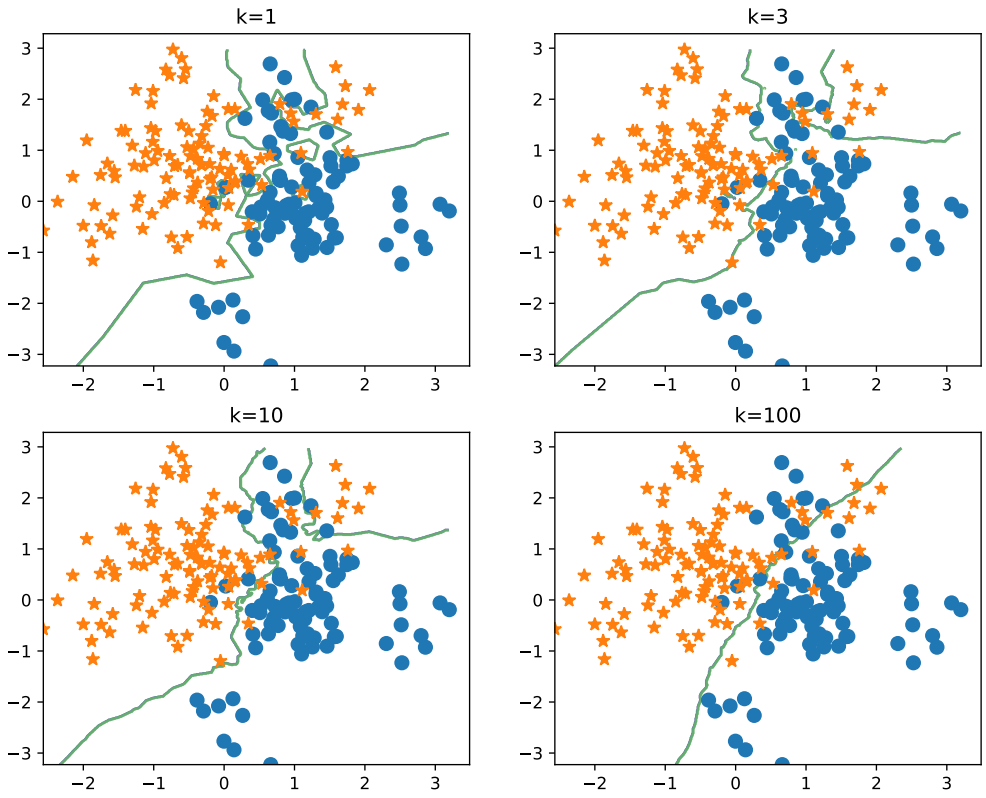


Figure 4.2: The decision boundary of four k-nearest neighbour classifiers.

large number of additional samples) and the Bayes risk (which we can compute here, since we know the distribution from which the data was generated). In this example, the data consists of a mixture of two mixtures of 10 Gaussians each.

We see that the 1-nearest neighbour classifier does best on the training data (100% accuracy) but becomes less accurate on the training data as k increases. On the other hand, the generalization risk appears to be optimal at around a value of $k = 13$. The parameter k is related to the **variance** of a classifier. For small k the variance is high, in the sense that the resulting classifier is closely adapted ("overfitted") to a particular set of samples, while for large k the variance is small, in the sense that the decision boundary will not change much with changes in the training data (see, for example, the difference between the $k = 1$ and the $k = 100$ displays in Figure 4.2).

The Bias-Variance Tradeoff

The dichotomy between estimation and approximation is closely related to the concept of **bias-variance** tradeoff in statistics. To see this connection, note that we can express the generalization risk as means squared error:

$$R(\hat{h}_n) = \mathbb{P}(\hat{h}_n(X) \neq Y) = \mathbb{E}[(\hat{h}_n(X) - Y)^2].$$

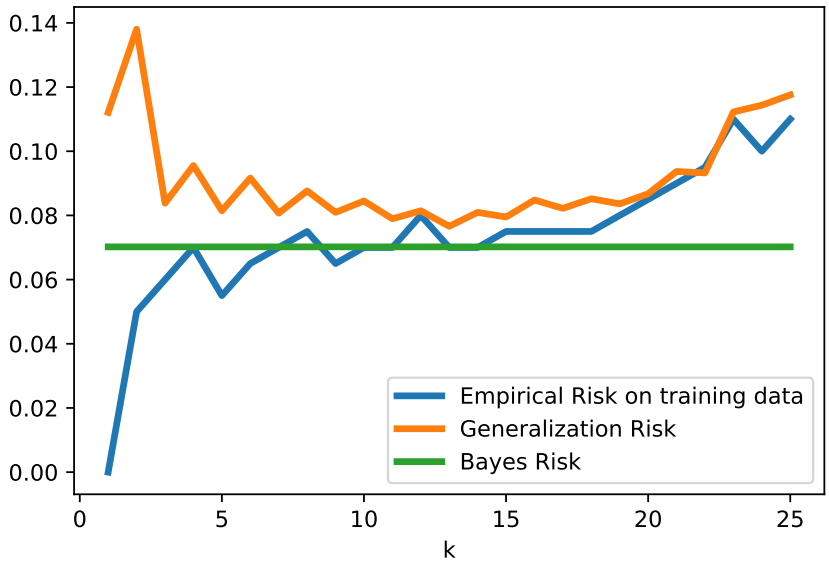


Figure 4.3: The empirical risk on training data compared to the generalization risk and the Bayes risk for k-nearest neighbour classifiers and different values of k. The different between the upper curve and the Bayes error at each k is the approximation error for the set of k-nearest neighbour classifiers.

Recall that the Bayes classifier was defined in terms of the regression function $f(X) = \mathbb{E}[Y|X]$, and that the response variable Y could be characterized as

$$Y = f(X) + \epsilon,$$

where ϵ can be interpreted as random noise with $\mathbb{E}[\epsilon|X] = 0$, while f describes the relationship between input and output data that we would like to capture. The variance

$$\sigma^2 = \mathbb{E}[\epsilon^2] = \mathbb{E}[(Y - f(X))^2]$$

is called the **irreducible error**. A positive irreducible error prevents the Bayes risk from being zero. The well-known bias-variance decomposition in statistics, applied to our context, can be formulated as

$$\begin{aligned}\mathbb{E}[(Y - \hat{h}_n(X))^2] &= \mathbb{E}[(Y - \mathbb{E}[\hat{h}_n])^2] + \mathbb{E}[(\hat{h}_n(X) - \mathbb{E}[\hat{h}_n])^2] \\ &= \mathbb{E}[\underbrace{(f(X) - \mathbb{E}[\hat{h}_n])}_{\text{bias}}{}^2] + \underbrace{\mathbb{E}[(\hat{h}_n(X) - \mathbb{E}[\hat{h}_n])^2]}_{\text{variance}} + \underbrace{\sigma^2}_{\text{irreducible error}}.\end{aligned}$$

The expectation here is taken over both (X, Y) and random training data $\{(X_i, Y_i)\}_{i=1}^n$.

Bounding the estimation error

In this chapter we focus on the estimation error and leave the problem of choosing an adequate class $\mathcal{H}$ for later. Assume there exists a best possible classifier $\overline{h} \in \mathcal{H}$,

$$\overline{h} \in \arg\min_{h \in \mathcal{H}} R(h).$$

The notation emphasizes that such an optimal classifier does not have to be unique. The classifier $\overline{h}$ depends only on the class $\mathcal{H}$ and the probability distribution. Since $R(\hat{h}_n)$ is a random variable (as mentioned above, it depends on the n i.i.d. samples (X_i, Y_i)), any bounds on the difference $R(\hat{h}_n) - R(\overline{h})$ are necessarily probabilistic. More precisely, for any given *tolerance* $\delta \in (0, 1)$, we want to find a bound $C(n, \delta)$ such that

$$R(\hat{h}_n) - R(\overline{h}) \leq C(n, \delta)$$

holds with probability $1 - \delta$. Ideally, the constants should also depend on properties of the set $\mathcal{H}$, for example the size of $\mathcal{H}$ if this set is finite. In addition, we would like the bound to decrease to 0 as $n \to \infty$.

If we denote by $\overline{h}$ the minimizer of $R(h)$ over $\mathcal{H}$, then we can decompose the estimation error as

$$\begin{aligned} R(\hat{h}_n) - R(\overline{h}) &= \overbrace{\hat{R}(\hat{h}_n) - \hat{R}(\overline{h})}^{\leq 0} + R(\hat{h}_n) - \hat{R}(\hat{h}_n) + \hat{R}(\overline{h}) - R(\overline{h}) \\ &\leq 2 \sup_{h \in \mathcal{H}} |R(h) - \hat{R}(h)|. \end{aligned} \tag{4.1}$$

The inequality $\hat{R}(\hat{h}_n) - \hat{R}(\overline{h}) \leq 0$ holds by definition, since $\hat{h}_n$ is the minimizer of the empirical risk. We thus arrive at the following problem: given $\delta \in (0, 1)$, find a bound $C(n, \delta)$ such that

$$\mathbb{P}\left(\sup_{h \in \mathcal{H}} |R(h) - \hat{R}(h)| > C(n, \delta) \right) \leq \delta.$$

This problem brings us into the territory of the study of suprema of stochastic processes, and concentration of measure inequalities will play an important role in deriving such bounds.

Notes

See [35, Chapter 2] for a more detailed discussion of the bias-variance tradeoff in the case of linear regression and k-nearest neighbour classification. In particular, our Example 4.2 is taken from this source. The decomposition (4.1) was observed by Vapnik and Chervonenkis [79], see also [25, Chapter 8] for a comprehensive treatment.

5

Finite Hypothesis Sets

In this chapter we assume that the set of classifiers $\mathcal{H} = \{h_1, \ldots, h_K\}$ is finite. Let $\hat{h}_n$ be a minimizer of the empirical risk in $\hat{h}_n$. We are interested in how well this classifier performs in relation to the best possible risk $R(h)$ for $h \in \mathcal{H}$:

$$R(\hat{h}_n) - \min_{h \in \mathcal{H}} R(h).$$

We will derive a probabilistic bound on this difference that depend logarithmically on the size $K = |\mathcal{H}|$ and decays with order $n^{-1/2}$. This bound makes use of Hoeffding's inequality, and serves as a prototype for a whole range of similar risk bounds in statistical learning theory.

Finite Hypothesis Sets

The following theorem gives us an effective bound on the estimation error.

Theorem 5.1. *Let $\mathcal{H} = \{h_1, \ldots, h_K\}$ be a finite dictionary. Then for $\delta \in (0, 1)$,*

$$\mathbb{P}\left(R(\hat{h}_n) - \min_{h \in \mathcal{H}} R(h) \leq \sqrt{\frac{2 \log\left(\frac{2K}{\delta}\right)}{n}} \right) \geq 1 - \delta.$$

Unless otherwise stated, the logarithm refers to the natural logarithm (though this is not important). This important result shows that (with high probability) we can bound the estimation error by a term that is *logarithmic* in the size of $\mathcal{H}$, and proportional to $n^{-1/2}$, where n is the number of samples. For fixed or moderately growing K, this error goes to zero as n goes to infinity.

Recall from Chapter 4 the inequality

$$R(\hat{h}_n) - \min_{h \in \mathcal{H}} R(h) \leq 2 \max_{h \in \mathcal{H}} |\hat{R}(h) - R(h)|, \tag{5.1}$$

where $\hat{R}$ is the empirical risk. As a first step towards bounding the supremum on the right-hand side, we need to bound the difference $|R(h) - \hat{R}(h)|$ of an individual, fixed h. The key ingredient for such a bound Hoeffding's inequality, which we recall here.

Theorem 5.2 (Hoeffding's Inequality)**.** *Let $Z_1, \ldots, Z_n$ be independent random variables taking values in $[0, 1]$, and let $\overline{Z}_n = (1/n) \sum_{i=1}^n Z_i$ be the average. Then for $t \geq 0$,*

$$\mathbb{P}\left(|\overline{Z}_n - \mathbb{E}[\overline{Z}_n]| > t \right) \leq 2\mathrm{e}^{-2nt^2}.$$

Using Hoeffding's Inequality we obtain the following bound on the difference between the empirical risk and the risk of a classifier.

Lemma 5.3. *For any classifier h and $\delta \in (0,1)$,*

$$|\hat{R}(h) - R(h)| \leq \sqrt{\frac{\log(2/\delta)}{2n}}$$

holds with probability at least $1-\delta$.

Proof. Set $Z_i = \mathbf{1}\{h(X_i) \neq Y_i\}$. Then

$$\overline{Z}_n = \frac{1}{n}\sum_{i=1}^{n} Z_i = \hat{R}(h),$$
$$\mathbb{E}[\overline{Z}_n] = \mathbb{E}[\hat{R}(h)] = R(h),$$

and the Z_i satisfy the conditions of Hoeffding's inequality. Set $\delta = 2\mathrm{e}^{-2nt^2}$ and resolve for t, which gives

$$t = \sqrt{\frac{\log(2/\delta)}{2n}}.$$

Hence, by Hoeffding's inequality,

$$\mathbb{P}(|\hat{R}(h) - R(h)| > t) = \mathbb{P}(|\overline{Z}_n - \mathbb{E}[\overline{Z}_n]| > t) \leq \delta$$

and therefore, by taking the complement,

$$\mathbb{P}(|\hat{R}(h) - R(h)| \leq t) = 1 - \mathbb{P}(|\hat{R}(h) - R(h)| > t) \geq 1 - \delta,$$

which was claimed. □

Proof of Theorem 5.1. Courtesy of (5.1), the goal is to bound the supremum

$$\max_{h\in\mathcal{H}} |\hat{R}(h) - R(h)|. \tag{5.2}$$

For this, we use the union bound. Indeed, for each h_i we can apply Lemma 5.3 with δ/K to show that

$$\mathbb{P}\left(|\hat{R}(h_i) - R(h_i)| > \frac{t}{2}\right) \leq \frac{\delta}{K},$$

where $t = \sqrt{\frac{2\log(2K/\delta)}{n}}$. The probability of (5.2) being bounded by t can be expressed equivalently as

$$\begin{aligned}&\mathbb{P}\left(\max_{h\in\mathcal{H}} |\hat{R}(h) - R(h)| \leq t/2\right)\\ &\quad= \mathbb{P}(|\hat{R}(h_1) - R(h_1)| \leq t/2, \ldots, |\hat{R}(h_K) - R(h_K)| \leq t/2).\end{aligned}$$

Since the right-hand side is an *intersection* of events, the *complement* of this event is the *union* of the events $|\hat{R}(h_i) - R(h_i)| > t$, and we can apply the union bound:

$$\begin{aligned}\mathbb{P}\left(\max_{h\in\mathcal{H}} |\hat{R}(h) - R(h)| > t/2\right) &\leq \sum_{i=1}^{K} \mathbb{P}(|\hat{R}(h_i) - R(h_i)| > t/2)\\ &\leq K \cdot \frac{\delta}{K} = \delta.\end{aligned}$$

Therefore, with probability at least $1-\delta$ we have

$$2\max_{h\in\mathcal{H}} |\hat{R}(h) - R(h)| \leq \sqrt{\frac{2\log(2K/\delta)}{n}},$$

and using (5.1) the claim follows. □

Generalization bounds and noise

One drawback of the bound in Theorem 5.1 is that it does not take into account properties of the underlying distribution. In particular, it is the same in the case where Y is completely determined by X as it is in the case in which Y is completely independent on X. Intuitively, in the first situation we would hope to get better rates of convergence than in the second. Using more refined concentration inequalities such as the Bernstein inequality, that take into account the variance of the random variables, we can get better rates of convergence in some situation.

Recall the definition of the regression function $f(X) = \mathbb{E}[Y|X]$ and the associated error $\epsilon = Y - f(X)$. Let $\gamma \in (0, 1/2]$ and assume that

$$|f(X) - 1/2| \geq \gamma$$

almost surely. This condition is known as **Massart's noise condition**. If $\gamma = 1/2$, then $f(X)$ is either 1 or 0 and we are in the deterministic case, where Y is completely determined by X. If, on the other hand, $\gamma \approx 0$, then we are barely placing any restrictions on $f(X)$, and we are allowing for the case where $f(X)$ is close to 0, and hence where Y is almost independent of X.

Theorem 5.4. *Let $\mathcal{H} = \{h_1, \ldots, h_K\}$ be a finite dictionary and assume that $h^* \in \mathcal{H}$, where h^* is the Bayes classifier. Then for $\delta \in (0, 1)$,*

$$\mathbb{P}\left(R(\hat{h}_n) - R(h^*) \leq \frac{\log\left(\frac{K}{\delta}\right)}{\gamma n} \right) \geq 1 - \delta.$$

Note that the bound decays with order n^{-1}, rather than $n^{-1/2}$ as with the bound derived from Hoeffding's inequality. The proof of this result relies on a concentration of measure result similar to Hoeffding's inequality, called Bernstein's inequality.

Theorem 5.5 (Bernstein Inequality)**.** *Let $\{Z_i\}_{i=1}^n$ be centred random variables (that is, $\mathbb{E}[Z_i] = 0$) with $|Z_i| \leq c$ and set $\sigma^2 = \frac{1}{n}\sum_{i=1}^n \mathrm{Var}(Z_i)$. Then for $t > 0$,*

$$\mathbb{P}\left(\frac{1}{n}\sum_{i=1}^n Z_i > t \right) \leq \exp\left(-\frac{nt^2}{2(\sigma^2 + ct/3)} \right).$$

We outline the idea of the proof of Theorem 5.4. The proof proceeds by defining the random variables

$$Z_i(h) = \mathbf{1}\{h^*(X_i) \neq Y_i\} - \mathbf{1}\{h(X_i) \neq Y_i\}$$

for each $h \in \mathcal{H}$. The average and expectation of these random variables is then

$$\begin{aligned} \hat{R}(h^*) - \hat{R}(h) &= \frac{1}{n}\sum_{i=1}^n Z_i(h), \\ R(h^*) - R(h) &= \mathbb{E}[Z_i(h)]. \end{aligned}$$

Based on this, one gets a bound

$$\begin{aligned} R(\hat{h}_n) - R(h^*) &\leq \frac{1}{n}\sum_{i=1}^n \underbrace{(Z_i(\hat{h}_n) - \mathbb{E}[Z_i(\hat{h}_n)])}_{\overline{Z}_i(\hat{h}_n)} \\ &\leq \max_{h \in \mathcal{H}} \frac{1}{n}\sum_{i=1}^n \overline{Z}_i(h). \end{aligned} \tag{5.3}$$

The random variables $\overline{Z}_i(h)$ are centred, satisfy $|Z_i(h)| \leq 2 =: c$ and we can bound the variance by

$$\mathrm{Var}(\overline{Z}_i(h)) = \mathrm{Var}(Z_i(h)) \leq \mathbb{P}(h(X_i) \neq h^*(X_i)) =: \sigma^2(h).$$

We can now apply Bernstein's inequality to the probability that the sum (5.3) exceeds a certain value for each individual h, and use a union bound to get a corresponding bound for the maximum that involves the variance $\sigma^2(\hat{h}_n)$. Using the property that $h^* \in \mathcal{H}$, one can also derive a lower bound on the excess risk in terms of the variance, and hence combine both bounds to get the desired result.

Notes

The application of Hoeffding's inequality is a standard tool in statistical learning, and references are [25, Section 8.2] and [72, Chapter 4]. The bounds on derived in this chapter are bounds on the difference between the empirical risk and the generalization risk over the whole class $\mathcal{H}$, a setting usually also referred to as **uniform convergence**.

6

Probably Approximately Correct

As before, we consider a fixed *dictionary* $\mathcal{H}$ and select one classifier $\hat{h}_n$ that optimizes the empirical risk $\hat{R}(h)$. Recall:

- The *empirical risk* $\hat{R}$ and the classifier $\hat{h}_n$ depend on the *data* (X_i, Y_i), $1 \leq i \leq n$, and are random variables. In particular, they depend on n;
- The *risk* $R(h) = \mathbb{E}[\hat{R}(h)]$ depends on the underlying distribution on $\mathcal{X} \times \mathcal{Y}$, but not on n.

We have seen that if $\mathcal{H} = \{h_1, \ldots, h_K\}$ is finite, then with probability $1 - \delta$, we have

$$R(\hat{h}_n) - \inf_{h \in \mathcal{H}} R(h) \leq \sqrt{\frac{2\log(K) + 2\log(2/\delta)}{n}}. \tag{6.1}$$

Note that $\log(K)$ is proportional to the *bit size* of K: this is the amount of bits needed to represent numbers up to K, and can be seen as a measure of complexity for the set $\mathcal{H}$ (the "space" necessary to represent K elements). Bounds such as (6.1) are called **generalization bounds**.

Probably Approximately Correct Learning

An alternative point of view to generalization bounds would be to ask, for given **accuracy** $\epsilon > 0$ and **confidence** $\delta \in (0, 1)$, how many samples n are needed to get an accuracy of ϵ with confidence δ:

$$\mathbb{P}(R(\hat{h}_n) - \inf_{h \in \mathcal{H}} R(h) \leq \epsilon) \geq 1 - \delta.$$

Assuming $h^* \in \mathcal{H}$ and $Y = f(X)$, we have $R(h^*) = 0$, and h^* would be the *correct* classifier. The classifier $\hat{h}_n$ is then **probably** (with probability $1 - \delta$) **approximately** (up to an misclassification probability of at most ϵ) **correct**. This leads us to the notion of **Probably Approximately Correct** (**PAC**) learning. In what follows, we denote by $\mathrm{size}(\mathcal{H})$ the complexity of representing an element of $\mathcal{H}$. This is not a precise definition, but depends on the case at hand. For example, if $\mathcal{H} = \{h_1, \ldots, h_K\}$ is a finite set, then we can index this set using K numbers. On a computer, numbers up to K can be represented as binary numbers using $\lceil \log_2(K) \rceil$ bits, and hence (up to a constant factor) $\mathrm{size}(\mathcal{H}) = \log(K)$ would be adequate here. Similarly, we denote by $\mathrm{size}(\mathcal{X})$ the complexity of representing an element of the input space. For example, if $\mathcal{X} \subset \mathbb{R}^d$, then we would use d as size parameter (possibly multiplied by a constant factor to account for the size of representing a real number in floating point arithmetic on a computer). Note that $\mathrm{size}(\mathcal{X})$ or $\mathrm{size}(\mathcal{H})$ is not the same as the cardinality of these sets!

Definition 6.1. (PAC Learning [1]) A hypothesis class $\mathcal{H}$ is called **PAC-learnable** if there exists a classifier $\hat{h}_n \in \mathcal{H}$ depending on n random samples (X_i, Y_i), $i \in \{1, \ldots, n\}$, and a polynomial function $p(x, y, z, w)$, such that for any $\epsilon > 0$ and $\delta \in (0, 1)$, for *all* distributions on $\mathcal{X} \times \mathcal{Y}$,

$$\mathbb{P}\left(R(\hat{h}_n) \leq \inf_{h \in \mathcal{H}} R(h) + \epsilon\right) \geq 1 - \delta$$

holds whenever $n \geq p(1/\epsilon, 1/\delta, \text{size}(\mathcal{X}), \text{size}(\mathcal{H}))$. We also say that $\mathcal{H}$ is **efficiently** PAC-learnable, if the algorithm that produces $\hat{h}_n$ from the data runs in time polynomial in $1/\epsilon$, $1/\delta$, $\text{size}(\mathcal{X})$ and $\text{size}(\mathcal{H})$.

Remark 6.2. In our context, to say that an algorithm "runs in time $p(n)$" means that the number of steps, with respect to some suitable model of computation, is bounded by $p(n)$. Note that in this definition we disregard specific constants in the lower bound on n, but only require that it is polynomial. In computer science, polynomial time or space is considered *efficient*, while problems that require exponential time and/or space to solve are considered inefficient. For example, sorting n numbers can be performed in $O(n \log(n))$ operations and is efficient, while it is not known if finding the shortest route through n cities (the Traveling Salesman Problem) can be solved in a number of computational steps that is polynomial in n. This is the subject of the famous P vs NP conjecture.

In the case of a finite hypothesis space $\mathcal{H}$ with K elements, we have seen that $\mathcal{H}$ is PAC-learnable, since

$$n \geq \left(\frac{2}{\epsilon^2}\right)\left(\log(K) + \log\left(\frac{2}{\delta}\right)\right),$$

which is polynomial in all the relevant parameters. Under Massart's noise condition (see Chapter 5), we see that

$$n \geq \frac{1}{\gamma\epsilon}\left(\log(K) + \log\left(\frac{1}{\delta}\right)\right),$$

samples are necessary to approximate the Bayes classifier up to a factor of ϵ with confidence $1 - \delta$. We also see here that the number of samples needed increases as $\gamma \to 0$, reflecting the fact that in the presence of high uncertainty, more observations are needed than if we have low uncertainty.

Learning Rectangles

So far we have considered learning with finite dictionaries of classifiers $\mathcal{H}$. We now illustrate an example where $\mathcal{H}$ is not finite, and show how PAC-learnability and generalization bounds can be derived in this setting.

Assume that our data describes a rectangle: the input space $\mathcal{X}$ is a subset of $\mathbb{R}^2$, and the function $f\colon \mathcal{X} \to \{0, 1\}$ is the indicator function of a closed rectangle B, so that for any point x, $f(x) = 1$ if x is in the rectangle, and 0 if not. Suppose that all we have access to is a random sample of labelled points, (x_i, y_i), $i \in \{1, \ldots, n\}$ (see Figure 6.1, left panel).

We compute a candidate $\hat{h}\colon \mathbb{R}^2 \to \{0, 1\}$ as the indicator function of the *smallest enclosing rectangle* of the point with label 1 (i.e., the blue points in Figure 6.1). It is clear that if we have *lots* of sampled points, then we should get a good approximation to the "true" rectangle that generated the data, while with few points this is not possible. How can we quantify this? Let $\epsilon > 0$ and $\delta \in (0, 1)$ be given, and let X be a random point with associated label $Y = f(X)$. Then $\mathbb{P}(f(X) = 1)$ is the probability measure of the true rectangle, while

$$R(\hat{h}) = \mathbb{P}(\hat{h}(X) \neq f(X))$$

[1] In some references, such as the book "Foundations of Machine Learning" by Mohri, Rostamizadeh and Talwalkar, this version of PAC learning is called *Agnostic PAC Learning*.

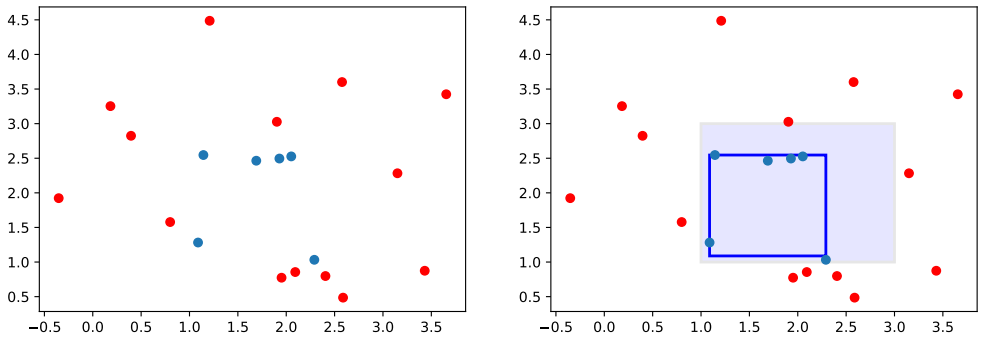


Figure 6.1: The blue points are in the true rectangle, while the red points are outside of it. The smaller blue rectangle represents the computed classifier $\hat{h}$, while the shaded are corresponds to the true rectangle that generated the original labelling.

is the risk of $\hat{h}$. We would like to find out the number of samples that would ensure

$$\mathbb{P}(R(\hat{h}) \leq \epsilon) \geq 1 - \delta.$$

First, note that since the rectangle defined by $\hat{h}$ is always contained in the true rectangle that we would like to discover, we can only get false negatives from $\hat{h}$ (that is, if $\hat{h}(x) = 1$ then x is in the true rectangle, but there may be points x in the true rectangle for which $\hat{h}(x) = 0$). Hence,

$$\begin{aligned} R(\hat{h}) &= \mathbb{P}(\hat{h}(X) \neq f(X)) \\ &= \underbrace{\mathbb{P}(\hat{h}(X) = 1, f(X) = 0)}_{=0} + \mathbb{P}(\hat{h}(X) = 0, f(X) = 1) \\ &= \mathbb{P}(\hat{h}(X) = 0, f(X) = 1). \end{aligned}$$

If we denote by $\hat{B} = \{x \in \mathbb{R}^2 \colon \hat{h}(x) = 1\}$ the *computed* smallest enclosing rectangle, then the risk $R(\hat{h})$ can be described more geometrically as

$$R(\hat{h}) = \mathbb{P}(X \in B \backslash \hat{B}),$$

namely the probability of an input being in the true rectangle but not in the computed one.

Now let $\epsilon > 0$ and $\delta \in (0, 1)$ be given. If $\mathbb{P}(X \in B) \leq \epsilon$, then clearly also $R(\hat{h}) \leq \epsilon$. Assume therefore that $\mathbb{P}(X \in B) > \epsilon$. Denote by R_i, $i \in \{1, 2, 3, 4\}$, the smallest sub-rectangles or B with $\mathbb{P}(X \in R_i) \geq \epsilon/4$ that bound each of the four sides of B, respectively (see Figure 6.2). We could, for example, start with the whole rectangle and move one of its sides towards the opposite side for as long as the measure is not less than $\epsilon/4$.

Denote by R_i° the rectangles with their inward-facing sides removed. Then clearly the probability measure of the union of these sets is $\mathbb{P}(X \in \bigcup_i R_i^\circ) \leq \epsilon$, since the measure of each of the R_i° is at most $\epsilon/4$. If the computed rectangle $\hat{B}$ intersects all the R_i, then

$$\mathbb{P}(X \in B \backslash \hat{B}) = \mathbb{P}\left(X \in \bigcup_i R_i^\circ \backslash \hat{B}\right) \leq \epsilon.$$

We now need to show that the probability that $\hat{B}$ *does not* intersect all the rectangles is small:

$$\mathbb{P}(\exists i \colon \hat{B} \cap R_i = \emptyset) = \mathbb{P}\left(\bigcup_i \{\hat{B} \cap R_i = \emptyset\}\right) \leq \sum_{i=1}^{4} \mathbb{P}(\hat{B} \cap R_i = \emptyset),$$

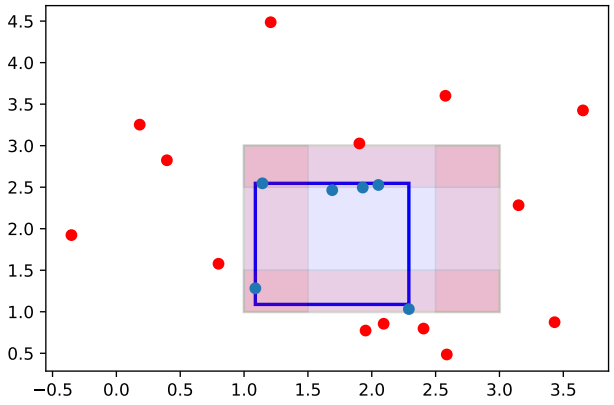


Figure 6.2: Four boundary regions with probability mass $\epsilon/4$ each.

where we used the union bound. The probability that $\hat{B}$ does not intersect one of the rectangles R_i, each of which has probability mass $\epsilon/4$, is equal to the probability that the n randomly sampled points that gave rise to $\hat{B}$ do not fall in R_i. For each of these points, the probability of *not* falling into R_i is at most $1 - \epsilon/4$, so the probability that none of the points falls into R_i is $(1 - \epsilon/4)^n$. Hence,

$$\mathbb{P}(\exists i\colon \hat{B} \cap R_i = \emptyset) \leq 4(1 - \epsilon/4)^n \leq 4\mathrm{e}^{-n\epsilon/4},$$

where we used the inequality $1 - x \leq \mathrm{e}^{-x}$. Setting the righ-hand side to δ, we conclude that if

$$n \geq \frac{4}{\epsilon} \log\left(\frac{4}{\delta}\right)$$

then $R(\hat{h}) \leq \epsilon$ holds with probability at least $1 - \delta$.

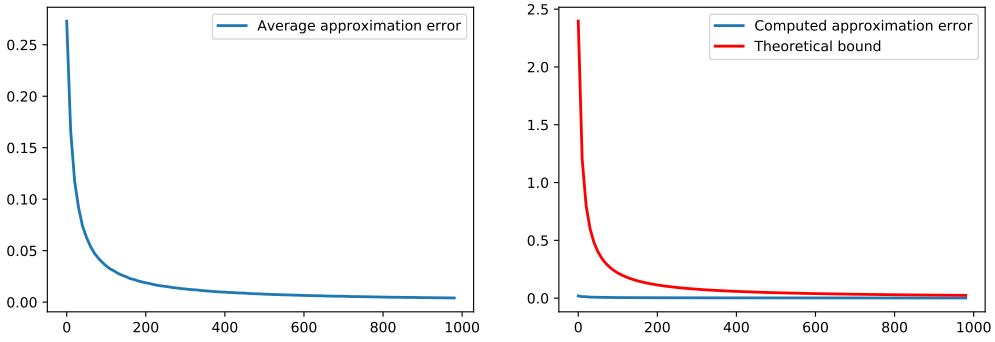


Figure 6.3: The left graphs shows the *average* risk as n increases. The right right graph shows the risk bound ϵ when given a confidence $1 - \delta = 0.99$ (blue curve), and the theoretical generalization bound as derived in the example.

Remark 6.3. Note that we did not make any assumptions on the probability distribution when deriving the bound on n in the rectangle-learning example. If the distribution is absolutely continuous with respect

to the Lebesgue measure on $\mathbb{R}^2$, then we could have required the probability measures of the rectangles to be exactly $\epsilon/4$, but the way the proof is written it also applies to distributions that are not supported on all of $\mathbb{R}^2$, such as the uniform distribution on a compact subset of $\mathbb{R}^2$ that may or may not cover the area of B, or a discrete distribution supported on countably many points. The requirement $\mathbb{P}(X \in B) > \epsilon$ still ensures that enough probability mass is contained within the confines of B for the argument to work. We may, however, end up looking at degenerate cases where, for example, all the probability mass is on an edge, one of the rectangles R_i is an edge or the whole of B, etc. Note that in such cases the intuitive view of the generalization risk as the "area" of the complement $B\backslash\hat{B}$ is no longer accurate! In practice we will only consider distributions that are natural to the situation under consideration.

Notes

The PAC framework is due to Warwick alumnus and Turing Award laureate Leslie Valiant [78]. Our presentation of the problem of learning rectangles is based on [56, Chapter 2].

7

Rademacher Complexity

The key to deriving generalization bounds for sets of binary classifiers $\mathcal{H}$ was to bound the maximum possible difference between the empirical risk and its expected value,

$$\sup_{h\in\mathcal{H}} |\hat{R}(h) - R(h)|. \tag{A}$$

For finite set of binary classifiers $\mathcal{H}$, the probability that this quantity exceeds a given t can be bounded using the union bound. Here we derive a different method that is based on an intrinsic complexity measure of $\mathcal{H}$ instead of its cardinality, and that applies in greater generality.

Rademacher Complexity

A random variable σ with values in $\{1, -1\}$ has the **Rademacher distribution** if $\mathbb{P}(\sigma = 1) = \mathbb{P}(\sigma = -1) = 1/2$. A Rademacher vector $\boldsymbol{\sigma} = (\sigma_1, \dots, \sigma_n)^\top$ is a random vector with i.i.d. Rademacher distributed coordinates.

Definition 7.1 (Rademacher complexity of a set)**.** The Rademacher complexity of a set $S \subset \mathbb{R}^n$ is defined as

$$\mathcal{R}(S) = \mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{\boldsymbol{x}\in S} \frac{\boldsymbol{\sigma}^\top \boldsymbol{x}}{n}\right] = \mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{\boldsymbol{x}\in S} \frac{1}{n}\sum_{i=1}^{n} \sigma_i x_i\right], \tag{7.1}$$

where the expectation is with respect to a Rademacher vector $\boldsymbol{\sigma}$.

One often encounters the definition without the absolute values. Since the distribution of $\boldsymbol{\sigma}$ is discrete, we could also rewrite the empirical Rademacher complexity as average:

$$\mathcal{R}(S) = \frac{1}{2^n} \sum_{\boldsymbol{\sigma}\in\{-1,1\}^n} \left(\sup_{g\in\mathcal{G}} \frac{\boldsymbol{\sigma}^\top \boldsymbol{x}}{n}\right).$$

We next consider a set $\mathcal{Z}$ with a probability distribution and a class of functions $\mathcal{G} = \{g\colon \mathcal{Z} \to \mathbb{R}\}$. For any $\boldsymbol{z} = (z_1, \dots, z_n) \in \mathcal{Z}^n$, we have the set of values $S = \{g(\boldsymbol{z})\colon g \in \mathcal{G}\}$, where $g(\boldsymbol{z}) = (g(z_1), \dots, g(z_m))^\top$.

Definition 7.2 (Rademacher Complexity of a class of functions)**.** Let $\mathcal{G}$ be a class of real-valued functions on $\mathcal{Z}$ and let $\boldsymbol{z} \in \mathcal{Z}^n$. The **empirical Rademacher complexity** of $\mathcal{G}$ with respect to $\boldsymbol{z}$ is defined as

$$\hat{\mathcal{R}}_{\boldsymbol{z}}(\mathcal{G}) = \mathcal{R}(S), \tag{7.2}$$

where $S = \{g(\boldsymbol{z})\colon g \in \mathcal{G}\}$. The **Rademacher complexity** of $\mathcal{G}$ is

$$\mathcal{R}_n(\mathcal{G}) = \mathbb{E}_{\boldsymbol{Z}}[\hat{\mathcal{R}}_{\boldsymbol{Z}}(\mathcal{G})], \tag{7.3}$$

where the expectation is with respect to a random vector $\boldsymbol{Z} = (Z_1, \dots, Z_n)$ with i.i.d. coordinates.

Note that the expected value in (7.2) is only over the random signs, and that the point $\boldsymbol{z}$ is fixed. It is only in (7.3) that we replace the point by a random vector $\boldsymbol{Z}$ and take the expectation. We have three definitions here: the Rademacher complexity of a set, of a class of functions with respect to a point $\boldsymbol{z}$, and of a class of functions on a space with a probability distribution.

In what follows, $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ is a data space, and $Z_i = (X_i, Y_i)$, $i \in [n]$, are random i.i.d. data points with points in $\mathcal{Z}$. We assume a set $\mathcal{H}$ of functions $h\colon \mathcal{X} \to \mathcal{Y}$ is given (not necessarily binary classifiers), and fix a bounded loss function $L(h(x), y) \leq C$. Then to every $h \in \mathcal{H}$ we associate a function $g\colon \mathcal{Z} \to \mathbb{R}$ by setting

$$g(z) = L(h(x), y)$$

and we denote the class of these functions by $\mathcal{G}$ (thus every $h \in \mathcal{H}$ gives rise to a function $g \in \mathcal{G}$, though the association $h \mapsto g$ need not be injective). Using this notation, we can express the supremum (A) as

$$\sup_{h \in \mathcal{H}} |\hat{R}(h) - R(h)| = \sup_{g \in \mathcal{G}} \left| \mathbb{E}[g(Z)] - \frac{1}{n} \sum_{i=1}^{n} g(Z_i) \right|,$$

where we recall that $R(h) = \mathbb{E}[L(h(X), Y)]$ and $\hat{R}(h) = {}^1\!/_n \sum_{i=1}^n L(h(X_i), Y_i)$. Using this notion of complexity, we can derive the following bound.

Theorem 7.3. *Let $\delta \in (0, 1)$ be given. Then with probability at least $1 - \delta$,*

$$\sup_{g \in \mathcal{G}} \left| \mathbb{E}[g(Z)] - \frac{1}{n} \sum_{i=1}^{n} g(Z_i) \right| \leq 2\mathcal{R}_n(\mathcal{G}) + C\sqrt{\frac{\log(2/\delta)}{2n}}. \tag{7.4}$$

Example 7.4. Assume $\mathcal{H} = \{h\}$ consists of only one binary classifier. Then for any point $(z_1, \dots, z_n) \in \mathcal{Z}^n$, the values $g(z_i) = \mathbf{1}\{h(x_i) \neq y_i\}$ form a fixed 0-1 vector . The empirical Rademacher complexity is

$$\frac{1}{n}\, \mathbb{E}_{\boldsymbol{\sigma}} \left[\sum_{i=1}^{n} \sigma_i g(z_i) \right] = 0,$$

since for each $\boldsymbol{\sigma}$, $\boldsymbol{\sigma}^\top g(\boldsymbol{z})$ and $-\boldsymbol{\sigma}^\top g(\boldsymbol{z})$ are just as likely and cancel out.

Example 7.5. Let $\mathcal{H}$ be the set of *all* binary classifiers. It follows that for any $(z_1, \dots, z_n) \in \mathcal{Z}^n$, the set of vectors $(g(z_1), \dots, g(z_n))$ for $g \in \mathcal{G}$ runs through *all* binary 0-1 vectors. The empirical Rademacher complexity of the set of functions $\mathcal{G}$ is thus the same as the Rademacher complexity of the vertices of a hypercube $S = [0, 1]^n$ as a set. Note that for each sign vector $\boldsymbol{\sigma}$ we can always pick out a function g such that $g(z_i) = 1$ if $\sigma_i = 1$ and $g(z_i) = 0$ if $\sigma_i = -1$, and this function maximizes the sum $\sum_i \sigma_i g(z_i)$. From this observation it is not hard to conclude that the supremum $\sup_g \boldsymbol{\sigma}^\top g(\boldsymbol{z})$ is binomially distributed and that $\hat{\mathcal{R}}_{\boldsymbol{z}}(\mathcal{G}) = 1/2$.

Example 7.6. For a finite set of binary classifiers $\mathcal{H} = \{h_1, \dots, h_K\}$ and $\boldsymbol{z} \in \mathcal{Z}^n$, one can derive the bound

$$\hat{\mathcal{R}}_{\boldsymbol{z}}(\mathcal{G}) \leq \frac{r\sqrt{2\log(K)}}{n},$$

where $r = \max_{g \in \mathcal{G}} \sqrt{\sum_{i=1}^n g(z_i)^2}$. This bound is known as *Massart's Lemma*. In combination with Theorem 7.3, we recover (up to constants) the risk bounds derived for finite hypothesis sets.

Example 7.7. The Rademacher complexity of a set S equals the Rademacher complexity of the convex hull of S. For example, the Rademacher complexity of the hypercube $[0,1]^n$ equals the Rademacher complexity of the set of its vertices. Since there are 2^n vertices, Example 7.6 gives the bound $\sqrt{2\log(2)}$ (we used that $r = \sqrt{n}$ here). We saw in Example 7.5 that the exact value is $1/2$.

The proof of Theorem 7.3 depends on yet another concentration of measure inequality for averages of random variables, namely McDiarmid's inequality.

Theorem 7.8 (McDiarmid's Inequality)**.** *Let $Z_1, \dots, Z_n$ be independent random variables and let $c_1, \dots, c_n$ be positive constants. Set $\boldsymbol{c} = (c_1, \dots, c_n)^\intercal$. Lef f be a real valued function such that for each $i \in \{1, \dots, n\}$ and values $z_i \neq z_i'$ we have*

$$|f(z_1, \dots, z_i, \dots, z_n) - f(z_1, \dots, z_i', \dots, z_n)| \leq c_i.$$

Set $\mu = \mathbb{E}[f(Z_1, \dots, Z_n)]$. Then for any $t \geq 0$,

$$\mathbb{P}\left(f(Z_1, \dots, Z_n) \geq \mu + t\right) \leq \exp\left(-\frac{2t^2}{\|\boldsymbol{c}\|^2}\right),$$
$$\mathbb{P}\left(f(Z_1, \dots, Z_n) \leq \mu - t\right) \leq \exp\left(-\frac{2t^2}{\|\boldsymbol{c}\|^2}\right).$$

Proof of Theorem 7.3. We first prove an upper bound on the function

$$\Phi(z_1, \dots, z_n) = \sup_{g \in \mathcal{G}} \mathbb{E}[g(Z)] - \frac{1}{n}\sum_{i=1}^{n} g(z_i), \tag{7.5}$$

which is the same as (7.4) but without the absolute value. Then for $i \in [n]$ and $z_i \neq z_i'$, and using the fact that the difference of suprema is not bigger than the supremum of a difference, and the difference of absolute values is not bigger than the absolute value of a difference, we get

$$\Phi(z_1, \dots, z_i, \dots, z_n) - \Phi(z_1, \dots, z_i', \dots, z_n) \leq \sup_{g \in \mathcal{G}} \frac{1}{n}|g(z_i') - g(z_i)| \leq \frac{C}{n},$$

from which we conclude that the function Φ satisfies the conditions of McDiarmid's inequality with $c_i = C/n$. Hence,

$$\mathbb{P}(\Phi(Z_1, \dots, Z_n) - \mathbb{E}[\Phi(Z_1, \dots, Z_n)] > t) \leq \mathrm{e}^{-2nt^2/C^2}.$$

Setting the right-hand bound to $\delta/2$ and resolving for t, we conclude that with probability at least $1 - \delta/2$,

$$\Phi(Z_1, \dots, Z_n) \leq \mathbb{E}[\Phi(Z_1, \dots, Z_n)] + C\sqrt{\frac{\log(2/\delta)}{2n}}.$$

The last, and crucial, step is to bound the expected value on the right-hand side with the Rademacher complexity. The idea is to introduce identical but independent copies Z_i' of the random variables Z_i. Denote by $\boldsymbol{Z}$ and $\boldsymbol{Z}'$ the vectors of random variables Z_i and Z_i', respectively. We will use repeatedly the fact that if $f(\boldsymbol{Z}')$ only depends on the random variables in $\boldsymbol{Z}'$, then $f(\boldsymbol{Z}') = \mathbb{E}_{\boldsymbol{Z}}[f(\boldsymbol{Z}')]$, that is we can pull an expression "into the expectation" if the terms involved are independent of the variables over which

the expectation is taken. We will also use the linearity of expectation repeatedly without explicitly saying so. We can then set

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{Z}}[\Phi(Z_1,\ldots,Z_n)] &= \mathbb{E}_{\boldsymbol{Z}}\left[\sup_{g\in\mathcal{G}}\mathbb{E}[g(Z)] - \frac{1}{n}\sum_{i=1}^{n} g(Z_i)\right] \\
&= \mathbb{E}_{\boldsymbol{Z}}\left[\sup_{g\in\mathcal{G}}\mathbb{E}_{\boldsymbol{Z}'}\left[\frac{1}{n}\sum_{i=1}^{n} g(Z_i')\right] - \frac{1}{n}\sum_{i=1}^{n} g(Z_i)\right] \\
&= \mathbb{E}_{\boldsymbol{Z}}\left[\sup_{g\in\mathcal{G}}\mathbb{E}_{\boldsymbol{Z}'}\left[\frac{1}{n}\sum_{i=1}^{n} g(Z_i') - \frac{1}{n}\sum_{i=1}^{n} g(Z_i)\right]\right] \\
&\leq \mathbb{E}_{\boldsymbol{Z}}\left[\mathbb{E}_{\boldsymbol{Z}'}\left[\sup_{g\in\mathcal{G}}\frac{1}{n}\sum_{i=1}^{n}(g(Z_i') - g(Z_i))\right]\right] \\
&= \mathbb{E}_{\boldsymbol{Z},\boldsymbol{Z}'}\left[\sup_{g\in\mathcal{G}}\frac{1}{n}\sum_{i=1}^{n}(g(Z_i') - g(Z_i))\right],
\end{aligned}$$

where for the inequality we used the fact that the sup of an expectation is not more than the expectation of the sup. [1] We next use an idea known as *symmetrization*. The key observation is that the expectation does not change if we replace each summand $g(Z_i') - g(Z_i)$ with its negative, $g(Z_i) - g(Z_i')$. More generally, we can pick any sign vector $\boldsymbol{\sigma} = (\sigma_1,\ldots,\sigma_n)$ with $\sigma_i \in \{-1, 1\}$, and will then have

$$\mathbb{E}_{\boldsymbol{Z},\boldsymbol{Z}'}\left[\sup_{g\in\mathcal{G}}\frac{1}{n}\sum_{i=1}^{n}\sigma_i\cdot(g(Z_i') - g(Z_i))\right] = \mathbb{E}_{\boldsymbol{Z},\boldsymbol{Z}'}\left[\sup_{g\in\mathcal{G}}\frac{1}{n}\sum_{i=1}^{n}(g(Z_i') - g(Z_i))\right].$$

Now we use a "sheep counting trick"[2], and sum these terms over all possible sign patterns, dividing by the total number of sign patterns, 2^n:

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{Z},\boldsymbol{Z}'}\left[\sup_{g\in\mathcal{G}}\frac{1}{n}\sum_{i=1}^{n}(g(Z_i') - g(Z_i))\right] \\
&\quad= \frac{1}{2^n}\sum_{\boldsymbol{\sigma}\in\{-1,1\}^n}\mathbb{E}_{\boldsymbol{Z},\boldsymbol{Z}'}\left[\sup_{g\in\mathcal{G}}\frac{1}{n}\sum_{i=1}^{n}\sigma_i(g(Z_i') - g(Z_i))\right] \\
&\quad= \mathbb{E}_{\boldsymbol{\sigma},\boldsymbol{Z},\boldsymbol{Z}'}\left[\sup_{g\in\mathcal{G}}\frac{1}{n}\sum_{i=1}^{n}\sigma_i(g(Z_i') - g(Z_i))\right],
\end{aligned}$$

where for the last equality we simply rewrote the average as an expectation over a vector of Rademacher random variables. We can now bound the supremum of the differences by the difference of suprema in order to get

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{\sigma},\boldsymbol{Z},\boldsymbol{Z}'}\left[\sup_{g\in\mathcal{G}}\frac{1}{n}\sum_{i=1}^{n}\sigma_i(g(Z_i') - g(Z_i))\right] &\leq \mathbb{E}_{\boldsymbol{\sigma},\boldsymbol{Z}'}\left[\sup_{g\in\mathcal{G}}\frac{1}{n}\sum_{i=1}^{n}\sigma_i g(Z_i')\right] \\
&\quad + \mathbb{E}_{\boldsymbol{\sigma},\boldsymbol{Z}}\left[\sup_{g\in\mathcal{G}}\frac{1}{n}\sum_{i=1}^{n}(-\sigma_i g(Z_i))\right] \\
&= 2\mathcal{R}_n(\mathcal{G}).
\end{aligned}$$

[1] Prove this.

[2] When a shepherd wants to count her sheep, she counts the legs and divides the result by four.

The last equality shows why we averaged over all sign vectors: by symmetry, averaging over $-\boldsymbol{\sigma}$ is the same as averaging over $\boldsymbol{\sigma}$. To get the full result, we merely have to repeat the argument with expectation and sum in (7.5) reversed; we omit the details. Together, we get that

$$\mathbb{P}\left(\sup_{g\in\mathcal{G}} \mathbb{E}[g(Z)] - \frac{1}{n}\sum_{i=1}^{n} g(Z_i) > 2\mathcal{R}_n(\mathcal{G}) + C\sqrt{\frac{\log(2/\delta)}{2n}}\right) \leq \frac{\delta}{2}$$
$$\mathbb{P}\left(\sup_{g\in\mathcal{G}} \frac{1}{n}\sum_{i=1}^{n} g(Z_i) - \mathbb{E}[g(Z)] > 2\mathcal{R}_n(\mathcal{G}) + C\sqrt{\frac{\log(2/\delta)}{2n}}\right) \leq \frac{\delta}{2},$$

and with the union bound, the claimed result follows. □

It is important to understand why the symmetrization is necessary, and why we could not simply have decoupled the difference before averaging over random sign vectors. The empirical Rademacher complexity $\hat{\mathcal{R}}_{\boldsymbol{z}}$ is a function of $(z_1, \dots, z_n)$, and by the same argument as for the Φ function in the proof of Theorem 7.3 one can show that if $\boldsymbol{z}'$ arises from $\boldsymbol{z}$ by changing z_i to z_i', then

$$|\hat{\mathcal{R}}_{\boldsymbol{z}'}(\mathcal{G}) - \hat{\mathcal{R}}_{\boldsymbol{z}}(\mathcal{G})| \leq \frac{C}{n}.$$

We can therefore apply McDiarmid's inequality to the random variable $\hat{\mathcal{R}}_{\boldsymbol{Z}}(\mathcal{G})$ to conclude that

$$\hat{\mathcal{R}}_{\boldsymbol{Z}}(\mathcal{G}) \leq \mathcal{R}_n(\mathcal{G}) + C\sqrt{\frac{\log(1/\delta)}{2n}} \tag{7.6}$$

with probability at least $1-\delta$. One can combine (7.6) with (7.4) using the union bound to get a generalization bound analogous to (7.4) but in terms of the empirical Rademacher complexity (with slightly different parameters).

To conclude, we note that if $\mathcal{H}$ is a set of binary classifiers with values in $\{-1, 1\}$ (instead of $\{0, 1\}$), then we can relate the Rademacher complexity of $\mathcal{G}$ to that of $\mathcal{H}$ as follows.

Lemma 7.9. *The Rademacher complexities of $\mathcal{H}$ and $\mathcal{G}$ satisfy $\mathcal{R}_n(\mathcal{G}) = \frac{1}{2}\mathcal{R}_n(\mathcal{H})$.*

The proof uses $\mathbf{1}\{h(x) \neq y\} = (1 - yh(x))/2$, and is left as an exercise.

Notes

Generalization bounds in terms of the Rademacher complexity were introduced in [3], and a recommended classic reference is [4]. Symmetrization is a classic technique in the theory of empirical processes in statistics, a general reference is [50] and an accessible overview can be found in [28, Section 8.2]. Replacing the Rademacher random vector with a Gaussian vector leads to the notion of Gaussian width, which plays an important role in the study of suprema of random processes. To see this connection, let $\boldsymbol{g}$ be a random vector (Gaussian or more general) and consider for each $\boldsymbol{x} \in S$ the random variable $X_{\boldsymbol{x}} = \langle \boldsymbol{g}, \boldsymbol{x} \rangle = \boldsymbol{g}^\top \boldsymbol{x}$. This is then a random process indexed by the elements of S, and we are interested in studying the supremum

$$\sup_{\boldsymbol{x}\in S} X_{\boldsymbol{x}}.$$

For a comprehensive treatment of this and related concepts, we refer to [83].

8

VC Theory

When studying Rademacher complexity we did not make any assumptions on the space of responses or labels $\mathcal{Y}$. In this chapter we go back to the setting of binary classification, and this time it is more convenient to use $\mathcal{Y} = \{-1, 1\}$ and the unit loss function. For the supremum of the difference between empirical and generalization risk we derived a bound

$$\sup_{h \in \mathcal{H}} |\hat{R}(h) - R(h)| \leq 2\mathcal{R}_n(\mathcal{G}) + \sqrt{\frac{\log(2/\delta)}{2n}}, \tag{8.1}$$

where $\mathcal{R}_n(\mathcal{G})$ is the Rademacher complexity. The definition of Rademacher complexity involved taking the expectation with respect to a distribution on the space $\mathcal{Z}$ that we do not know. We saw, however, that in some examples we could bound this expectation in a way that does not depend on the distribution. We next develop this idea in a more principled way, deriving generalization bounds in terms of parameters of the set of classifiers $\mathcal{H}$ that do not make reference to the underlying distribution.

Vapnik-Chervonenkis Theory

In binary classification, a classifier h can take at most two possible values on a fixed input. Denote by $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathcal{X}^n$ an n-tuple of inputs and set

$$h(\boldsymbol{x}) := (h(x_1), \ldots, h(x_n)) \in \{-1, 1\}^n$$

for the corresponding vector of values of the classifier. For any fixed $\boldsymbol{x} \in \mathcal{X}^n$, we could get up to 2^n different values $h(\boldsymbol{x})$ as h runs through $\mathcal{H}$. Note that his is a finite bound even if the set $\mathcal{H}$ is infinite! A possible classification $h(\boldsymbol{x})$ is called a **dichotomy** and one way to measure the expressiveness or richness of a set of classifiers $\mathcal{H}$ would be to count the possible dichotomies.

Example 8.1. Let $\mathcal{X} = \mathbb{R}$ and let $\mathcal{H}$ be the set of "indicator functions" of closed half-lines: for each $a \in \mathbb{R}$,

$$h_a^+(x) = \begin{cases} 1 & x \geq a \\ -1 & x < a \end{cases}, \quad h_a^-(x) = \begin{cases} 1 & x \leq a \\ -1 & x > a \end{cases}.$$

Given two distinct samples, $\{x_1, x_2\}$, there are 4 possible dichotomies: for each pattern $\rho \in \{-1, 1\}^2$ we can find $h \in \mathcal{H}$ such that the tuple $(h(x_1), h(x_2)) = \rho$. For three distinct points $\{x_1, x_2, x_3\} \subset \mathbb{R}$ this is no longer possible. If we assume, for example, that $x_1 < x_2 < x_3$, then any classifier h with $h(x_1) = -1$ and $h(x_2) = 1$ will automatically also satisfy $h(x_3) = 1$.

Clearly, if $\mathcal{H}$ consists of only one element, then for every $\boldsymbol{x} \in \mathcal{X}^n$ there is only one possible dichotomy, while if $\mathcal{H}$ consists of *all* classifiers then there are 2^n possible dichotomies if the entries of $\boldsymbol{x}$ are distinct. Somewhere in between these two extreme cases, the number of dichotomies may depend on $\boldsymbol{x}$ in more intricate ways.

Definition 8.2. Let $\mathcal{H}$ be a set of classifiers $h\colon \mathcal{X} \to \mathcal{Y}$. The **growth function** $\Pi_{\mathcal{H}}$ is defined as

$$\Pi_{\mathcal{H}}(n) = \max_{\boldsymbol{x} \in \mathcal{X}^n} \left| \{h(\boldsymbol{x})\colon h \in \mathcal{H}\} \right|.$$

Note that this function depends only on the set $\mathcal{H}$. We can use it to bound the Rademacher complexity of a set of functions $\mathcal{G}$. The bound depends on a result known as Massart's Lemma, which is derived in the Exercises. Recall that the Rademacher complexity of a set $S \subset \mathbb{R}^n$ is defined as

$$\mathcal{R}(S) = \frac{1}{n}\, \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{\boldsymbol{x} \in S} \sum_{i=1}^{n} \sigma_i x_i \right],$$

where $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$ is a vector of i.i.d. Rademacher random variables.

Lemma 8.3. *(Massart's Lemma) If $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_K\} \subset \mathbb{R}^n$, then*

$$\mathcal{R}(S) \leq \frac{r \cdot \sqrt{2 \log(K)}}{n},$$

where $r = \max_{\boldsymbol{x} \in S} \|\boldsymbol{x}\|$ and $\|\boldsymbol{x}\| := \sqrt{\sum_{i=1}^{n} x_i^2}$.

Theorem 8.4. *The Rademacher complexity of $\mathcal{H}$ is bounded by*

$$\mathcal{R}_n(\mathcal{H}) \leq \sqrt{\frac{2 \log(\Pi_{\mathcal{H}}(n))}{n}}.$$

Proof. Consider a fixed tuple $\boldsymbol{x} = (x_1, \ldots, x_n)$ such that

$$\Pi_{\mathcal{H}}(n) = \left| \{h(\boldsymbol{x})\colon h \in \mathcal{H}\} \right|.$$

The vectors $(h(x_1), \ldots, h(x_n))$ all have norm $\sqrt{n}$, and the claim follows from Massart's Lemma. □

Combining this with (8.1) and Lemma 7.9 in Chapter 7 (which states that $2\mathcal{R}_n(\mathcal{G}) = \mathcal{R}_n(\mathcal{H})$), we immediately arrive at the following bound.

Corollary 8.5. *For a set of binary classifiers $\mathcal{H}$,*

$$\sup_{h \in \mathcal{H}} |\hat{R}(h) - R(h)| \leq \sqrt{\frac{2 \log(\Pi_{\mathcal{H}}(n))}{n}} + \sqrt{\frac{\log(2/\delta)}{2n}}. \tag{8.2}$$

Let $\mathcal{H}$ be a set of classifiers and $S = \{x_i\}_{i=1}^n \subset \mathcal{X}$ a set of inputs. Then S is **shattered** by $\mathcal{H}$, if

$$\left| \{h(\boldsymbol{x})\colon h \in \mathcal{H}\} \right| = 2^n,$$

where $\boldsymbol{x} = (x_1, \ldots, x_n)$.

Example 8.6. If $\mathcal{H}$ is the set of all binary classifiers and all the samples in S are distinct, then S is shattered by $\mathcal{H}$.

Example 8.7. If $\mathcal{H} = \{h_1, h_2\}$ consists of only two classifiers, one of which is constant 1 and the other is constant -1, then a subset $S \subset \mathcal{X}$ of samples is shattered by $\mathcal{H}$ if and only if $|S| = 1$, that is, we only look at one sample.

There exists a subset $S = \{x_i\}_{i=1}^n \subset \mathcal{X}$ that can be shattered by $\mathcal{H}$ if and only if

$$\Pi_{\mathcal{H}}(n) = 2^n.$$

If the number of samples n increases, it can become harder to find a subset that can be shattered. While in the case where $\mathcal{H}$ consists of all binary classifiers we always have $\Pi_{\mathcal{H}}(n) = 2^n$, in the case where $\mathcal{H}$ consists of indicator functions of half-lines we saw that three distinct points on the line cannot be shattered. The maximum possible n such that a subset of n points can be shattered is the VC dimension.

Definition 8.8. The Vapnik-Chernovenkis (VC) dimension of a set of classifiers $\mathcal{H}$ is

$$\mathrm{VC}(\mathcal{H}) = \max\{n \in \mathbb{N} \colon \Pi_{\mathcal{H}}(n) = 2^n\}.$$

If $\mathrm{VC}(\mathcal{H}) = d$, then there exists a set of d samples that can be shattered by $\mathcal{H}$: all possible dichotomies occur when applying classifiers in $\mathcal{H}$ to these n samples. Note that this does not mean, however, that *all* sets of n samples can be shattered by $\mathcal{H}$.

We will see that if $\mathcal{H}$ has VC dimension $\mathrm{VC}(\mathcal{H}) = d$, then we can bound

$$\log(\Pi_{\mathcal{H}}(n)) \leq d \log\left(\frac{\mathrm{e}d}{n}\right),$$

which allows to rephrase the bound (8.2) in terms of the VC dimension. We will then study plenty of examples (including practical ones) where we can compute or bound the VC dimension.

VC dimension of families of sets

The notion of VC dimension was defined for classes of functions $\mathcal{H}$. Equivalently, we can identify each h with the indicator function of a set $A \subset \mathcal{X}$ and consider the set of sets

$$\mathcal{A} = \{A \subset \mathcal{X} \colon h(x) = 1 \Leftrightarrow x \in A\}.$$

Given a subset $S \subset \mathcal{X}$, we say that $\mathcal{A}$ *shatters* S, if every subset of S (including the empty set) can be obtained by intersecting S with elements of $\mathcal{A}$:

$$\{A \cap S \colon A \in \mathcal{A}\} = \mathcal{P}(S) := \{S' \colon S' \subset S\}.$$

In other words, we can use the collection $\mathcal{A}$ to *select* any subset of S. It should be clear that if S is shattered by $\mathcal{A}$, then so is any subset of S. In this context, the growth function is defined analogously,

$$\Pi_{\mathcal{A}}(n) = \max\{|\{A \cap S \colon A \in \mathcal{A}\}| \colon S \subset \mathcal{X}, |S| = n\},$$

as the maximal number of subsets of a set of cardinality n that can be selected using $\mathcal{A}$. The VC dimension of the set system $\mathcal{A}$ is

$$\mathrm{VC}(\mathcal{A}) = \max\{n \in \mathbb{N} \colon \Pi_{\mathcal{A}}(n) = 2^n\}.$$

Note that the VC dimension is monotone in the sense that it does not decrease when $\mathcal{A}$ is enlarged. One of the most important results in VC theory is a bound on $\Pi_{\mathcal{A}}(n)$ in terms of the VC dimension. This result is usually attributed to Sauer (who credits Perles) and Shelah, but was discovered independently by Vapnik & Chervonenkis.

Lemma 8.9. *If* $\mathrm{VC}(\mathcal{A}) = d$*, then for* $n \geq d$*,*

$$\Pi_{\mathcal{A}}(n) \leq \sum_{i=0}^{d} \binom{n}{i} \leq \left(\frac{\mathrm{e}n}{d}\right)^{d}.$$

Proof. The proof of the first inequality is by induction on $n + d$. If $d = 0$ and $n = 0$, then $\Pi_{\mathcal{A}}(0) = 1$ (only the empty set can be considered) and the bound is valid. The statement is also easily verified for $n = 1$ and $d \leq 1$. Assume now that $n > 0$ and $d > 0$, and that the statement holds for the pairs $(d, n-1)$ and $(d-1, n-1)$. Let $S \subset \mathcal{X}$ be a subset with $|S| = n$, such that $|\{A \cap S \colon A \in \mathcal{A}\}| = \Pi_{\mathcal{A}}(n)$, and select an arbitrary element $s \in S$. Consider the sets

$$\mathcal{A}' = \{A \backslash \{s\} \colon A \in \mathcal{A}\}, \quad \mathcal{A}'' = \{A \in \mathcal{A} \colon s \notin A,\ A \cup \{s\} \in \mathcal{A}\}.$$

Note that

$$\mathrm{VC}(\mathcal{A}') \leq \mathrm{VC}(\mathcal{A}) = d, \quad \text{and} \quad \mathrm{VC}(\mathcal{A}'') \leq \mathrm{VC}(\mathcal{A}) - 1 = d - 1.$$

The first inequality follows from the fact that if $\mathcal{A}'$ shatters a set T, then so does $\mathcal{A}$. The second inequality follows from the fact that if a set T is shattered by $\mathcal{A}''$, then the set $T \cup \{s\}$ is shattered by $\mathcal{A}$.

Consider the map

$$\begin{aligned} \{A \cap S \colon A \in \mathcal{A}\} \to \{A \cap S \colon A \in \mathcal{A}'\} &= \{A \cap S \backslash \{s\} \colon A \in \mathcal{A}\}, \\ A \cap S &\mapsto A \cap S \backslash \{s\}. \end{aligned}$$

This map is one-to-one, except in the case where $A \in \mathcal{A}''$. For such A, both $s \notin A$ and $\tilde{A} = A \cup \{s\} \in \mathcal{A}$ hold, and therefore $A \cap S \neq \tilde{A} \cap S$, but

$$A \cap S \backslash \{s\} = \tilde{A} \cap S \backslash \{s\}.$$

It follows that

$$\begin{aligned} |\{A \cap S \colon A \in \mathcal{A}\}| &= |\{A \cap S \backslash \{s\} \colon A \in \mathcal{A}'\}| + |\{A \cap S \colon A \in \mathcal{A}''\}| \\ &= |\{A \cap (S \backslash \{s\}) \colon A \in \mathcal{A}'\}| + |\{A \cap (S \backslash \{s\}) \colon A \in \mathcal{A}''\}|. \end{aligned}$$

By the induction hypothesis,

$$\begin{aligned} |\{A \cap (S \backslash \{s\}) \colon A \in \mathcal{A}'\}| &\leq \Pi_{\mathcal{A}'}(n-1) \leq \sum_{i=0}^{d} \binom{n-1}{i}, \\ |\{A \cap (S \backslash \{s\}) \colon A \in \mathcal{A}''\}| &\leq \Pi_{\mathcal{A}''}(n-1) \leq \sum_{i=0}^{d-1} \binom{n-1}{i}, \end{aligned}$$

so that combining the terms we get

$$|\{A \cap S \colon A \in \mathcal{A}\}| \leq 1 + \sum_{i=1}^{d} \left[\binom{n-1}{i} + \binom{n-1}{i-1} \right] = \sum_{i=0}^{d} \binom{n}{i}.$$

For the second claimed inequality, we extend the sum to n and multiply each summand by $(n/d)^{d-i}$, to obtain

$$\begin{aligned}\sum_{i=0}^{d}\binom{n}{i} &\le \sum_{i=0}^{n}\binom{n}{i}\left(\frac{n}{d}\right)^{d-i}\\ &= \left(\frac{n}{d}\right)^{d}\sum_{i=0}^{n}\binom{n}{i}\left(\frac{d}{n}\right)^{i}\\ &= \left(\frac{n}{d}\right)^{d}\left(1+\frac{d}{n}\right)^{n} \le \left(\frac{\mathrm{e}n}{d}\right)^{d},\end{aligned}$$

where we used the inequality $1 + x \le \mathrm{e}^x$ at the end. □

The VC Inequality

A central result in statistical learning is an inequality that relates the VC dimension of a set of classifiers to the difference between the empirical and the generalization risk.

Theorem 8.10. ***(VC Inequality)*** *Let $\mathcal{H}$ be a set of classifiers $h\colon \mathcal{X} \to \{-1, 1\}$ with VC dimension $\mathrm{VC}(\mathcal{H}) = d$, and let $\delta \in (0, 1)$. Then with probability at least $1 - \delta$,*

$$\sup_{h\in\mathcal{H}} |R(h) - \hat{R}(h)| \le \sqrt{\frac{2d\log\left(\frac{\mathrm{e}n}{d}\right)}{n}} + \sqrt{\frac{\log(2/\delta)}{2n}}.$$

Proof. In Corollary 8.6 we saw that

$$\sup_{h\in\mathcal{H}} |R(h) - \hat{R}(h)| \le \sqrt{\frac{2\log\left(\Pi_{\mathcal{H}}(n)\right)}{n}} + \sqrt{\frac{\log(2/\delta)}{2n}}.$$

The claim now follows directly from Lemma 8.9. □

Rectangle learning revisited

Let $\mathcal{H}$ be the set of functions that take the value 1 on rectangles in the plane $\mathcal{X} = \mathbb{R}^2$ and -1 otherwise[1]. The question is:

Given n, can we find a configuration of n points in the plane such that for *any* labelling we can find a rectangle containing those points labelled with 1?

This is clearly possible when $n = 2$ (just choose two distinct points) and $n = 3$ (choose three points that form a triangle such as △). For $n = 4$ there are 16 possible labellings, and if we arrange the points in diamond form ⋄, then all labellings can be captured by rectangles (try this!). For $n = 5$ this is no longer possible: take the smallest enclosing rectangle of five points $\{x_i\}_{i=1}^5$. This rectangle will contain (at least) one of the x_i on each boundary (if not, we could make it smaller). If each x_i, $i \in \{1, \dots, 4\}$, lies on a different boundary, we can assign 1 to these points and -1 to x_5. This dichotomy cannot be realized by a rectangle: any rectangle containing $\{x_i\}_{i=1}^4$ must also contain their smallest enclosing rectangle, hence also x_5.

[1] Depending on context, we will consider $\mathcal{H}$ as consisting of functions into $\{0, 1\}$ or into $\{-1, 1\}$, this does not alter any of the results involving the VC dimension.

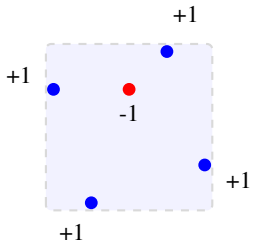


Figure 8.1: A dichotomy that is not captured by rectangles.

Notes

VC Theory is a central part of statistical learning theory. The idea developed out of the work of Vapnik and Chervonenkis [81], a comprehensive account of their theory is given in [80]. The concept of VC dimension has come to play an important role in other fields such as computational and discrete geometry, see for example [53]. Lemma 8.9 is commonly referred to as the Sauer-Shelah Lemma, after Norbert Sauer [69] and Saharon Shelah [73], and was discovered independently (and earlier) by Vapnik and Chervonenkis [81].

Part II

Elements of Optimization

9

Overview of Optimization

"[N]othing at all takes place in the universe in which some rule of maximum or minimum does not appear."
— Leonhard Euler

Mathematical optimization, traditionally also known as mathematical programming, is the theory of optimal decision making. Other than in machine learning, optimization problems arise in a large variety of contexts, including scheduling and logistics problems, finance, optimal control and signal processing. The underlying mathematical problem always amounts to finding parameters that minimize (cost) or maximize (utility) an objective function in the presence or absence of a set of constraints. In the context of machine learning, the objective function is usually related to the empirical risk, but we first take a step back and consider optimization problems in greater generality.

What is an optimization problem?

A general mathematical optimization problem is a problem of the form

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{w}) \\ &\text{subject to} && \boldsymbol{w} \in \Omega \end{aligned} \tag{9.1}$$

where $f\colon \mathbb{R}^d \to \mathbb{R}$ is a real-valued **objective function** and $\Omega \subseteq \mathbb{R}^d$ is a set defining the **constraints**. In the context of machine learning, the variables $\boldsymbol{w}$ are usually the parameters defining a classifier or predictor, such as the weights of a support vector machine or neural network. If $\Omega = \mathbb{R}^d$, then the problem is an **unconstrained optimization problem**. The constraint set Ω often consists of $\boldsymbol{w} \in \mathbb{R}^d$ that satisfy certain equations and inequalities,

$$f_1(\boldsymbol{w}) \leq 0, \ldots, f_m(\boldsymbol{w}) \leq 0, g_1(\boldsymbol{w}) = 0, \ldots, g_p(\boldsymbol{w}) = 0.$$

A vector $\boldsymbol{w}^*$ satisfying the constraints is called an **optimum**, a **solution**, or a **minimizer** of the problem (9.1), if $f(\boldsymbol{w}^*) \leq f(\boldsymbol{w})$ for all other $\boldsymbol{w}$ that satisfy the constraints. Note that replacing f by $-f$, we could equivalently state the problem as a maximization problem.

Optimality conditions

In what follows we will study the unconstrained problem

$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{minimize}}\, f(\boldsymbol{w}). \tag{9.2}$$

Optimality conditions aim to identify properties that potential minimizers need to satisfy in relation to $f(\boldsymbol{w})$. We will review the well known local optimality conditions for differentiable functions from calculus. We first identify different types of minimizers.

Definition 9.1. A point $\boldsymbol{w}^* \in \mathbb{R}^d$ is a

- *global minimizer* of (9.2) if for all $\boldsymbol{w} \in \mathbb{R}^d$, $f(\boldsymbol{w}^*) \leq f(\boldsymbol{w})$;
- a *local minimizer*, if there is an open neighbourhood U of $\boldsymbol{w}^*$ such that $f(\boldsymbol{w}^*) \leq f(\boldsymbol{w})$ for all $\boldsymbol{w} \in U$;
- a *strict local minimizer*, if there is an open neighbourhood U of $\boldsymbol{w}^*$ such that $f(\boldsymbol{w}^*) < f(\boldsymbol{w})$ for all $\boldsymbol{w} \in U$;
- an *isolated minimizer* if there is an open neighbourhood U of $\boldsymbol{w}^*$ such that $\boldsymbol{w}^*$ is the only local minimizer in U.

Without any further assumptions on f, finding a minimizer is a hopeless task: we cannot simply examine the function at *all* points in $\mathbb{R}^d$. The situation becomes more tractable if we assume some *smoothness* conditions. Recall that $C^k(U)$ denotes the set of functions that are k times continuously differentiable on some set U. The following *first-order* necessary condition for optimality is well known. We write $\nabla f(\boldsymbol{w})$ for the gradient of f at $\boldsymbol{w}$, i.e., the vector

$$\nabla f(\boldsymbol{w}) = \left(\frac{\partial f}{\partial w_1}(\boldsymbol{w}), \ldots, \frac{\partial f}{\partial w_d}(\boldsymbol{w})\right)^\intercal$$

Theorem 9.2. *Let $\boldsymbol{w}^*$ be a local minimizer of f and assume that $f \in C^1(U)$ for a neighbourhood of U of $\boldsymbol{w}^*$. Then $\nabla f(\boldsymbol{w}^*) = \boldsymbol{0}$.*

There are simple examples that show that this is not a sufficient condition: maxima and saddle points will also have a vanishing gradient. If we have access to *second-order information*, in form of the second derivative, or Hessian, of f, then we can say more. Recall that the Hessian of f at $\boldsymbol{w}$, $\nabla^2 f(\boldsymbol{w})$, is the $d \times d$ symmetric matrix given by the second derivatives,

$$\nabla^2 f(\boldsymbol{w}) = \left(\frac{\partial^2 f}{\partial w_i \partial w_j}\right)_{1\leq i,j\leq d}.$$

In the one-variable case we know that if w^* is a local minimizer of $f \in C^2([a,b])$, then $f'(w^*) = 0$ *and* $f''(w^*) \geq 0$. Moreover, the conditions $f'(w^*) = 0$ *and* $f''(w^*) > 0$ guarantee that we have a local minimizer. These conditions generalise to higher dimension, but first we need to know what $f''(w) > 0$ when we have more than one variable.

Recall also that a matrix $\boldsymbol{A}$ is **positive semidefinite**, written $\boldsymbol{A} \succeq \boldsymbol{0}$, if for every $\boldsymbol{w} \in \mathbb{R}^d$, $\boldsymbol{w}^\top \boldsymbol{A}\boldsymbol{w} \geq 0$, and positive definite, written $\boldsymbol{A} \succ \boldsymbol{0}$, if $\boldsymbol{w}^\top \boldsymbol{A}\boldsymbol{w} > 0$. The property that the Hessian matrix is positive semidefinite is a multivariate generalization of the property that the second derivative is nonnegative. The known conditions for a minimizer involving the second derivative generalize accordingly.

Theorem 9.3. *Let $f \in C^2(U)$ for some open set U and $\boldsymbol{w}^* \in U$. If $\boldsymbol{w}^*$ is a local minimizer, then $\nabla f(\boldsymbol{w}^*) = 0$ and $\nabla^2 f(\boldsymbol{w}^*)$ is positive semidefinite. Conversely, if $\nabla f(\boldsymbol{w}^*) = \mathbf{0}$ and $\nabla^2 f(\boldsymbol{w}^*)$ is positive definite, then $\boldsymbol{w}^*$ is a strict local minimizer.*

Unfortunately, the above criteria are not able to identify global minimizers, as differentiability is a local property. For **convex** functions, however, local optimality implies global optimality.

Examples

We present two examples of optimization problems that can be interpretred as machine learning problems, but have mainly been studied outside of the context of machine learning. The examples below come with associated Python code and it is not expected that you understand them in detail; they are merely intended to illustrate some of the problems that optimization deals with, and how they can be solved.

Example 9.4. Suppose we want to understand the relationship of a quantity y (for example, sales data) to a series of *predictors* $x_1, \dots, x_p$ (for example, advertising budget in different media). We can often assume the relationship to be *approximately linear*, with distribution

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon,$$

with noise ϵ that satisfies $\mathbb{E}[\epsilon] = 0$. In the setting of statistical learning, the input space is $\mathcal{X} = \mathbb{R}^p$, the output space is $\mathcal{Y} = \mathbb{R}$, and the set $\mathcal{H}$ consists of functions of the form

$$h(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p. \tag{9.3}$$

The goal is to determine the *model parameters* $\boldsymbol{w} = (\beta_0, \dots, \beta_p)^\intercal$ from observed data. To determine these, we can collect $n \geq p$ sample realizations (from observations or experiments),

$$\{(x_{i1}, \dots, x_{ip}, y_i)\}_{i=1}^n.$$

and minimize the empirical risk with respect to the (normalized) ℓ_2 loss function:

$$\hat{R}(h) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}))^2.$$

Collecting the data in matrices and vectors,

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \quad \boldsymbol{X} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}, \quad \boldsymbol{w} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix},$$

we can write the empirical risk concisely as

$$\hat{R}(h) = \frac{1}{2n} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|^2.$$

Minimizing over $h \in \mathcal{H}$ means minimizing over vectors $\boldsymbol{w} \in \mathbb{R}^{p+1}$, and the best $\boldsymbol{w}$ is then the vector that solves the unconstrained optimization problem

$$\underset{\boldsymbol{w} \in \mathbb{R}^{p+1}}{\text{minimize}} \, \frac{1}{2n} \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2.$$

This is an example of an optimization problem with variables $\boldsymbol{w}$, no constraints (*all* $\boldsymbol{w}$ are valid candidates and the constraint set is $\Omega = \mathbb{R}^{p+1}$), and a *quadratic* objective

$$\begin{aligned} f(\boldsymbol{w}) = \frac{1}{2n}\|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 &= \frac{1}{2n}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^\intercal(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) \\ &= \frac{1}{2n}\boldsymbol{w}^\intercal\boldsymbol{X}^\intercal\boldsymbol{X}\boldsymbol{w} - 2\boldsymbol{y}^\intercal\boldsymbol{X}\boldsymbol{w} + \boldsymbol{y}^\intercal\boldsymbol{y}, \end{aligned} \tag{9.4}$$

where $\boldsymbol{X}^\intercal$ is the matrix transpose. If the columns of $\boldsymbol{X}$ are linearly independent (which, by the way, requires there to be more data than the number of parameters p), this simple optimization problem has a unique closed form solution,

$$\boldsymbol{w}^* = (\boldsymbol{X}^\intercal\boldsymbol{X})^{-1}\boldsymbol{X}^\intercal\boldsymbol{y}. \tag{9.5}$$

In practice one would not compute $\boldsymbol{w}^*$ by evaluating (9.5). There are more efficient methods available, such as gradient descent, the conjugate gradient method, and variations of these. It is important to note that even in this simple example, solving the optimization problem can be difficult if the number of samples is large.

To illustrate the least squares setting using a concrete example, assume that we have data relating the basal metabolic rate (energy expenditure per time unit) in mammals to their mass.[1] Using data for

573 mammals from the PanTHERIA database[2], we see that the relationship is approximately linear (see Figure 9.1), and can be modelled as

$$Y = \beta_0 + \beta_1 X + \epsilon.$$

From the data we can assemble the vector $\boldsymbol{y}$ and the matrix $\boldsymbol{X} \in \mathbb{R}^{n\times(p+1)}$ in order to compute the $\boldsymbol{w} = (\beta_0, \beta_1)^\intercal$ (here, $p = 1$ and $n = 573$). We can find β_0 and β_1 by solving an optimization problem as described above (or solving the problem in closed form). Solving the problem, we get the values $\beta_0 = 1.362$ and $\beta_1 = 0.702$, and we can plot the line and see how it fits the data.

Example 9.5. (Image inpainting) Even problems in image processing that do not appear to be machine learning problems can be cast as such. An image can be viewed as an $m \times n$ matrix $\boldsymbol{U}$, with each entry u_{ij} corresponding to a light intensity (for greyscale images), or a colour vector, represented by a triple of red, green and blue intensities (usually with values between 0 and 255 each). For simplicity the following discussion assumes a greyscale image. For computational purposes, the matrix of an image is often viewed as an mn-dimensional vector $\boldsymbol{u}$, with the columns of the matrix stacked on top of each other.

[1] This example is from the episode "Size Matters" of the BBC series Wonders of Life.

[2] http://esapubs.org/archive/ecol/E090/184/#data

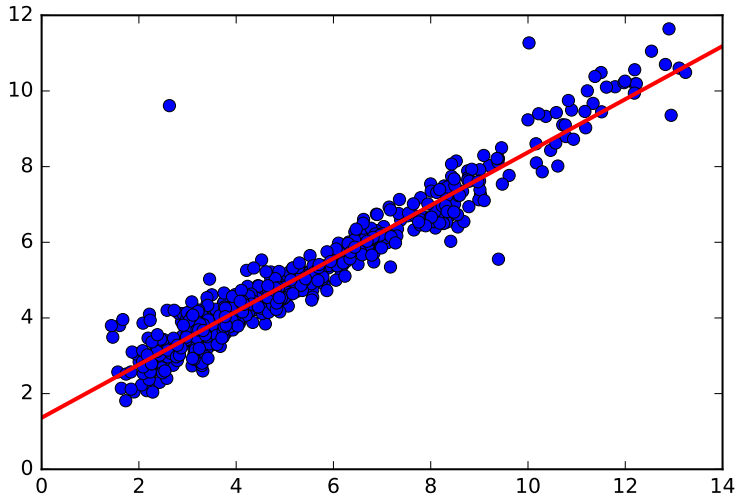


Figure 9.1: Fitting a regression line

In the *image inpainting* problem, one aims to *learn* the true value of missing or corrupted entries of an image. There are different approaches to this problem. A conceptually simple approach is to replace the image with the *closest* image among a set of images satisfying typical properties. But what are typical properties of a typical image? Some properties that come to mind are:

- Images tend to have large homogeneous areas in which the colour doesn't change much;
- Images have approximately low rank, when interpreted as matrices.

Total variation image analysis takes advantage of the first property. The **total variation** or TV-norm is the sum of the norm of the horizontal and vertical differences,

$$\|\boldsymbol{U}\|_{\mathrm{TV}} = \sum_{i=1}^{m}\sum_{j=1}^{n}\sqrt{(u_{i+1,j}-u_{i,j})^2+(u_{i,j+1}-u_{i,j})^2},$$

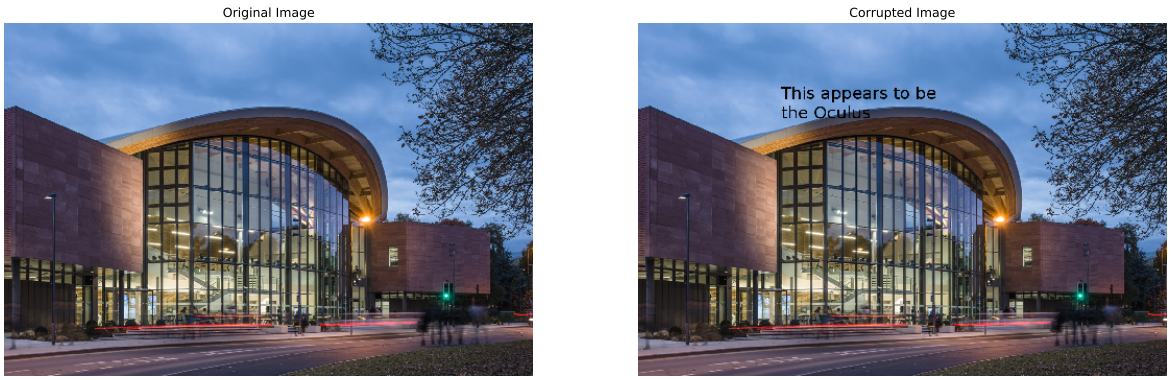
where we set entries with out-of-bounds indices to 0. The TV-norm naturally increases with increased variation or sharp edges in an image. Consider for example the two following matrices (imagine that they represent a 3×3 pixel block taken from an image).

$$\boldsymbol{U}_1 = \begin{pmatrix} 0 & 17 & 3 \\ 7 & 32 & 0 \\ 2 & 9 & 27 \end{pmatrix}, \quad \boldsymbol{U}_2 \begin{pmatrix} 1 & 1 & 3 \\ 1 & 0 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

The left matrix has TV-norm $\|\boldsymbol{U}_1\|_{\mathrm{TV}} = 200.637$, while the right one has TV-norm $\|\boldsymbol{U}_2\|_{\mathrm{TV}} = 14.721$ (verify this!) Intuitively, we would expect a natural image with artifacts added to it to have a higher TV norm.

Now let $\boldsymbol{U}$ be an image with entries u_{ij}, and let $\Omega \subset [m]\times[n] = \{(i,j) \mid 1 \le i \le m, 1 \le j \le n\}$ be the set of indices where the original image and the corrupted image coincide (all the other entries are missing). One could attempt to find the image with the *smallest* TV-norm that coincides with the known pixels u_{ij} for $(i,j) \in \Omega$. This is an optimization problem of the form

$$\text{minimize } \|\boldsymbol{W}\|_{\mathrm{TV}} \quad \text{subject to} \quad w_{ij} = u_{ij} \text{ for } (i,j) \in \Omega. \tag{9.6}$$



The TV-norm is an example of a convex function and the constraints are linear conditions which define a convex set. This is an example of a **convex optimization problem** and can be solved efficiently by a range of algorithms.

In our first example, we consider an image that has been corrupted. We first determine which entries of the corrupted image are known (the complement of the corrupted pixels), and these will specify the constraints in (9.6). As the problem is rather large (more than a million variables), it is important to choose a good solver that will solve the problem to sufficient accuracy in an acceptable amount of time.

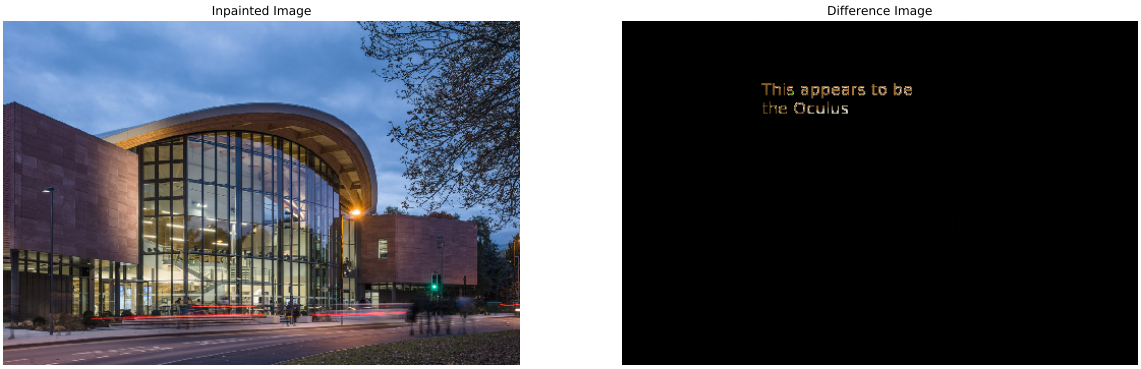


Figure 9.2: Removing writing from the Oculus

Another typical structure of images is that the **singular values** of the image, considered as matrix, decay quickly. The **singular value decomposition** (SVD) of a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is the matrix product

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T,$$

where $\boldsymbol{U} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{V} \in \mathbb{R}^{n \times n}$ are orthogonal matrices, and $\boldsymbol{\Sigma} \in \mathbb{R}^{m \times n}$ is a diagonal matrix with entries $\sigma_1, \ldots, \sigma_{\min\{m,n\}}$ on the diagonal. Instead of minimizing the TV-norm of an image $\boldsymbol{X}$, one may instead try to minimize the **Schatten 1-norm**, defined as the sum of the singular values, $\|\boldsymbol{U}\|_{S_1} = \sigma_1 + \cdots + \sigma_{\min\{m,n\}}$. The problem is then

$$\text{minimize } \|\boldsymbol{W}\|_{S_1} \quad \text{subject to} \quad w_{ij} = u_{ij} \text{ for } (i,j) \in \Omega.$$

This is an instance of a type of convex optimization problem known as **semidefinite programming**. Alternatively, one may also use the 1-norm of the image applied to a discrete cosine transform (DCT) or a

discrete wavelet transform (DWT). As this examples (and many more to come) shows: there is no unique choice of loss function, and hence of the objective function, for a particular problem. These choices depend on model assumptions and require some knowledge of the problem one is trying to solve.

We conclude by using total variation inpainting to liberate a parrot.

Caged parrot

Free parrot

Notes

In the past, optimization theory has largely been confined to applications from physics, engineering, and in operations research, economics and finance. In these settings, the objective function has a clear physical interpretation: either as some form of energy or cost to be minimized, or as a utility to be maximized. Optimization problems have featured implicitly in statistics, either as a measure of data misfit (as in linear regression) or in parameter estimation (as in maximum likelihood estimation). Despite this, the fields of statistics and of optimization theory have only recently started to come together through machine learning. A good overview of classical optimization theory is [60]. A reference for applications in imaging, such as those mentioned in this chapter, is [52].

10

Convexity

Convexity is a central theme in optimization. The reason is that convex optimization problems have many favourable properties, such the existence of unique global minima. In addition, convex optimization problems can be solved efficiently, leading to the often uttered statement that a problem can be considered solved if it can be cast as a convex optimization problem. Despite being a seemingly special property, convex optimization problems arise surprisingly often, and many difficult optimization problems can be approximated by convex problems.

Convex functions

Definition 10.1. A set $C \subseteq \mathbb{R}^d$ is **convex** if for all $\boldsymbol{v}, \boldsymbol{w} \in C$ and $\lambda \in [0, 1]$, the line $\lambda\boldsymbol{v} + (1 - \lambda)\boldsymbol{w} \in C$. A **convex body** is a convex set that is closed and bounded.

Definition 10.2. Let $S \subseteq \mathbb{R}^d$. A function $f \colon S \to \mathbb{R}$ is called *convex* if S is convex and for all $\boldsymbol{v}, \boldsymbol{w} \in S$ and $\lambda \in [0, 1]$,

$$f(\lambda\boldsymbol{v} + (1 - \lambda)\boldsymbol{w}) \leq \lambda f(\boldsymbol{v}) + (1 - \lambda)f(\boldsymbol{w}).$$

The function f is called *strictly convex* if

$$f(\lambda\boldsymbol{v} + (1 - \lambda)\boldsymbol{w}) < \lambda f(\boldsymbol{v}) + (1 - \lambda)f(\boldsymbol{w}).$$

A function f is called *concave*, if $-f$ is convex.

Figure 10.1 illustrates what a convex function of one variable looks like. The graph of the function lies below any line connecting two points on it. A function f has a **domain** $\mathrm{dom}(f)$, which is where the function is defined. For example, if $f(x) = \log(x)$, then $\mathrm{dom}(f) = \mathbb{R}_+$, the positive integers. The definition of a convex function thus states that the domain of f is a convex set S. We can also *restrict* a function on a smaller domain, even though the function could be defined more generally. For example, $f(x) = x^3$ is a convex function if restricted to the domain $\mathbb{R}_{\geq 0}$, but is not convex on $\mathbb{R}$.

A **convex optimization** problem is an optimization problem in which the set of constraints Ω and the function f are convex. While most general optimization problems are practically intractable, convex optimization problems can be solved efficiently, and still cover a surprisingly large range of applications!

Convex function have pleasant properties, while at the same time covering many of the functions that arise in applications. Perhaps the most important property is that local minima are global minima.

Theorem 10.3. *Let $f \colon \mathbb{R}^d \to \mathbb{R}$ be a convex function. Then any local minimizer of f is a global minimizer.*

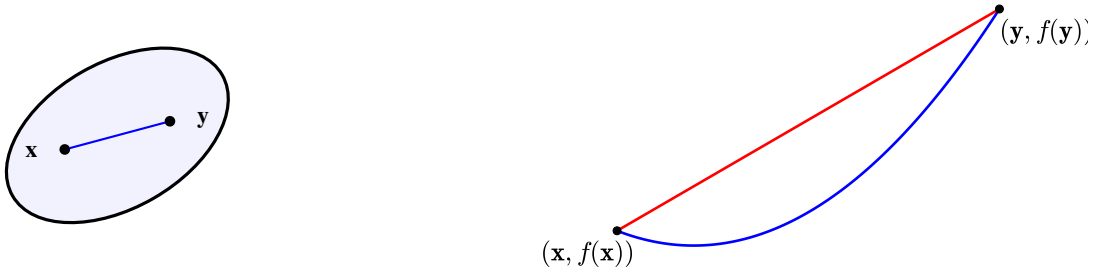


Figure 10.1: A convex set and a convex function

Proof. Let $\boldsymbol{v}^*$ be a local minimizer and assume that it is not a global minimizer. Then there exists a vector $\boldsymbol{w} \in \mathbb{R}^d$ such that $f(\boldsymbol{w}) < f(\boldsymbol{v}^*)$. Since f is convex, for any $\lambda \in [0,1]$ and $\boldsymbol{v} = \lambda \boldsymbol{w} + (1-\lambda)\boldsymbol{v}^*$ we have

$$f(\boldsymbol{v}) \leq \lambda f(\boldsymbol{w}) + (1-\lambda) f(\boldsymbol{v}^*) < \lambda f(\boldsymbol{v}^*) + (1-\lambda) f(\boldsymbol{v}^*) = f(\boldsymbol{v}^*).$$

This holds for all $\boldsymbol{v}$ on the line segment connecting $\boldsymbol{w}$ and $\boldsymbol{v}^*$. Since every open neighbourhood U of $\boldsymbol{v}^*$ contains a bit of this line segment, this means that every open neighbourhood U of $\boldsymbol{v}^*$ contains an $\boldsymbol{v} \neq \boldsymbol{v}^*$ such that $f(\boldsymbol{v}) \leq f(\boldsymbol{v}^*)$, in contradiction to the assumption that $\boldsymbol{v}^*$ is a local minimizer. It follows that $\boldsymbol{v}^*$ has to be a global minimizer. □

Remark 10.4. Note that in the above theorem we made no assumptions about the differentiability of the function f! In fact, while a convex function is always *continuous* on the interior of its domain, it need not be differentiable. The function $f(x) = |x|$ is a typical example: it is convex, but not differentiable at $x = 0$.

Example 10.5. Affine functions $f(\boldsymbol{v}) = \langle \boldsymbol{v}, \boldsymbol{a} \rangle + b$ and the exponential function e^x are examples of convex functions.

Example 10.6. In optimization we will often work with functions of matrices, where an $m \times n$ matrix is considered as a vector in $\mathbb{R}^{m \times n} \cong \mathbb{R}^{mn}$. If the matrix is symmetric, that is, if $\boldsymbol{A}^\intercal = \boldsymbol{A}$, then we only care about the upper diagonal entries, and we consider the space $\mathcal{S}^n$ of symmetric matrices as a vector space of dimension $d = n(n+1)/2$ (the number of entries on and above the main diagonal). Important functions on symmetric matrices that are convex are the operator norm $\|\boldsymbol{A}\|_2$, defined as

$$\|\boldsymbol{A}\|_2 := \max_{\boldsymbol{v}\colon \|\boldsymbol{v}\|=1} \frac{\|\boldsymbol{A}\boldsymbol{v}\|_2}{\|\boldsymbol{v}\|_2},$$

or the function $\log\det(\boldsymbol{X})$, defined on the set of *positive semidefinite* symmetric matrices $\mathcal{S}^d_+$.

There are useful ways of characterising convexity using differentiability.

Theorem 10.7. *1. Let $f \in C^1(\mathbb{R}^d)$. Then f is convex if and only if for all $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^d$,*

$$f(\boldsymbol{w}) \geq f(\boldsymbol{v}) + \nabla f(\boldsymbol{v})^\intercal (\boldsymbol{w} - \boldsymbol{v}).$$

2. Let $f \in C^2(\mathbb{R}^d)$. Then f is convex if and only if $\nabla^2 f(\boldsymbol{v})$ is positive semidefinite for all $\boldsymbol{v}$. If $\nabla^2 f(\boldsymbol{v})$ is positive definite for all $\boldsymbol{v}$, then f is strictly convex.

Example 10.8. Consider a quadratic function of the form

$$f(\boldsymbol{v}) = \frac{1}{2}\boldsymbol{v}^\intercal \boldsymbol{A}\boldsymbol{v} + \boldsymbol{b}^\intercal \boldsymbol{v} + c,$$

where $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is symmetric. Writing out the product, we get

$$\begin{aligned}\boldsymbol{v}^T\boldsymbol{A}\boldsymbol{v} &= \begin{pmatrix} v_1 & \cdots & v_n \end{pmatrix} \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} \\ &= \begin{pmatrix} v_1 & \cdots & v_n \end{pmatrix} \begin{pmatrix} a_{11}v_1 + \cdots + a_{1n}v_n \\ \vdots \\ a_{n1}v_1 + \cdots + a_{nn}v_n \end{pmatrix} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}v_iv_j.\end{aligned}$$

Because $\boldsymbol{A}$ is symmetric, we have $a_{ij} = a_{ji}$, and the above product simplifies to

$$\boldsymbol{v}^T\boldsymbol{A}\boldsymbol{v} = \sum_{i=1}^{n} a_{ii}v_i^2 + 2\sum_{1\le i<j\le n} a_{ij}v_iv_j.$$

This is a quadratic function, because it involves products of the v_i. The gradient and the Hessian of $f(\boldsymbol{v})$ are found by computing the partial derivatives of f:

$$\frac{\partial f}{\partial v_i} = \sum_{j=1}^{n} a_{ij}v_j + b_i, \quad \frac{\partial^2 f}{\partial v_i \partial v_j} = a_{ij}.$$

In summary, we have

$$\nabla f(\boldsymbol{v}) = \boldsymbol{A}\boldsymbol{v} + \boldsymbol{b}, \quad \nabla^2 f(\boldsymbol{v}) = \boldsymbol{A}.$$

Using the previous theorem, we see that f is convex **if and only if** $\boldsymbol{A}$ is positive semidefinite. A typical example for such a function is

$$f(\boldsymbol{v}) = \|\boldsymbol{A}\boldsymbol{v} - \boldsymbol{b}\|^2 = (\boldsymbol{A}\boldsymbol{v} - \boldsymbol{b})^T(\boldsymbol{A}\boldsymbol{v} - \boldsymbol{b}) = \boldsymbol{v}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{v} - 2\boldsymbol{b}^T\boldsymbol{A}\boldsymbol{v} + \boldsymbol{b}^T\boldsymbol{b}.$$

The matrix $\boldsymbol{A}^T\boldsymbol{A}$ is always symmetric and positive semidefinite (why?) so that the function f is convex.

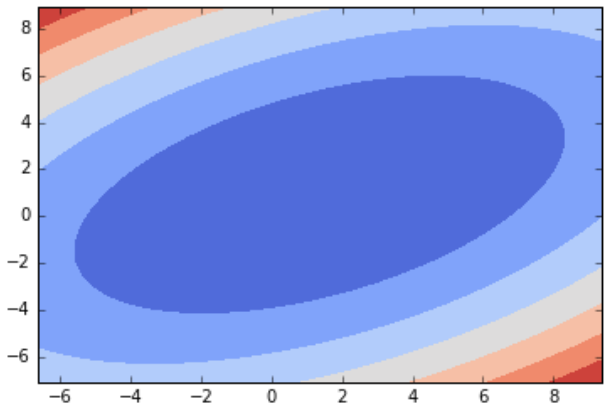
A convenient way to visualise a function $f\colon \mathbb{R}^2 \to \mathbb{R}$ is through **contour plots**. A **level set** of the function f is a set of the form

$$\{\boldsymbol{v} \mid f(\boldsymbol{v}) = c\},$$

where c is the **level**. Each such level set is a curve in $\mathbb{R}^2$, and a contour plot is a plot of a collection of such curves for various c. If one colours the areas between adjacent curves, one gets a plot as in the following figure. A *convex function* is has the property that there is only one *sink* in the contour plot.

Notes

Convexity plays a central role not only in optimization, but in many other branches of mathematics, including analysis, geometry and combinatorics. Convex sets, and in particular convex polyhedra, have been studied since antiquity. A good general introduction to convexity is [5]. A comprehensive treatment of convex optimization is [15].


8
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
8

11

Lagrangian Duality

In this chapter we study optimality conditions for convex problems of the form

$$\begin{aligned} \underset{\boldsymbol{w}}{\text{minimize}} \quad & f(\boldsymbol{w}) \\ \text{subject to} \quad & \boldsymbol{f}(\boldsymbol{w}) \leq \boldsymbol{0} \\ & \boldsymbol{h}(\boldsymbol{w}) = \boldsymbol{0}, \end{aligned} \tag{1}$$

where $\boldsymbol{w} \in \mathbb{R}^n$, $\boldsymbol{f} = (f_1, \ldots, f_m)^\intercal$, $\boldsymbol{h} = (h_1, \ldots, h_p)^\intercal$, and the inequalities are componentwise. We assume that f and the f_i are convex, and the h_j are linear. It is also customary to write the conditions $\boldsymbol{h}(\boldsymbol{w}) = \boldsymbol{0}$ as $\boldsymbol{A}\boldsymbol{w} = \boldsymbol{b}$, with $h_j(\boldsymbol{w}) = \boldsymbol{a}_j^\intercal \boldsymbol{w} - b_j$, $\boldsymbol{a}_j^\intercal$ being the j-th row of $\boldsymbol{A}$. The **feasible set** of (P) is the set

$$\mathcal{F} = \{\boldsymbol{w} \mid f_i(\boldsymbol{w}) \leq 0 \ \text{ for } i \in [m], \ \boldsymbol{A}\boldsymbol{w} = \boldsymbol{b}\}$$

It is easy to see that $\mathcal{F}$ is convex if the f_i are convex. If the objective f and the f_i are also linear, then (P) is called a **linear programming** problem, and $\mathcal{F}$ is a **polyhedron**. Such problems have been studied extensively and can be solved with efficient algorithms such as the simplex method.

A first-order optimality condition

We first generalize the standard first-order optimality conditions for differentiable functions to the setting of constrained convex optimization.

Theorem 11.1. *Let $f \in C^1(\mathbb{R}^d)$ be a convex, differentiable function, and consider a convex optimization problem of the form* (P)*. Then $\boldsymbol{w}^*$ is a minimizer of the optimization problem*

$$\text{minimize } f(\boldsymbol{w}) \quad \textit{subject to} \quad \boldsymbol{w} \in \mathcal{F}$$

if and only if for all $\boldsymbol{w} \in \mathcal{F}$,

$$\nabla f(\boldsymbol{w}^*)^\intercal (\boldsymbol{w} - \boldsymbol{w}^*) \geq 0, \tag{11.1}$$

where $\mathcal{F}$ is the feasible set of the problem.

Proof. Suppose $\boldsymbol{w}^*$ is such that (11.1) holds. Then, since f is a convex function, for all $\boldsymbol{w} \in \mathcal{F}$ we have,

$$f(\boldsymbol{w}) \geq f(\boldsymbol{w}^*) + \langle \nabla f(\boldsymbol{w}^*), (\boldsymbol{w} - \boldsymbol{w}^*) \rangle \geq f(\boldsymbol{w}^*),$$

which shows that $\boldsymbol{w}^*$ is a minimizer in $\mathcal{F}$. To show the opposite direction, assume that $\boldsymbol{w}^*$ is a minimizer but that (11.1) does not hold. This means that there exists a $\boldsymbol{w} \in \mathcal{F}$ such that $\langle \nabla f(\boldsymbol{w}^*), (\boldsymbol{w} - \boldsymbol{w}^*) \rangle < 0$. Since both $\boldsymbol{w}^*$ and $\boldsymbol{w}$ are in $\mathcal{F}$ and $\mathcal{F}$ is convex, any point $\boldsymbol{v}(\lambda) = (1-\lambda)\boldsymbol{w}^* + \lambda \boldsymbol{w}$ with $\lambda \in [0,1]$ is also in $\mathcal{F}$. At $\lambda = 0$ we have

$$\frac{df}{d\lambda} f(\boldsymbol{v}(\lambda))|_{\lambda=0} = \langle \nabla f(\boldsymbol{w}^*), (\boldsymbol{w} - \boldsymbol{w}^*) \rangle < 0.$$

Since the derivative at $\lambda = 0$ is negative, the function $f(\boldsymbol{v}(\lambda))$ is decreasing at $\lambda = 0$, and therefore, for small $\lambda > 0$, $f(\boldsymbol{v}(\lambda)) < f(\boldsymbol{v}(0)) = f(\boldsymbol{w}^*)$, in contradiction to the assumption that $\boldsymbol{w}^*$ is a minimizer. □

Example 11.2. In the absence of constraints, $\mathcal{F} = \mathbb{R}^d$, and the statement says that

$$\forall \boldsymbol{w} \in \mathbb{R}^n \colon \langle \nabla f(\boldsymbol{w}^*), (\boldsymbol{w} - \boldsymbol{w}^*) \rangle \geq 0.$$

Given $\boldsymbol{w}$ such that $\langle \nabla f(\boldsymbol{w}^*), (\boldsymbol{w} - \boldsymbol{w}^*) \rangle \geq 0$, then replacing $\boldsymbol{w}$ by $2\boldsymbol{w}^* - \boldsymbol{w}$ we also have the converse inequality, and therefore the optimality condition is equivalent to saying that $\nabla f(\boldsymbol{w}^*) = \boldsymbol{0}$. We therefore recover the well-known first order optimality condition.

Geometrically, the first order optimality condition means that the set

$$\{\boldsymbol{w} \mid \langle \nabla f(\boldsymbol{w}^*), \boldsymbol{w} \rangle = \langle \nabla f(\boldsymbol{w}^*), \boldsymbol{w}^* \rangle\}$$

defines a supporting hyperplane to the set $\mathcal{F}$.

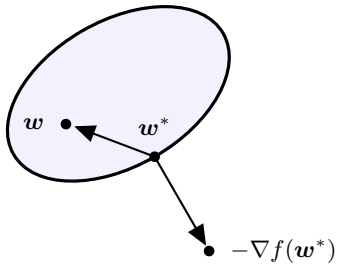


Figure 11.1: Optimality condition

Lagrangian duality

Recall the method of Lagrange multipliers. Given two functions $f(x,y)$ and $h(x,y)$, if the problem

$$\text{minimize } f(x,y) \qquad \text{subject to} \quad h(x,y) = 0$$

has a solution (x^*, y^*), then there exists a parameter λ, the *Lagrange multiplier*, such that

$$\nabla f(x^*, y^*) = \lambda \nabla h(x^*, y^*). \tag{11.2}$$

In other words, if we define the *Lagrangian* as

$$\mathcal{L}(x, y, \lambda) = f(x,y) - \lambda h(x,y),$$

then (11.2) says that $\nabla \mathcal{L}(x^*, y^*, \lambda) = 0$ for some λ. The intuition is as follows. The set

$$M = \{(x,y) \in \mathbb{R}^2 \mid h(x,y) = 0\}$$

is a curve in $\mathbb{R}^2$, and the gradient $\nabla h(x, y)$ is perpendicular to M at every point $(x, y) \in M$. For someone living inside M, a vector that is perpendicular to M is not visible, it is zero. Therefore the gradient $\nabla f(x, y)$ is zero as viewed from within M if it is perpendicular to M, or equivalently, a multiple of $\nabla h(x, y)$.

Alternatively, we can view the graph of $f(x, y)$ in three dimensions. A maximum or minimum of $f(x, y)$ along the curve defined by $h(x, y) = 0$ will be a point at which the direction of steepest ascent $\nabla f(x, y)$ is perpendicular to the curve $h(x, y) = 0$.

Example 11.3. Consider the function $f(x, y) = x^2 y$ with the constraint $h(x, y) = x^2 + y^2 - 3$ (a circle of radius $\sqrt{3}$). The Lagrangian is the function

$$\mathcal{L}(x, y, \lambda) = x^2 y - \lambda(x^2 + y^2 - 3).$$

Computing the partial derivatives gives the three equations

$$\begin{aligned}
\frac{\partial}{\partial x}\mathcal{L} &= 2xy - 2\lambda x = 0 \\
\frac{\partial}{\partial y}\mathcal{L} &= x^2 - 2\lambda y = 0 \\
\frac{\partial}{\partial \lambda}\mathcal{L} &= x^2 + y^2 - 3 = 0.
\end{aligned}$$

From the second equation we get $\lambda = \frac{x^2}{2y}$, and the first and third equations become

$$\begin{aligned}
2xy - \frac{x^3}{y} &= 0 \\
x^2 + y^2 - 3 &= 0.
\end{aligned}$$

Solving this system, we get six critical point $(\pm\sqrt{2}, \pm 1)$, $(0, \pm\sqrt{3})$. To find out which one of these is the minimizers, we just evaluate the function f on each of these.

We now turn to convex problems of the more general form form

$$\begin{aligned}
\underset{\boldsymbol{w}}{\text{minimize}} \quad & f(\boldsymbol{w}) \\
\text{subject to} \quad & \boldsymbol{f}(\boldsymbol{w}) \leq \mathbf{0} \\
& \boldsymbol{h}(\boldsymbol{w}) = \mathbf{0},
\end{aligned} \tag{11.3}$$

Denote by $\mathcal{D}$ the *domain* of all the functions f, f_i, h_j, i.e.,

$$\mathcal{D} = \operatorname{dom}(f) \cap \operatorname{dom}(f_1) \cap \cdots \cap \operatorname{dom}(f_m) \cap \operatorname{dom}(h_1) \cap \cdots \cap \operatorname{dom}(h_p).$$

Assume that $\mathcal{D}$ is not empty and let p^* be the optimal value of (11.3).

The **Lagrangian** of the system is defined as

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{w}) + \boldsymbol{\lambda}^\intercal \boldsymbol{f}(\boldsymbol{w}) + \boldsymbol{\mu}^\intercal h(\boldsymbol{w}) = f(\boldsymbol{w}) + \sum_{i=1}^{m} \lambda_i f_i(\boldsymbol{w}) + \sum_{i=1}^{p} \mu_i h_i(\boldsymbol{w}).$$

The vectors $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are called the **dual variables** or **Lagrange multipliers** of the system. The domain of $\mathcal{L}$ is $\mathcal{D} \times \mathbb{R}^m \times \mathbb{R}^p$.

Definition 11.4. The **Lagrange dual** of the problem (11.3) is the function

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\boldsymbol{w} \in \mathcal{D}} \mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}).$$

If the Lagrangian $\mathcal{L}$ is unbounded from below, then the value is $-\infty$.

The Lagrangian $\mathcal{L}$ is linear in the λ_i and μ_j variables. The infimum of a family of linear functions is concave, so that the Lagrange dual is a concave function. Therefore the negative $-g(\boldsymbol{\lambda}, \boldsymbol{\mu})$ is a convex function.

Lemma 11.5. *Let p^* be the optimal value of the problem 11.3. Then for any $\boldsymbol{\mu} \in \mathbb{R}^p$ and $\boldsymbol{\lambda} \geq \mathbf{0}$ we have*

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq p^*.$$

Proof. Let $\boldsymbol{w}^*$ be a feasible point for (11.3), that is,

$$f_i(\boldsymbol{w}^*) \leq 0, \quad h_j(\boldsymbol{w}^*) = 0,\ 1 \leq i \leq m,\ 1 \leq j \leq p.$$

Then for $\boldsymbol{\lambda} \geq \mathbf{0}$ and any $\boldsymbol{\mu}$, since each $h_j(\boldsymbol{w}^*) = 0$ and $\lambda_j f_j(\boldsymbol{w}^*) \leq 0$,

$$\mathcal{L}(\boldsymbol{w}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{w}^*) + \sum_{i=1}^{m} \lambda_i f_i(\boldsymbol{w}^*) + \sum_{j=1}^{p} \mu_j h_j(\boldsymbol{w}^*) \leq f(\boldsymbol{w}^*).$$

In particular,

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{\lambda}, \boldsymbol{\mu}) \leq f(\boldsymbol{w}^*).$$

Since this holds for *all* feasible $\boldsymbol{w}^*$, it holds in particular for the $\boldsymbol{w}^*$ that minimizes (11.3), for which $f(\boldsymbol{w}^*) = p^*$. □

The **Lagrange dual problem** of the optimization problem (11.3) is the problem

$$\text{maximize } g(\boldsymbol{\lambda}, \boldsymbol{\mu}) \qquad \text{subject to} \quad \boldsymbol{\lambda} \geq \mathbf{0}. \tag{11.4}$$

If d^* is the optimal value of (11.4), then $d^* \leq p^*$. In the special case of linear programming we actually have $d^* = p^*$. We will see that under certain conditions, we have $d^* = p^*$ for more general problems, but this is not always the case.

Notes

Duality is a central theme in optimization theory, and appears in many guises. The general idea is that an optimization problem can appear in two equivalent forms: a primal and a dual problem, where the solution of the dual problem provides a lower bound on the solution of the primal problem. The task is then to determine conditions under which this *duality gap* is zero. One of the features of duality is that the two complexity parameters of an optimization problem, namely the number of constraints and the number of variables, are exchanged under duality. This aspect can be used to simplify certain optimization problems. Another important aspect of duality is that one can often reformulate the requirement that certain primal and dual optimality conditions are satisfied as a system of equations. Duality in the setting of linear programming was first conjectured by John von Neumann, and subsequently developed by Tucker, Danzig, and others [24]. A treatment of convex optimization with a principled development of duality can be found in [10].

12

Karush-Kuhn-Tucker Conditions

For convex problems of the form

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{w}) \\ \text{subject to} & \boldsymbol{f}(\boldsymbol{w}) \leq \boldsymbol{0} \\ & \boldsymbol{A}\boldsymbol{w} = \boldsymbol{b}, \end{array} \tag{P}$$

we introduced the Lagrangian $\mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ and defined the Lagrange dual as

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\boldsymbol{w} \in \mathcal{D}} \mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}).$$

We saw that $g(\boldsymbol{\lambda}, \boldsymbol{\mu})$ is a lower bound on the optimal value of (P). Note that we wrote the linear equality conditions $h_j(\boldsymbol{w}) = 0$ as system of linear equations $\boldsymbol{A}\boldsymbol{w} = \boldsymbol{b}$. We will derive conditions under which the lower bound provided by the Lagrange dual matches the upper bound, and derive a system of equations and inequalities that certify optimality, the Karush-Kuhn-Tucker (KKT) conditions. These conditions can be seen as generalizations of the first-order optimality conditions to the setting when equality and inequality constraints are present.

Constraint qualification

Let p^* and d^* denote the primal and dual optimal values, so that

$$d^* = \sup_{\boldsymbol{\lambda} \geq 0, \boldsymbol{\mu}} g(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \inf_{\boldsymbol{w} \in \mathcal{D}} \{f(\boldsymbol{w}) \mid f_i(\boldsymbol{w}) \leq 0,\ \boldsymbol{A}\boldsymbol{w} = \boldsymbol{b}\} = p^*.$$

Once certain conditions, called **constraint qualifications**, hold, we can ensure that **strong duality** holds, which means $d^* = p^*$. One particular such constraint qualification is Slater's Theorem.

Theorem 12.1. *(Slater conditions) Assume that the interior of the domain $\mathcal{D}$ of* (P) *is non-empty, that the problem* (P) *is convex, and that there exists a point $\boldsymbol{w} \in \mathcal{D}$ such that*

$$f_i(\boldsymbol{w}) < 0,\ 1 \leq i \leq m, \quad \boldsymbol{A}\boldsymbol{w} = \boldsymbol{b},\ 1 \leq j \leq p.$$

Assume that $\boldsymbol{A}$ has maximal rank. Then $d^* = p^*$.

The proof makes use of a fundamental result on convex sets, the **separating hyperplane theorem**. For an affine hyperplane $H = \{\boldsymbol{w} \colon \boldsymbol{a}^T\boldsymbol{w} + b = 0\}$, we denote by $H_+ = \{\boldsymbol{w} \colon \boldsymbol{a}^\intercal\boldsymbol{w} + b \geq 0\}$ one of the two closed halfspaces defined by the hyperplane, and by H_- the other.

Theorem 12.2 (Separating Hyperplane Theorem)**.** *Let C and D be disjoint, nonempty convex subsets of $\mathbb{R}^d$. Then there exists an affine hyperplane H such that $C \subset H_+$ and $D \subset H_-$.*

A hyperplane with the properties of Theorem 12.2 is called a **separating hyperplane**. The hyperplane separates the two sets strictly, if each of the sets is contained in the interior of the respective halfspaces. It can be shown that for *compact* convex sets, there exists a hyperplane separating them strictly.

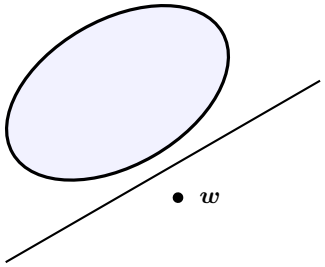


Figure 12.1: A hyperplane separating a convex sets and a point

Proof of Theorem 12.1. Assume $\boldsymbol{A}$ has rank p (the number of rows). Assume moreover that p^* is finite, since if $p^* = -\infty$, then by weak duality we already have $d^* = p^*$. Define the convex set

$$\mathcal{A} = \{(\boldsymbol{u}, \boldsymbol{v}, t) \in \mathbb{R}^m \times \mathbb{R}^p \times \mathbb{R} \mid \forall \boldsymbol{w} \in \mathcal{D}, f_i(\boldsymbol{w}) \leq u_i, \boldsymbol{A}\boldsymbol{w} - \boldsymbol{b} = \boldsymbol{v}, f(\boldsymbol{w}) \leq t\}.$$

Then the optimal value of (P) is

$$p^* = \inf\{t \mid (\boldsymbol{0}, \boldsymbol{0}, t) \in \mathcal{A}\}.$$

Define the convex set $\mathcal{B}$ as

$$\mathcal{B} = \{(\boldsymbol{0}, \boldsymbol{0}, s) \in \mathbb{R}^m \times \mathbb{R}^p \times \mathbb{R} \mid s < p^*\}.$$

The sets $\mathcal{A}$ and $\mathcal{B}$ are disjoint. To see this, assume to the contrary that there is a point $\boldsymbol{z} \in \mathcal{A} \cap \mathcal{B}$. Then since $\boldsymbol{z} \in \mathcal{B}$, $\boldsymbol{z} = (\boldsymbol{0}, \boldsymbol{0}, s)$ with $s < p^*$, but also since $\boldsymbol{z} \in \mathcal{A}$, ther exists an $\boldsymbol{w} \in \mathcal{D}$ with $f_i(\boldsymbol{w}) \leq 0$, $\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b} = \boldsymbol{0}$, and $f(\boldsymbol{w}) \leq s < p^*$, in contradiction to the optimality of p^*.

By the separating hyperplane theorem, there exists a hyperplane separating $\mathcal{A}$ and $\mathcal{B}$ (but not necessarily strictly!), defined by a vector $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\mu}}, \nu) \neq \boldsymbol{0}$ and $\alpha \neq 0$ with the property that

$$(\boldsymbol{u}, \boldsymbol{v}, t) \in \mathcal{A} \Longrightarrow \tilde{\boldsymbol{\lambda}}^\intercal \boldsymbol{u} + \tilde{\boldsymbol{\mu}}^\intercal \boldsymbol{v} + \nu t \geq \alpha \tag{12.1}$$

and

$$(\boldsymbol{u}, \boldsymbol{v}, t) \in \mathcal{B} \Longrightarrow \tilde{\boldsymbol{\lambda}}^\intercal \boldsymbol{u} + \tilde{\boldsymbol{\mu}}^\intercal \boldsymbol{v} + \nu t \leq \alpha. \tag{12.2}$$

If $\tilde{\boldsymbol{\lambda}} < \boldsymbol{0}$ or $\nu < 0$, we could find values of $\boldsymbol{u}$ and t for which $(\boldsymbol{u}, \boldsymbol{v}, t) \in \mathcal{A}$, but that make the right-hand side of (21.2) arbitrary small, contradicting the bound by α. It follows that $\tilde{\boldsymbol{\lambda}} \geq \boldsymbol{0}$ and $\nu \geq 0$. Condition (12.2) simply means that $\nu t \leq \alpha$ for all $t < p^*$, so that $\nu p^* \leq \alpha$. Combining this bound with (21.2), we get the two inequalities, valid for any $\boldsymbol{w} \in \mathcal{D}$,

$$\sum_{i=1}^{m} \tilde{\lambda}_i f_i(\boldsymbol{w}) + \tilde{\boldsymbol{\mu}}^\intercal (\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}) + \nu f(\boldsymbol{w}) \geq \alpha \geq \nu p^*. \tag{12.3}$$

Note that the left-hand side has the form of a Lagrangian function scaled by ν. If $\nu > 0$ we can divide by ν and set $\lambda_i = \tilde{\lambda}_i / \nu$, $\mu_i = \tilde{\mu}_i / \nu$, to obtain

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \geq p^*.$$

By weak duality we have $p^* \geq g(\boldsymbol{\lambda}, \boldsymbol{\mu})$, so that we get strong duality if $\nu > 0$. If $\nu = 0$, then (12.3) implies

$$\sum_{i=1}^{m} \tilde{\lambda}_i f_i(\boldsymbol{w}) + \tilde{\boldsymbol{\mu}}^\intercal (\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}) \geq 0, \tag{12.4}$$

and for a point $\tilde{\boldsymbol{w}}$ satisfying the conditions $\boldsymbol{A}\boldsymbol{w} = \boldsymbol{b}$ and $f_i(\boldsymbol{w}) < 0$ for $1 \leq i \leq m$, this means that $\tilde{\boldsymbol{\lambda}} = \boldsymbol{0}$. As $(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\mu}}, \nu) \neq \boldsymbol{0}$, we have $\boldsymbol{\mu} \neq \boldsymbol{0}$. Since $\tilde{\boldsymbol{\mu}}^\intercal (\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}) \geq 0$ and $= 0$ for some $\tilde{\boldsymbol{w}}$ in the interior of $\mathcal{D}$, there must be a $\boldsymbol{w}$ in the interior such that $\tilde{\boldsymbol{\mu}}^\intercal (\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}) < 0$, in contradiction to (12.4) (unless $\tilde{\boldsymbol{\mu}}^\intercal \boldsymbol{A} = \boldsymbol{0}$, which would contradict the condition that $\boldsymbol{A}$ has maximal rank p). □

Example 12.3. Consider a linear programming problem of the form

$$\begin{aligned} \text{minimize} \quad & \langle \boldsymbol{c}, \boldsymbol{w} \rangle \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{w} = \boldsymbol{b} \\ & \boldsymbol{w} \geq \boldsymbol{0}. \end{aligned}$$

The inequality constraints are $-w_i \leq 0$, while the equality constraints are $\boldsymbol{a}_i^\intercal \boldsymbol{w} - b_i$. The Lagrangian has the form

$$\begin{aligned} \mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{\mu}) &= \langle \boldsymbol{c}, \boldsymbol{w} \rangle - \sum_{i=1}^{m} \lambda_i w_i + \sum_{j=1}^{p} \mu_j (\boldsymbol{a}_j^\intercal \boldsymbol{w} - b_j) \\ &= (\boldsymbol{c} - \boldsymbol{\lambda} + \boldsymbol{A}^\intercal \boldsymbol{\mu})^\intercal \boldsymbol{w} - \boldsymbol{b}^\intercal \boldsymbol{\mu}. \end{aligned}$$

The infimum over $\boldsymbol{w}$ of this function is $-\infty$ unless $\boldsymbol{c} - \boldsymbol{\lambda} + \boldsymbol{A}^\intercal \boldsymbol{\mu} = \boldsymbol{0}$. The Lagrange dual is therefore

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \begin{cases} -\boldsymbol{\mu}^\intercal \boldsymbol{b} & \text{if } \boldsymbol{c} - \boldsymbol{\lambda} + \boldsymbol{A}^\intercal \boldsymbol{\mu} = \boldsymbol{0} \\ -\infty & \text{else.} \end{cases}$$

From the previous chapter we conclude that

$$\max\{-\boldsymbol{b}^\intercal \boldsymbol{\mu} \mid \boldsymbol{c} - \boldsymbol{\lambda} + \boldsymbol{A}^\intercal \boldsymbol{\mu} = \boldsymbol{0},\ \boldsymbol{\lambda} \geq \boldsymbol{0}\} \leq \min\{\boldsymbol{c}^\intercal \boldsymbol{w} \mid \boldsymbol{A}\boldsymbol{w} = \boldsymbol{b},\ \boldsymbol{w} \geq \boldsymbol{0}\}.$$

Note that if we write $\boldsymbol{v} = -\boldsymbol{\mu}$ and $\boldsymbol{s} = \boldsymbol{\lambda}$, then we get the dual version of the linear programming problem we started out with, and in this case it is known that

$$\max_{\boldsymbol{\lambda} \geq \boldsymbol{0}, \boldsymbol{\mu}} g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = p^*.$$

A more common way to express **linear programming duality** is as follows: the optimal value of

$$\text{maximize} \quad \langle \boldsymbol{b}, \boldsymbol{v} \rangle \quad \text{subject to} \quad \boldsymbol{A}^\intercal \boldsymbol{v} \leq \boldsymbol{c} \tag{D}$$

coincides with the optimal value of

$$\text{minimize} \quad \langle \boldsymbol{c}, \boldsymbol{w} \rangle \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{w} = \boldsymbol{b},\ \boldsymbol{w} \geq \boldsymbol{0}, \tag{P}$$

provided both (D) and (P) have a finite solution (here, we set $\boldsymbol{v} = -\boldsymbol{\mu}$).

Example 12.4. Consider the problem

$$\text{minimize } \|\boldsymbol{w}\|^2 \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{w} = \boldsymbol{b}.$$

Note that $\|\boldsymbol{w}\|^2 = \boldsymbol{w}^\mathsf{T}\boldsymbol{w}$. The Lagrangian is $\mathcal{L}(\boldsymbol{w}, \boldsymbol{\mu}) = \|\boldsymbol{w}\|^2 + \boldsymbol{\mu}^\mathsf{T}(\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b})$. For any $\boldsymbol{\mu}$, we can find the infimum

$$g(\boldsymbol{\mu}) = \inf_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, \boldsymbol{\mu})$$

by setting the derivative of the Lagrangian to $\boldsymbol{w}$ to zero:

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, \boldsymbol{\mu}) = 2\boldsymbol{w} + \boldsymbol{A}^\mathsf{T}\boldsymbol{\mu} = \boldsymbol{0},$$

which gives the solution

$$\boldsymbol{w} = -\frac{1}{2}\boldsymbol{A}^\mathsf{T}\boldsymbol{\mu}.$$

The dual function is therefore

$$g(\boldsymbol{\mu}) = -\frac{1}{4}\boldsymbol{\mu}^\mathsf{T}\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\boldsymbol{\mu} - \boldsymbol{b}^\mathsf{T}\boldsymbol{\mu}.$$

As the negative of a positive semidefinite quadratic function, it is concave. Moreover, we get the lower bound

$$-\frac{1}{4}\boldsymbol{\mu}^\mathsf{T}\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\boldsymbol{\mu} - \boldsymbol{b}^\mathsf{T}\boldsymbol{\mu} \leq \inf\{\|\boldsymbol{w}\|^2 \mid \boldsymbol{A}\boldsymbol{w} = \boldsymbol{b}\}.$$

The problem we started out with is convex, and if we assume that there exists a feasible primal point, then the above inequality is in fact an equality by Slater's conditions.

Karush-Kuhn-Tucker optimality conditions

Consider now a not necessarily convex problem of the form

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{w}) \\ \text{subject to} \quad & \boldsymbol{f}(\boldsymbol{w}) \leq \boldsymbol{0} \\ & \boldsymbol{h}(\boldsymbol{w}) = \boldsymbol{0}, \end{aligned} \tag{12.5}$$

If p^* is the optimal solution of (12.5) and $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ dual variables, then we have seen that (this holds even in the non-convex case)

$$p^* \geq g(\boldsymbol{\lambda}, \boldsymbol{\mu}).$$

From this is follows that for any primal feasible point $\boldsymbol{w}$,

$$f(\boldsymbol{w}) - p^* \leq f(\boldsymbol{w}) - g(\boldsymbol{\lambda}, \boldsymbol{\mu}).$$

The difference $f(\boldsymbol{w}) - g(\boldsymbol{\lambda}, \boldsymbol{\mu})$ between the primal objective function at a primal feasible point and the dual objective function at a dual feasible point is called the **duality gap** at $\boldsymbol{w}$ and $(\boldsymbol{\lambda}, \boldsymbol{\mu})$. For any such points we know that

$$p^*, q^* \in [g(\boldsymbol{\lambda}, \boldsymbol{\mu}), f(\boldsymbol{w})],$$

and if the gap is small we have a good approximation of the primal and dual optimal values. The duality gap can be used in iterative algorithms to define stopping criteria: if the algorithm generates a sequence of primal-dual variables $(\boldsymbol{w}^k, \boldsymbol{\lambda}^k, \boldsymbol{\mu}^k)$, then we can stop if the duality gap is less than, say, a predefined tolerance ε.

Now suppose that $(\boldsymbol{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ is such that the duality gap is zero. Then

$$\begin{aligned} f(\boldsymbol{w}^*) = g(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) &= \inf_{\boldsymbol{w}} \left(f(\boldsymbol{w}) + \sum_{i=1}^{m} \lambda_i^* f_i(\boldsymbol{w}) + \sum_{j=1}^{p} \mu_j^* h_j(\boldsymbol{w}) \right) \\ &\leq f(\boldsymbol{w}^*) + \sum_{i=1}^{m} \lambda_i^* f_i(\boldsymbol{w}^*) + \sum_{j=1}^{p} \mu_j^* h_j(\boldsymbol{w}^*) \\ &\leq f(\boldsymbol{w}^*), \end{aligned}$$

where the last inequality follows from the fact that $h_j(\boldsymbol{w}^*) = 0$ and $\lambda_i^* f_i(\boldsymbol{w}^*) \leq 0$ for $1 \leq j \leq p$ and $1 \leq i \leq m$. It follows that the inequalities are in fact equalities. From the identity

$$f(\boldsymbol{w}^*) = f(\boldsymbol{w}^*) + \sum_{i=1}^{m} \lambda_i^* f_i(\boldsymbol{w}^*)$$

and $\lambda_i^* \geq 0$ and $f_i(\boldsymbol{w}^*) \leq 0$ we also conclude that at such optimal points,

$$\lambda_i^* f_i(\boldsymbol{w}^*) = 0, \quad 1 \leq i \leq m.$$

This condition is known as **complementary slackness**. From the above we also see that $\boldsymbol{w}^*$ minimizes the Lagrangian $\mathcal{L}(\boldsymbol{w}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$, so that

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \mathbf{0}.$$

Collecting these conditions (primal and dual feasibility, complementary slackness, vanishing gradient), we arrive at a set of optimality conditions known as the Karush-Kuhn-Tucker (KKT) conditions.

Theorem 12.5. *(KKT conditions) Let $\boldsymbol{w}^*$ and $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ be primal and dual optimal solutions of* (12.5) *with zero duality gap. The the following conditions are satisfied:*

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{w}^*) &\leq \mathbf{0} \\ \boldsymbol{h}(\boldsymbol{w}^*) &= \mathbf{0} \\ \boldsymbol{\lambda}^* &\geq \mathbf{0} \\ \lambda_i^* f_i(\boldsymbol{w}^*) &= 0, \ 1 \leq i \leq m \\ \nabla_{\boldsymbol{w}} f(\boldsymbol{w}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla_{\boldsymbol{w}} f_i(\boldsymbol{w}^*) + \sum_{j=1}^{p} \mu_j^* \nabla_{\boldsymbol{w}} h_j(\boldsymbol{w}^*) &= \mathbf{0}. \end{aligned}$$

Moreover, if the problem is convex and the Slater Conditions (Theorem 12.1) are satisfied, then any points satisfying the KKT conditions have zero duality gap.

Notes

The Karush-Kuhn-Tucker conditions were introduced by Kuhn and Tucker [47], and the necessity was shown by William Karush in his 1939 MSc thesis at the University of Chicago. These conditions generalize and unify different previously known results: the optimality conditions for linear programming based on linear programming duality, and Lagrange duality in the presence of inequalities. The KKT conditions also form the basis of algorithms for non-linear optimization, such as interior-point methods. A detailed

treatment can be found in [60]. The separating hyperplane theorem is a central result in convexity, and lies at the heart of many duality results. In particular, it is the basis of a classic "theorem of the alternative" known as Farkas' Lemma, which states that given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times d}$ and $\boldsymbol{b} \in \mathbb{R}^m$, there exists a vector $\boldsymbol{w}$ such that

$$\boldsymbol{A}\boldsymbol{w} = \boldsymbol{b}, \quad \boldsymbol{w} \geq 0$$

if and only if there is no $\boldsymbol{v} \in \mathbb{R}^m$ such that

$$\boldsymbol{A}^\intercal \boldsymbol{v} \geq \boldsymbol{0}, \quad \langle \boldsymbol{v}, \boldsymbol{b} \rangle < 0.$$

This result, in turn, is an ingredient for deriving linear programming duality.

13

Support Vector Machines

In this lecture we return to the task of classification, and introduce a popular method for classification, **Support Vector Machines (SVMs)**, from the point of view of convex optimization.

Linear Support Vector Machines

In the simplest case there is a set of labels $\mathcal{Y} = \{-1, 1\}$ and the set of training points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^d$ is *linearly separable*: this means that there exists an affine hyperplane $h(\boldsymbol{x}) = \boldsymbol{w}^\intercal \boldsymbol{x} + b$ such that $h(\boldsymbol{x}_i) > 0$ if $y_i = 1$ and $h(\boldsymbol{x}_j) < 0$ if $y_j = -1$. We call the points for which $y_i = 1$ *positive*, and the ones for which $y_j = -1$ *negative*. The problem of finding such a hyperplane can be posed as a linear programming feasibility problem as follows: we look for a vector of *weights* $\boldsymbol{w}$ and a *bias term* b (together a $(p+1)$-dimensional vector) such that

$$\boldsymbol{w}^\intercal \boldsymbol{x}_i + b \geq 1, \text{ for } y_i = 1, \quad \boldsymbol{w}^\intercal \boldsymbol{x}_j + b \leq -1, \text{ for } y_j = -1.$$

Note that we can replace the $+1$ and -1 with any other positve or negative quantity by rescaling the $\boldsymbol{w}$ and $\boldsymbol{b}$, so this is just convention. We can also describe the two inequalities concisely as

$$y_i(\boldsymbol{w}^\intercal \boldsymbol{x}_i + b) - 1 \geq 0. \tag{13.1}$$

A hyperplane separating the two point sets will in general not be unique. As we want to use the linear classifier on new, yet unknown data, we want to find a separating hyperplane with best possible **margin**. Let δ_+ and δ_- denote the distance of a separating hyperplane to the closest positive and closest negative point, respectively. The quantity $\delta = \delta_+ + \delta_-$ is then called the margin of the classifier, and we want to find a hyperplane with largest possible margin.

We next show that the margin for a separating hyperplane that satisfies (13.1) is $\delta = 2/\|\boldsymbol{w}\|_2$. Given a hyperplane H described in (13.1) and a point $\boldsymbol{x}$ such that we have the equality $\boldsymbol{w}^\intercal \boldsymbol{x} + b = 1$ (the point is as close as possible to the hyperplane, also called a **support vector**), the distance of that point to the hyperplane can be computed by first taking the difference of $\boldsymbol{x}$ with a point $\boldsymbol{p}$ on H (an *anchor*), and then computing the dot product of $\boldsymbol{x} - \boldsymbol{p}$ with the unit vector $\boldsymbol{w}/\|\boldsymbol{w}\|$ normal to H.

As anchor point $\boldsymbol{p}$ we can just choose a multiple $c\boldsymbol{w}$ that is on the plane, i.e., that satisfies $\langle \boldsymbol{w}, c\boldsymbol{w} \rangle + b = 0$. This implies that $c = -b/\|\boldsymbol{w}\|^2$, and consequently $\boldsymbol{p} = -(b/\|\boldsymbol{w}\|^2)\boldsymbol{w}$. The distance is then

$$\begin{aligned} \delta_+ &= \langle \boldsymbol{x} + \frac{b}{\|\boldsymbol{w}\|^2}\boldsymbol{w}, \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \rangle = \frac{\langle \boldsymbol{x}, \boldsymbol{w} \rangle}{\|\boldsymbol{w}\|} + \frac{b}{\|\boldsymbol{w}\|^2} \langle \boldsymbol{w}, \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \rangle \\ &= \frac{1-b}{\|\boldsymbol{w}\|} + \frac{b}{\|\boldsymbol{w}\|} = \frac{1}{\|\boldsymbol{w}\|}. \end{aligned}$$

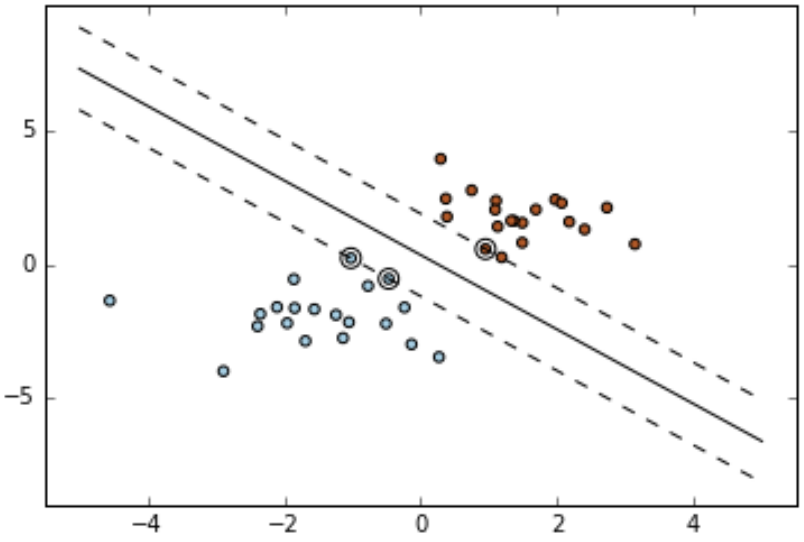


Figure 13.1: A hyperplane separating two sets of points with margin and support vectors.

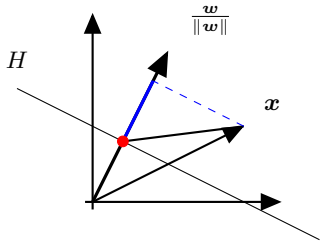


Figure 13.2: Computing the distance to the hyperplane

Similarly, we get $\delta_- = 1/\|\boldsymbol{w}\|$. The margin of this particular separating hyperplane is thus $\delta = 2/\|\boldsymbol{w}\|$. If we want to find a hyperplane with *largest* margin, we thus have to solve the quadratic optimization problem

$$\begin{aligned} \underset{\boldsymbol{w},b}{\text{minimize}} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 \\ \text{subject to} \quad & y_i(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_i + b) - 1 \geq 0, \quad 1 \leq i \leq n. \end{aligned}$$

Note that b is also an unknown variable in this problem! The factor $1/2$ in the objective function is just to make the gradient look nicer. The Lagrangian of this problem is

$$\begin{aligned} \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda}) &= \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{m} \lambda_i y_i \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_i - \lambda_i y_i b + \lambda_i \\ &= \frac{1}{2}\boldsymbol{w}^\mathsf{T}\boldsymbol{w} - \boldsymbol{\lambda}^\mathsf{T}\boldsymbol{X}\boldsymbol{w} - b\boldsymbol{\lambda}^\mathsf{T}\boldsymbol{y} + \sum_{i=1}^{m} \lambda_i, \end{aligned}$$

where we denote by $\boldsymbol{X}$ the matrix with the $y_i \boldsymbol{x}_i^\intercal$ as rows. We can then write the conditions on the gradient with respect to $\boldsymbol{w}$ and b of the Lagrangian as

$$\begin{aligned} \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda}) &= \boldsymbol{w} - \boldsymbol{X}^\intercal \boldsymbol{\lambda} = \boldsymbol{0} \\ \frac{\partial \mathcal{L}}{\partial b}(\boldsymbol{w}, b, \boldsymbol{\lambda}) &= \boldsymbol{y}^\intercal \boldsymbol{\lambda} = 0. \end{aligned} \tag{13.2}$$

If $\boldsymbol{y}^T \boldsymbol{\lambda} \neq 0$, then the conditions (13.2) cannot be satisfied and the problem is unbounded from below. If $\boldsymbol{y}^T \boldsymbol{\lambda} = 0$, then the first condition in (13.2) is necessary and sufficient for a minimizer. Replacing $\boldsymbol{w}$ by $\boldsymbol{X}^\intercal \boldsymbol{\lambda}$ and $\boldsymbol{\lambda}^\intercal \boldsymbol{y}$ by 0 in the Lagrangian function then gives the expression for the Lagrange dual $g(\boldsymbol{\lambda})$,

$$g(\boldsymbol{\lambda}) = \begin{cases} -\frac{1}{2} \boldsymbol{\lambda}^\intercal \boldsymbol{X} \boldsymbol{X}^\intercal \boldsymbol{\lambda} + \sum_{i=1}^m \lambda_i & \text{if } \boldsymbol{y}^T \boldsymbol{\lambda} = 0 \\ -\infty & \text{else.} \end{cases}$$

Finally, maximizing this function and changing the sign, so that the maximum becomes a minimum, we can formulate the Lagrange dual optimization problem as

$$\underset{\boldsymbol{\lambda}}{\text{minimize}} \quad \frac{1}{2} \boldsymbol{\lambda}^\intercal \boldsymbol{X} \boldsymbol{X}^\intercal \boldsymbol{\lambda} - \boldsymbol{\lambda}^\intercal \boldsymbol{e} \quad \text{subject to} \quad \boldsymbol{\lambda} \geq \boldsymbol{0}, \tag{13.3}$$

where $\boldsymbol{e}$ is the vector of all ones.

Assuming that the data is linearly separable, the primal and dual optimization problems for finding the parameters of a separating hyperplane of maximal margin can therefore be formulated as

$$\text{minimize} \, \frac{1}{2} \|\boldsymbol{w}\|^2 \quad \text{subject to} \quad \boldsymbol{e} - b\boldsymbol{y} - \boldsymbol{X}^T \boldsymbol{x} \leq \boldsymbol{0}, \tag{P}$$

where $\boldsymbol{e}$ is the vector of all ones. The Lagrange dual optimization problem is

$$\text{minimize} \, \frac{1}{2} \boldsymbol{\lambda}^\intercal \boldsymbol{X} \boldsymbol{X}^\intercal \boldsymbol{\lambda} - \boldsymbol{\lambda}^\intercal \boldsymbol{e} \quad \text{subject to} \quad \boldsymbol{\lambda} \geq \boldsymbol{0}. \tag{D}$$

Note that there is one dual variable λ_i per data point $\boldsymbol{x}_i$. We can find the optimal value by solving the dual problem (D), but that does not give us automatically the weights $\boldsymbol{w}$ and the bias b. We can find the weights by $\boldsymbol{w} = \boldsymbol{X}^\intercal \boldsymbol{\lambda}$. As for b, this is best determined from the KKT conditions of the problem. These can be written by combining the constraints of the primal problem with the conditions on the gradient of the Lagrangian, the condition $\boldsymbol{\lambda} \geq \boldsymbol{0}$, and complementary slackness as

$$\begin{aligned} \boldsymbol{X}\boldsymbol{w} + b\boldsymbol{y} - \boldsymbol{e} &\geq \boldsymbol{0} \\ \boldsymbol{\lambda} &\geq \boldsymbol{0} \\ \lambda_i (1 - y_i (\boldsymbol{w}^\intercal \boldsymbol{x}_i + b)) &= 0 \text{ for } 1 \leq i \leq n \\ \boldsymbol{w} - \boldsymbol{X}^\intercal \boldsymbol{\lambda} &= \boldsymbol{0} \\ \boldsymbol{y}^\intercal \boldsymbol{\lambda} &= 0. \end{aligned}$$

To get b, we can choose one of the equations in which $\lambda_i \neq 0$, and then find b by setting $b = y_i (1 - y_i \boldsymbol{w}^\intercal \boldsymbol{x}_i)$. With the KKT conditions written down, we can go about solving the problem of finding a maximum margin linear classifier using standard optimization methods.

Extensions

So far we looked at the particularly simple case where (a) the data falls into two classes, (b) the points can actually be well separated, and (c) they can be separated by an affine hyperplane. In reality, these three assumptions may not hold. We briefly discuss extensions of the basic model to account for the three situations just mentioned.

Non-exact separation

What happens when the data can not be separated by a hyperplane? In this case the constraints can not be satisfied: there is no feasible solution to the problem. We can still modify the problem to allow for *misclassification*: we want to find a hyperplane that separates the two point sets as good as possible, but we allow for some mistakes.

One approach is to add an additional set of n *slack variables* $s_1, \ldots, s_n$, and modify the constraints to

$$\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_i + b \geq 1 - s_i, \text{ for } y_i = 1, \quad \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_j + b \leq -1 + s_j, \text{ for } y_j = -1, \quad s_i \geq 0.$$

The i-th data point can land on the wrong side of the hyperplane if $s_i > 1$, and consequently the sum $\sum_{i=1}^n s_i$ is an upper bound on the number of errors possible. If we want to minimize the number of misclassified points, we may want to minimize this upper bound, so a sensible choice for objective function would be to add a multiple of this sum. The new problem thus becomes

$$\begin{aligned} \text{minimize} & \frac{1}{2}\|\boldsymbol{w}\|^2 + \mu \sum_{j=1}^{n} s_j \\ \text{subject to} \quad & y_i(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_i + b) - 1 + s_i \geq 0, \quad 1 \leq i \leq n \\ & s_i \geq 0, \quad 1 \leq i \leq n, \end{aligned}$$

for some parameter μ. The Lagrangian of this problem and the KKT conditions can be derived in a similar way as in the separable case and are left as an exercise.

Non-linear separation and kernels

The key to extending SVMs from linear to non-linear separation is the observation that the dual form of the optimization problem (D) depends only on the dot products $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ of the data points. In fact, the (i, j)-th entry of the matrix $\boldsymbol{X}\boldsymbol{X}^\mathsf{T}$ is precisely $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$!

If we map our data into a higher (possibly infinite) dimensional space $\mathcal{H}$,

$$\varphi \colon \mathbb{R}^d \to \mathcal{H},$$

and consider the data points $\varphi(\boldsymbol{x}_i)$, $1 \leq i \leq n$, then applying the support vector machine to these higher dimensional vectors will only depend on the dot products

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \varphi(\boldsymbol{x}_i), \varphi(\boldsymbol{x}_j) \rangle.$$

The function K is called a **kernel function**. A typical example, often used in practice, is the Gaussian radial basis function (RBF),

$$K(\boldsymbol{x}, \boldsymbol{y}) = e^{-\|\boldsymbol{x}-\boldsymbol{y}\|^2/2\sigma^2}.$$

Note that we *don't need to know how the function φ looks like*! In the equation for the hyperplane we simply replace $\boldsymbol{w}^\mathsf{T}\boldsymbol{x}$ with $K(\boldsymbol{w}, \boldsymbol{x})$. The only difference now is that the function ceases to be linear in $\boldsymbol{x}$: we get a non-linear decision boundary.

Multiple classes

One is often interested in classifying data into more than two classes. There are two simple ways in which support vector machines can be extended for such problems: one-vs-one and one-vs-rest. In the one-vs-one case, given k classes, we train one classifier for each pair of classes in the training data,

obtaining a total of $k(k-1)/2$ classifiers. When it comes to prediction, we apply each of the classifiers to our test data and choose the class that was chosen the most among all the classifiers. In the one-vs-rest approach, each train k binary classifiers: in each one, one class corresponds to a chosen class, and the second class corresponds to the rest. By associating confidence scores to each classifier, we choose the one with the highest confidence score.

Example 13.1. An example that uses all three extensions mentioned is handwritten digit recognition. Suppose we have a series of pixels, each representing a number, and associated labels $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. We would like to train a support vector machine to recognize new digits. Given the knowledge we have, we can implement this task using standard optimization software such as CVXPY. Luckily, there are packages that have this functionality already implemented, such as the SCIKIT-LEARN package for Python. We illustrate its functioning below. The code also illustrates some standard procedures when tackling a machine learning problem:

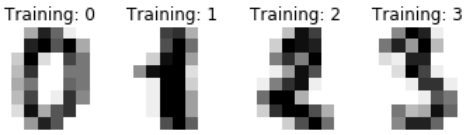
- **Separate** the data set randomly into *training data* and *test data*;
- **Create** a support vector classifier with optional parameters;
- **Train** (using FIT) the classifier with the training data;
- **Predict** the response using the test data and compare with the true response;
- **Report** the results.

An important aspect to keep in mind is that when testing the performance using the test data, we should compare the classification accuracy to a naive baseline: if, for example, 80% of the test data is classified as $+1$, then making a prediction of $+1$ for all the data will give us an accuracy of 80%; in this case, we would want our classifier to perform considerably better than getting the right answer 80% of the time!

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
% matplotlib inline
from sklearn import svm, datasets, metrics
from sklearn.model_selection import train_test_split
```

In [2]:
```
digits = datasets.load_digits()

# Display images and labels
images_and_labels = list(zip(digits.images, digits.target))
for index, (image, label) in enumerate(images_and_labels[:4]):
    plt.subplot(2, 4, index + 1)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Training: %i' % label)

# Turn images into 1-D arrays
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

# Create classifier
svc = svm.SVC(gamma=0.001)

# Randomly split data into train and test set
X_train, X_test, y_train, y_test = train_test_split(data,
 digits.target, test_size = 0.4, random_state=0)
svc.fit(X_train, y_train)
```

Out [2]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma=0.001,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```



Now apply prediction to test set and report performance.

In [3]:
```
predicted = svc.predict(X_test)
print("Classification report for classifier %s:\n%s\n"
      % (svc, metrics.classification_report(y_test, predicted)))
```

Out [3]:
```
Classification report for classifier SVC(C=1.0, cache_size=200,
 class_weight=None, coef0=0.0,vdecision_function_shape=None,
 degree=3, gamma=0.001, kernel='rbf', max_iter=-1, probability=False,
 random_state=None, shrinking=True, tol=0.001, verbose=False):
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        60
          1       0.97      1.00      0.99        73
          2       1.00      0.97      0.99        71
          3       1.00      1.00      1.00        70
          4       1.00      1.00      1.00        63
          5       1.00      0.98      0.99        89
          6       0.99      1.00      0.99        76
          7       0.98      1.00      0.99        65
          8       1.00      0.99      0.99        78
          9       0.99      1.00      0.99        74

avg / total       0.99      0.99      0.99       719
```

In [4]:
```
import skimage
from skimage import data
from skimage.transform import resize
from skimage import io
import os
```

Now try this out on some original data!

In [5]:
```
mydigit1 = io.imread('images/digit9.png')
mydigit2 = io.imread('images/digit4.png')
plt.figure(figsize=(8, 4))
plt.subplot(1,2,1)
plt.imshow(mydigit1, cmap=plt.cm.gray_r, interpolation='nearest')
plt.axis('off')
plt.subplot(1,2,2)
plt.imshow(mydigit2, cmap=plt.cm.gray_r, interpolation='nearest')
plt.axis('off')
plt.show()
```

In [6]:
```
smalldigit1 = resize(mydigit1, (8,8))
smalldigit2 = resize(mydigit2, (8,8))
mydigits = np.concatenate((np.round(15*(np.ones((8,8))-
                        smalldigit1[:,:,0])).reshape((64,1)).T,
            np.round(15*(np.ones((8,8))-
            smalldigit2[:,:,0])).reshape((64,1)).T),axis=0)
# After some preprocessing, make prediction
guess = svc.predict(mydigits)
print guess
```

```
[9 4]
```

Notes

The problem of linear separation is as old as machine learning itself. Even though the ideas underlying support vector machines go back to the 1960s, their systematic treatment in modern form is based on work by Vapnik and collaborators at AT& T in the 1990s [20]. Support vector machines and kernel methods are part of the main canon of machine learning and statistical learning, and are treated in most textbooks on the subject. Good comprehensive treatments dedicated to support vector machiens are [21, 71].

14

Iterative Algorithms

Most modern optimization methods are iterative: they generate a sequence of points $\boldsymbol{w}_0, \boldsymbol{w}_1, \ldots$ in $\mathbb{R}^d$ in the hope that this sequences converges to a local or global minimizer $\boldsymbol{w}^*$ of a function $f(\boldsymbol{w})$. A typical rule for generating such a sequence is to start with a vector $\boldsymbol{w}_0$, chosen by an educated guess, and then for $k \geq 0$, move from step k to $k+1$ by

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \alpha_k \boldsymbol{p}_k,$$

in a way that ensures that $f(\boldsymbol{w}_{k+1}) \leq f(\boldsymbol{w}_k)$. The parameter α_k is called the **step length** (or **learning rate** in machine learning), while $\boldsymbol{p}_k$ is the **search direction**. The are many ways in which the direction $\boldsymbol{p}_k$ and the step length α_k can be chosen. If we take

$$\boldsymbol{p}_k = -\nabla f(\boldsymbol{w}_k), \tag{14.1}$$

then we take a step in the direction of **steepest descent** and the resulting method is (unsurprisingly) called **gradient descent**. If there is second-order information available, then we can take steps of the form

$$\boldsymbol{p}_k = -\nabla^2 f(\boldsymbol{w}_k)^{-1} \nabla f(\boldsymbol{w}_k). \tag{14.2}$$

The resulting method is called **Newton's Method**. If applicable, Newton's method converges faster to a solution, but the computation at each step is more expensive.

Gradient descent

In the method of gradient descent, the search direction is chosen as

$$\boldsymbol{p}_k = -\nabla f(\boldsymbol{w}_k).$$

To see why this makes sense, let $\boldsymbol{p}$ be a direction and consider the Taylor expansion

$$f(\boldsymbol{w}_k + \alpha \boldsymbol{p}) = f(\boldsymbol{w}_k) + \alpha \boldsymbol{p}^\intercal \nabla f(\boldsymbol{w}_k) + O(\alpha^2).$$

Considering this as a function of α, the rate of change in direction $\boldsymbol{p}$ at $\boldsymbol{w}_k$ is the derivative of this function at $\alpha = 0$,

$$\frac{\mathrm{d} f(\boldsymbol{w}_k + \alpha \boldsymbol{p})}{\mathrm{d}\alpha}|_{\alpha=0} = \langle \boldsymbol{p}, \nabla f(\boldsymbol{w}_k) \rangle,$$

also known as the **directional derivative** of f at $\boldsymbol{w}_k$ in the direction $\boldsymbol{p}$. This formula indicates that the rate of change is *negative*, and we have a **descent direction**, if $\langle \boldsymbol{p}, \nabla f(\boldsymbol{w}_k) \rangle < 0$.

The Cauchy-Schwarz inequality gives the bounds

$$-\|\boldsymbol{p}\|_2\|\nabla f(\boldsymbol{w}_k)\|_2 \leq \langle \boldsymbol{p}, \nabla f(\boldsymbol{w}_k)\rangle \leq \|\boldsymbol{p}\|_2\|\nabla f(\boldsymbol{w}_k)\|_2.$$

We see that the rate of change is the smallest when the first inequality is an equality, which happens if

$$\boldsymbol{p} = -\alpha\nabla f(\boldsymbol{w}_k)$$

for some $\alpha > 0$.

For a visual interpretation of what it means to be a descent direction, note that the **angle** θ between a vector $\boldsymbol{p}$ and the gradient $\nabla f(\boldsymbol{w})$ at a point $\boldsymbol{w}$ is given by (see Preliminaries, Page 9)

$$\langle \boldsymbol{p}, \nabla f(\boldsymbol{w})\rangle = \|\boldsymbol{p}\|_2\|\nabla f(\boldsymbol{w})\|_2 \cos(\theta).$$

This is negative if the vector $\boldsymbol{p}$ forms and angle greater than $\pi/2$ with the gradient. Recall that the gradient points in the direction of steepest ascent, and is orthogonal to the *level sets*. If you are standing on the slope of a mountain, walking along the level set lines will not change your elevation, the gradient points to the steepest upward direction, and the negative gradient to the steepest descent.

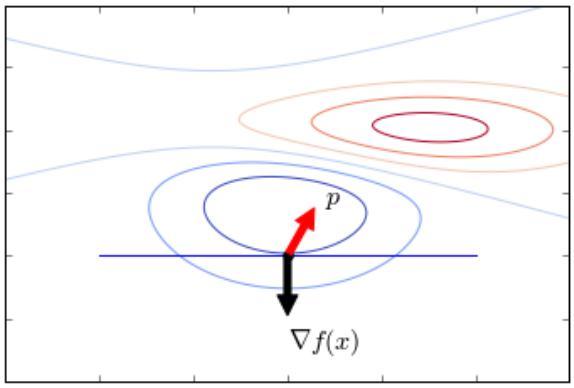


Figure 14.1: A descent direction

Any multiple $\alpha\nabla f(\boldsymbol{w}_k)$ points in the direction of steepest descent, but we have to choose a sensible parameter α to ensure that we make sufficient progress, but at the same time don't overshoot. Ideally, we would choose the value α_k that minimizes $f(\boldsymbol{w}_k - \alpha_k\nabla f(\boldsymbol{w}_k))$. While finding such a minimizer is in general not easy (see Section Lecture 4 for alternatives), for quadratic functions in can be given in closed form.

Linear least squares

Consider a function of the form

$$f(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}\|_2^2.$$

The Hessian is symmetric and positive semidefinite, with the gradient given by

$$\nabla f(\boldsymbol{w}) = \boldsymbol{A}^\intercal(\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}).$$

The method of gradient descent proceeds as

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \boldsymbol{A}^\top (\boldsymbol{A}\boldsymbol{w}_k - \boldsymbol{b}).$$

To find the best α_k, we compute the minimum of the function

$$\alpha \mapsto \varphi(\alpha) = f(\boldsymbol{w}_k - \alpha \boldsymbol{A}^\intercal (\boldsymbol{A}\boldsymbol{w}_k - \boldsymbol{b})). \tag{14.3}$$

If we set $\boldsymbol{r}_k := \boldsymbol{A}^\intercal(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{w}_k) = -\nabla f(\boldsymbol{w}_k)$ and compute the minimum of (14.3) by setting the derivative to zero,

$$\begin{aligned}
\varphi'(\alpha) &= \frac{d}{d\alpha} f(\boldsymbol{w}_k + \alpha \boldsymbol{r}_k) = \langle \nabla f(\boldsymbol{w}_k + \alpha \boldsymbol{r}_k), \boldsymbol{r}_k \rangle \\
&= \langle \boldsymbol{A}^\intercal (\boldsymbol{A}(\boldsymbol{w}_k + \alpha \boldsymbol{r}_k) - \boldsymbol{b}), \boldsymbol{r}_k \rangle \\
&= \langle \boldsymbol{A}^\intercal (\boldsymbol{A}\boldsymbol{w}_k - \boldsymbol{b}), \boldsymbol{r}_k \rangle + \alpha^2 \langle \boldsymbol{A}^\intercal \boldsymbol{A}\boldsymbol{r}_k, \boldsymbol{r}_k \rangle \\
&= -\boldsymbol{r}_k^\intercal \boldsymbol{r}_k + \alpha \boldsymbol{r}_k^\intercal \boldsymbol{A}^\intercal \boldsymbol{A}\boldsymbol{r}_k = 0,
\end{aligned}$$

we get the step length

$$\alpha_k = \frac{\boldsymbol{r}_k^\intercal \boldsymbol{r}_k}{\boldsymbol{r}_k^\intercal \boldsymbol{A}^\intercal \boldsymbol{A}\boldsymbol{r}_k} = \frac{\|\boldsymbol{r}_k\|_2^2}{\|\boldsymbol{A}\boldsymbol{r}_k\|_2^2}.$$

Note also that when we have $\boldsymbol{r}_k$ and α_k, we can compute the next $\boldsymbol{r}_k$ as

$$\begin{aligned}
r_{k+1} &= \boldsymbol{A}^\intercal (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{w}_{k+1}) \\
&= \boldsymbol{A}^\intercal (\boldsymbol{b} - \boldsymbol{A}(\boldsymbol{w}_k + \alpha_k \boldsymbol{r}_k)) \\
&= \boldsymbol{A}^\intercal (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{w}_k - \alpha_k \boldsymbol{A}\boldsymbol{r}_k) = \boldsymbol{r}_k - \alpha_k \boldsymbol{A}^\intercal \boldsymbol{A}\boldsymbol{r}_k.
\end{aligned}$$

The gradient descent algorithm for the linear least squares problem proceeds by first computing $\boldsymbol{r}_0 = \boldsymbol{A}^\intercal(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{w}_0)$, and then at each step

$$\begin{aligned}
\alpha_k &= \frac{\boldsymbol{r}_k^\intercal \boldsymbol{r}_k}{\boldsymbol{r}_k^\intercal \boldsymbol{A}^\intercal \boldsymbol{A}\boldsymbol{r}_k} \\
\boldsymbol{w}_{k+1} &= \boldsymbol{w}_k + \alpha_k \boldsymbol{r}_k \\
\boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha_k \boldsymbol{A}^\intercal \boldsymbol{A}\boldsymbol{r}_k.
\end{aligned}$$

Does this work? How do we know when to stop? It is worth noting that the residual satisfies $\boldsymbol{r} = 0$ if and only if $\boldsymbol{w}$ is a stationary point, in our case, a minimizer. One criteria for stopping could then be to check whether $\|\boldsymbol{r}_k\|_2 \leq \varepsilon$ for some given tolerance $\varepsilon > 0$. One potential problem with this criterion is that the function can become *flat* long before reaching a minimum, so an alternative stopping method would be to stop when the difference between two successive points, $\|\boldsymbol{w}_{k+1} - \boldsymbol{w}_k\|_2$, becomes smaller than some $\varepsilon > 0$.

Example 14.1. We plot the trajectory of gradient descent with the data

$$\boldsymbol{A} = \begin{pmatrix} 2 & 0 \\ 1 & 3 \\ 0 & 1 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}.$$

As can be seen from the plot, we always move in the direction orthogonal to a level set, and stop at a point where we are tangent to a level set.

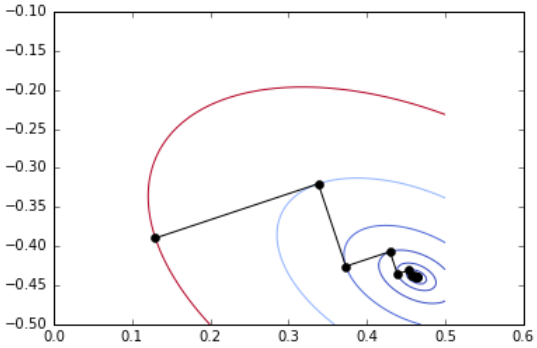


Figure 14.2: Trajectory of gradient descent

Step length selection

While for a quadratic function of the form $\|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}\|^2$ it was possible to find a step length α_k by minimizing the function in the direction of steepest descent, in general this may not be possible or even desirable. The step length is often called the **learning rate** in machine learning. A good step length

- is not too small (so that the algorithm does not take too long);
- is not too large (we might end up at a point with larger function value);
- is easy to compute.

There are conditions (such as the Armijo-Goldstein or the Wolfe conditions) that ensure a sufficient decrease at each step. Another common approach is **backtracking**: in this method one uses a high initial value of α (for example, $\alpha = 1$), and then decreases it until the sufficient descent condition is satisfied.

Convergence of iterative methods

A sequence of vectors $\{\boldsymbol{w}_k\}_{k\in\mathbb{N}_0} \subset \mathbb{R}^d$ *converges* to $\boldsymbol{w}^*$ with respect to a norm $\|\cdot\|$ as $k \to \infty$, written $\boldsymbol{w}_k \to \boldsymbol{w}$, if the sequence of numbers $\|\boldsymbol{w}_k - \boldsymbol{w}^*\|$ converges to zero. Iterative algorithms will typically not find the exact solution to a problem like (**??**). In fact, computers are not capable of telling very small numbers (say, 2^{-53} in double precision arithmetic) from 0, so finding a numerically exact solution is in general not possible. In addition, in machine learning, high accuracy is not necessary or even desirable due to the unavoidable statistical error.

Definition 14.2. Assume that a sequence of vectors $\{\boldsymbol{w}_k\}_{k\in\mathbb{N}_0}$ converges to $\boldsymbol{w}^*$. Then the sequence is said to converge

(a) **linearly** (or Q-linear, Q for Quotient), if there exist an $r \in (0, 1)$ such that for sufficiently large k,

$$\|\boldsymbol{w}_{k+1} - \boldsymbol{w}^*\| \leq r\|\boldsymbol{w}_k - \boldsymbol{w}^*\|.$$

(b) **superlinearly**, if

$$\lim_{k\to\infty} \frac{\|\boldsymbol{w}_{k+1} - \boldsymbol{w}^*\|}{\|\boldsymbol{w}_k - \boldsymbol{w}^*\|} = 0,$$

(c) with order p, if there exists a constant $M > 0$, such that for sufficiently large k,

$$\|\boldsymbol{w}_{k+1} - \boldsymbol{w}^*\| \leq M\|\boldsymbol{w}_k - \boldsymbol{w}^*\|^p.$$

The case $p = 2$ is called **quadratic convergence**.

These definitions depend on the choice of a norm, but any two norms on $\mathbb{R}^d$ are equivalent, convergence with respect to one norm implies convergence with respect to any other norm. Note that the definitions above start with the *assumption* that the sequence $\{\boldsymbol{w}_k\}$ converges to $\boldsymbol{w}^*$. Therefore, for sufficiently large k, $\|\boldsymbol{w}_k - \boldsymbol{w}^*\| < 1$ and if $\{\boldsymbol{w}_k\}$ converges with order of convergence $p > 1$, then

$$\frac{\|\boldsymbol{w}_{k+1} - \boldsymbol{w}^*\|}{\|\boldsymbol{w}_k - \boldsymbol{w}^*\|^{p-1}} \leq M\|\boldsymbol{w}_k - \boldsymbol{w}^*\| < M.$$

This shows that convergence of order p implies convergence of any lower order and also superlinear convergence.

Example 14.3. Consider the sequence of numbers $x_k = 1/2^{r^k}$ for some $r > 1$. Clearly, $x_k \to x^* = 0$ as $k \to \infty$. Moreover,

$$x_{k+1} = \frac{1}{2^{r^{k+1}}} = \frac{1}{2^{r^k r}} = \left(\frac{1}{2^{r^k}}\right)^r = x_k^r,$$

which shows that the sequence has rate of convergence r.

Convergence of gradient descent for least squares

Throughout this section, $\|\cdot\|$ refers to the 2-norm. We study the convergence of gradient descent for the least squares problem

$$\text{minimize } f(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}\|^2, \tag{14.4}$$

where $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$ is a matrix of full rank. The function $f(\boldsymbol{w})$ is convex, since it is a quadratic function with positive semi-definite Hessian $\boldsymbol{A}^T\boldsymbol{A}$. Gradient descent produces a sequence of vectors by the rule

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \alpha_k \boldsymbol{r}_k,$$

where the *step length* α_k and the *residual* $\boldsymbol{r}_k$ are given by

$$\alpha_k = \frac{\|\boldsymbol{r}_k\|^2}{\|\boldsymbol{A}\boldsymbol{r}_k\|^2}, \quad \boldsymbol{r}_k = \boldsymbol{A}^\intercal(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{w}_k) = -\nabla f(\boldsymbol{w}_k).$$

At the minimizer $\boldsymbol{w}^*$, the residual is $\boldsymbol{r} = -\nabla f(\boldsymbol{w}^*) = 0$ and if the sequence $\boldsymbol{w}_k$ converges to $\boldsymbol{w}^*$, the norms of the residuals converge to 0. Conversely, the residual is related to the difference $\boldsymbol{w}_k - \boldsymbol{w}^*$ by

$$\boldsymbol{r}_k = \boldsymbol{A}^\intercal(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{w}_k) = \boldsymbol{A}^\intercal(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{w}_k - (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{w}^*)) = \boldsymbol{A}^\intercal\boldsymbol{A}(\boldsymbol{x_k} - \boldsymbol{w}^*). \tag{14.5}$$

Therefore

$$\|\boldsymbol{w}_k - \boldsymbol{w}^*\| = \|(\boldsymbol{A}^\intercal\boldsymbol{A})^{-1}\boldsymbol{r}_k\| \leq \|(\boldsymbol{A}^\intercal\boldsymbol{A})^{-1}\|\|\boldsymbol{r}_k\|,$$

where $\|\boldsymbol{B}\| = \max_{\boldsymbol{w}\neq\boldsymbol{0}} \|\boldsymbol{B}\boldsymbol{w}\|/\|\boldsymbol{w}\|$ is the operator norm of a matrix $\boldsymbol{B}$ with respect to the 2-norm. Consequently, if the sequence $\|\boldsymbol{r}_k\|$ converges to zero, so does the sequence $\|\boldsymbol{w}_k - \boldsymbol{w}^*\|$. A reasonable criterion to stop the algorithm is therefore when the residual norm $\|\boldsymbol{r}_k\|$ is smaller than a predefined tolerance $\varepsilon > 0$.

The following theorem (whose proof we omit) shows that the gradient descent method for linear least squares converges linearly with respect to the $\boldsymbol{A}$ norm. The statement involves the *condition number* of $\boldsymbol{A}$[1]. This quantity is defined as

$$\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\| \cdot \|\boldsymbol{A}^\dagger\|,$$

where $\boldsymbol{A}^\dagger$ is the *pseudoinverse* of $\boldsymbol{A}$. If $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ with $m \geq n$ and linearly independent columns, it is defined as $\boldsymbol{A}^\dagger = (\boldsymbol{A}^\intercal \boldsymbol{A})^{-1} \boldsymbol{A}^\intercal$. The condition number is always greater than or equal to one.

Theorem 14.4. *The error in the $k+1$-th iterate is bounded by*

$$\|\boldsymbol{w}_{k+1} - \boldsymbol{w}^*\| \leq \left(\frac{\kappa^2(\boldsymbol{A}) - 1}{\kappa^2(\boldsymbol{A}) + 1} \right) \|\boldsymbol{w}_k - \boldsymbol{w}^*\|.$$

In particular, the gradient descent algorithm converges linearly. We can deduce Theorem 14.4 as a special case of a more general convergence result from convex function satisfying certain smoothness assumptions.

Notes

The precise definitions of rates of convergence may vary, but for all practical purposes they are equivalent. Our definitions are based on [60]. A well-known class of iterative methods in optimization are interior-point methods. Iterative methods have traditionally been studied in the context of solving equations. In fact, gradient descent methods aim to solve the system of equations $\nabla f(\boldsymbol{w}) = \boldsymbol{0}$. Newton's method for solving certain polynomial equations goes back to Isaac Newton. The first use of an iterative method for solving a system of linear equations is attributed to Gauss, who outlined a method for solving 4×4 systems in a letter to a student. Iterative methods gained increased attention with the advent of computers and floating-point arithmetic, and a milestone in the the development of iterative methods is the conjugate gradient method by Hestenes and Stiefel [36]. For a historical overview of iterative methods for solving systems of linear equations, see [67]. The analysis of Newton's method, which makes use of second-order information, is more complicated. Newton's method converges *quadratically* when starting sufficiently close to a minimizer (or root of a function, if used for solving equations). The convergence behaviour of Newton's method for root-finding over the complex numbers has lead to the discovery of some interesting phenomena, as shown in Figure 14.3.

A generalization of Newton's method to systems of non-linear equations forms the basis of homotopy methods, and their analysis in terms of condition numbers has been an active field of research, and an overview can be found in [12, 17].

[1]The concept of condition number, introduced by Alan Turing while in Manchester, is one of the most important ideas in numerical analysis, as it is indispensable in studying the performance of numerical algorithms.

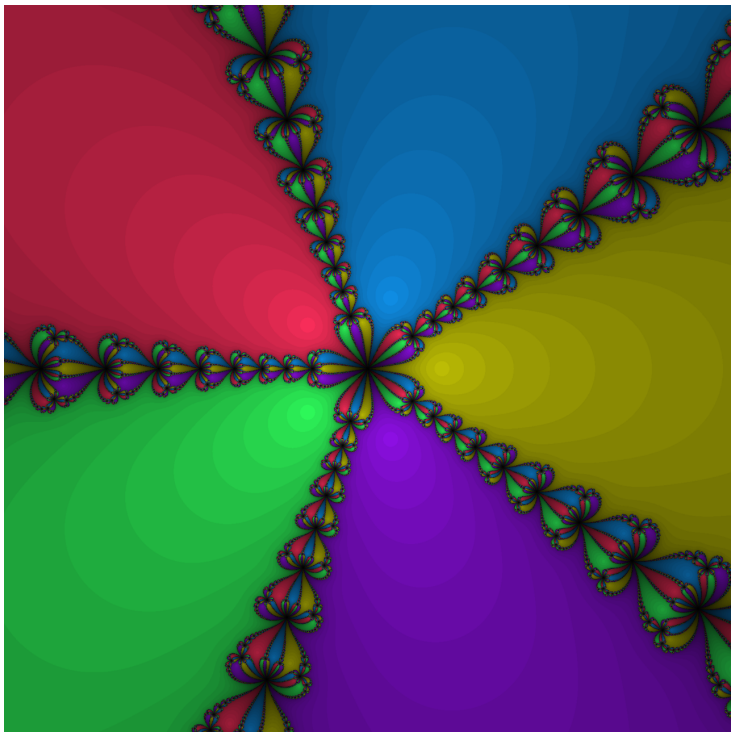


Figure 14.3: Basins of attraction of Newton's method for solving $x^5 - 1 = 0$. Each colour represents the region in which a starting point leads to a sequence of iterates that converges to one of the five complex roots. Image rights: GPL, https://commons.wikimedia.org/w/index.php?curid=644886

15

Gradient Descent

In this chapter we derive a convergence result for gradient descent applied to a convex function $f \in C^1(\mathbb{R}^d)$. In order to get linear convergence, we need to place certain restrictions on our function. Specifically, we require that the **Bregman divergence** associated to f,

$$\mathcal{D}_f(\boldsymbol{w}, \boldsymbol{v}) := f(\boldsymbol{w}) - f(\boldsymbol{v}) - \langle \nabla f(\boldsymbol{v}), (\boldsymbol{w} - \boldsymbol{v}) \rangle,$$

is bounded from below and above by quadratic functions,

$$\frac{\alpha}{2}\|\boldsymbol{w} - \boldsymbol{v}\|^2 \leq \mathcal{D}_f(\boldsymbol{w}, \boldsymbol{v}) \leq \frac{\beta}{2}\|\boldsymbol{w} - \boldsymbol{v}\|^2,$$

for some constants $\alpha \leq \beta$. Properties that ensure this are **strong convexity** and **Lipschitz continuity of the gradient**.

Gradient descent for smooth convex functions

The most common smoothness condition in optimization is Lipschitz continuity, applied to a function or to its gradient.

Definition 15.1. A function $f\colon \mathbb{R}^d \to \mathbb{R}^k$ is called **Lipschitz continuous** with Lipschitz constant $L > 0$, if for all $\boldsymbol{w}, \boldsymbol{v} \in \mathbb{R}^d$,

$$\|f(\boldsymbol{w}) - f(\boldsymbol{v})\| \leq L \cdot \|\boldsymbol{w} - \boldsymbol{v}\|.$$

A function $f\colon \mathbb{R}^d \to \mathbb{R}$ is called β**-smooth** for some $\beta > 0$ if the gradient is Lipschitz continuous:

$$\|\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v})\| \leq \beta \cdot \|\boldsymbol{w} - \boldsymbol{v}\|$$

for all $\boldsymbol{w}, \boldsymbol{v} \in \mathbb{R}^d$.

Lipschitz continuity lies somewhere between continuity and differentiability. If $f \in C^1(\mathbb{R}^d)$ is Lipschitz continuous with Lipschitz constant L, then $\|\nabla f(\boldsymbol{w})\| \leq L$. Similarly, β-smoothness implies that $\|\nabla^2 f(\boldsymbol{w})\| \leq \beta$.

Recall that a function $f \in C^1(\mathbb{R}^d)$ is convex if and only if

$$f(\boldsymbol{v}) + \langle \nabla f(\boldsymbol{v}), (\boldsymbol{w} - \boldsymbol{v}) \rangle \leq f(\boldsymbol{w}) \tag{15.1}$$

for all $\boldsymbol{w}, \boldsymbol{v} \in \mathbb{R}^d$. The following result shows that β-smoothness is equivalent to a quadratic upper bound on the difference between the function value $f(\boldsymbol{w})$ and its linear lower bound (15.1).

Lemma 15.2. *Let $f \in C^1(\mathbb{R}^d)$ be β-smooth and convex. Then for any $\boldsymbol{w}, \boldsymbol{v} \in \mathbb{R}^d$,*

$$f(\boldsymbol{w}) \leq f(\boldsymbol{v}) + \langle \nabla f(\boldsymbol{v}), (\boldsymbol{w} - \boldsymbol{v}) \rangle + \frac{\beta}{2} \|\boldsymbol{w} - \boldsymbol{v}\|^2. \tag{15.2}$$

Conversely, if a convex function $f \in C^1(\mathbb{R}^d)$ satisfies (15.2), then for all $\boldsymbol{w}, \boldsymbol{v} \in \mathbb{R}^d$,

$$\frac{1}{\beta} \|\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v})\|^2 \leq \langle (\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v})), (\boldsymbol{w} - \boldsymbol{v}) \rangle. \tag{15.3}$$

In particular, f is β-smooth.

Proof. For the first inequality, consider the function

$$\varphi(t) = f(\boldsymbol{v} + t(\boldsymbol{w} - \boldsymbol{v})).$$

The derivative $\varphi'(t)$ is then the directional derivative

$$\varphi'(t) = \langle \nabla f(\boldsymbol{v} + t(\boldsymbol{w} - \boldsymbol{v})), (\boldsymbol{w} - \boldsymbol{v}) \rangle,$$

and we can represent $f(\boldsymbol{w}) - f(\boldsymbol{v})$ as an integral:

$$f(\boldsymbol{w}) - f(\boldsymbol{v}) = \int_0^1 \langle \nabla f(\boldsymbol{v} + t(\boldsymbol{w} - \boldsymbol{v})), (\boldsymbol{w} - \boldsymbol{v}) \rangle \, \mathrm{d}t.$$

We can then write

$$\begin{aligned} f(\boldsymbol{w}) - f(\boldsymbol{v}) - \langle \nabla f(\boldsymbol{v}), (\boldsymbol{w} - \boldsymbol{v}) \rangle &= \int_0^1 \langle \nabla f(\boldsymbol{v} + t(\boldsymbol{w} - \boldsymbol{v})), (\boldsymbol{w} - \boldsymbol{v}) \rangle \\ &\qquad - \langle \nabla f(\boldsymbol{v}), (\boldsymbol{w} - \boldsymbol{v}) \rangle \, \mathrm{d}t \\ &\leq \int_0^1 \|\nabla f(\boldsymbol{v} + t(\boldsymbol{w} - \boldsymbol{v})) - \nabla f(\boldsymbol{v})\| \\ &\qquad \cdot \|\boldsymbol{w} - \boldsymbol{v}\| \, \mathrm{d}t \\ &\leq \beta \int_0^1 t \|\boldsymbol{w} - \boldsymbol{v}\|^2 \, \mathrm{d}t = \frac{\beta}{2} \|\boldsymbol{w} - \boldsymbol{v}\|^2, \end{aligned}$$

where the first inequality follows from applying Cauchy-Schwartz, and the second from the assumption of β-smoothness.

For the second claim, assume that f satisfies the bound (15.2). For any $\boldsymbol{w}, \boldsymbol{v}, \boldsymbol{z} \in \mathbb{R}^d$ we have

$$\begin{aligned} f(\boldsymbol{w}) - f(\boldsymbol{v}) &= f(\boldsymbol{w}) - f(\boldsymbol{z}) + f(\boldsymbol{z}) - f(\boldsymbol{v}) \\ &\leq \langle \nabla f(\boldsymbol{w}), (\boldsymbol{w} - \boldsymbol{z}) \rangle + \langle \nabla f(\boldsymbol{v}), (\boldsymbol{z} - \boldsymbol{v}) \rangle + \frac{\beta}{2} \|\boldsymbol{z} - \boldsymbol{v}\|^2, \end{aligned}$$

where we used the convexity of f to bound $f(\boldsymbol{w}) - f(\boldsymbol{z})$ and the β-smoothness to bound $f(\boldsymbol{z}) - f(\boldsymbol{v})$. If we now set $\boldsymbol{z} = \boldsymbol{v} - (1/\beta)(\nabla f(\boldsymbol{v}) - \nabla f(\boldsymbol{w}))$ and simplify the resulting expression, we get

$$f(\boldsymbol{w}) - f(\boldsymbol{v}) \leq \langle \nabla f(\boldsymbol{w}), (\boldsymbol{w} - \boldsymbol{v}) \rangle - \frac{1}{2\beta} \|\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v})\|^2.$$

Adding this expression to the same one with the roles of $\boldsymbol{w}$ and $\boldsymbol{v}$ exchanged, we get

$$0 \leq (\nabla f(\boldsymbol{w}) - \langle \nabla f(\boldsymbol{v})), (\boldsymbol{w} - \boldsymbol{v}) \rangle - \frac{1}{\beta} \|\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v})\|^2.$$

The fact that this implies β-smoothness of f follows from the Cauchy-Schwartz inequality. □

We can, and will, use (15.2) and (15.3) as alternative definitions of β-smoothness. While β-smoothness is enough to prove convergence of gradient descent, we can get better results, reminiscent of the condition number bounds for quadratic functions, when assuming **strong convexity**.

Definition 15.3. A function $f \in C^1(\mathbb{R}^d)$ is called **α-strongly convex** for some $\alpha > 0$ if for every $\boldsymbol{w}, \boldsymbol{v} \in \mathbb{R}^d$,

$$f(\boldsymbol{v}) + \langle \nabla f(\boldsymbol{v}), (\boldsymbol{w} - \boldsymbol{v}) \rangle + \frac{\alpha}{2}\|\boldsymbol{w} - \boldsymbol{v}\|^2 \leq f(\boldsymbol{w}).$$

If $\alpha = 0$ this is just the derivative characterization of convexity. Note that a function f is α-strongly convex if and only if the function $f(\boldsymbol{w}) - \alpha\|\boldsymbol{w}\|^2/2$ is convex.

Theorem 15.4. *Let $f \in C^1(\mathbb{R}^d)$ be a function that is α-strongly convex and β-smooth. Then for any $\boldsymbol{w}^0 \in \mathbb{R}^d$, the iterates of gradient descent with constant step length $2/(\alpha + \beta)$ satisfy*

$$\|\boldsymbol{w}^{k+1} - \boldsymbol{w}^*\| \leq \left(\frac{\kappa - 1}{\kappa + 1}\right) \|\boldsymbol{w}^k - \boldsymbol{w}^*\|,$$

where $\kappa = \beta/\alpha$.

Example 15.5. Assume that $\alpha = \beta$ (and therefore $\kappa = 1$) in Theorem 15.4. Then

$$f(\boldsymbol{w}) = \frac{\alpha}{2}\|\boldsymbol{w}\|^2.$$

The gradient is $\nabla f(\boldsymbol{w}) = \alpha \boldsymbol{w}$. Starting with $\boldsymbol{w}^0 \in \mathbb{R}^d$, gradient descent with step length $2/(\alpha+\beta) = 1/\alpha$ gives

$$\boldsymbol{w}_1 = \boldsymbol{w}^0 - \boldsymbol{w}^0 = \boldsymbol{0} = \boldsymbol{w}^*,$$

so that this converges in a single iteration.

Example 15.6. Let $f(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}\|^2$ for $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ with $m \geq n$, and assume $\boldsymbol{A}$ has full rank. The difference between the function and its approximation is

$$\frac{1}{2}\left(\|\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}\|^2 - \|\boldsymbol{A}\boldsymbol{v} - \boldsymbol{b}\|^2\right) - (\boldsymbol{A}\boldsymbol{v} - \boldsymbol{b})^T\boldsymbol{A}(\boldsymbol{w} - \boldsymbol{v}) = \frac{1}{2}\|\boldsymbol{A}(\boldsymbol{w} - \boldsymbol{v})\|^2. \tag{15.4}$$

The largest and smallest **singular values** of the matrix $\boldsymbol{A}$ are defined as

$$\sigma_1(\boldsymbol{A}) = \max_{\|\boldsymbol{w}\|=1} \|\boldsymbol{A}\boldsymbol{w}\|, \qquad \sigma_n(\boldsymbol{A}) = \min_{\|\boldsymbol{w}\|=1} \|\boldsymbol{A}\boldsymbol{w}\|.$$

The term (15.4) is therefore bounded from above and below by the squares of the *largest singular value* and by the *smallest singular value* of $\boldsymbol{A}$:

$$\sigma_1^2(\boldsymbol{A})\|\boldsymbol{w} - \boldsymbol{v}\|^2 \geq \|\boldsymbol{A}(\boldsymbol{w} - \boldsymbol{v})\|^2 \geq \sigma_n^2(\boldsymbol{A})\|\boldsymbol{w} - \boldsymbol{v}\|^2.$$

A well-known characterization of the condition number of a matrix is $\kappa(\boldsymbol{A}) = \sigma_1(\boldsymbol{A})/\sigma_n(\boldsymbol{A})$, and from this we recover the convergence result from the quadratic case.

The proof of Theorem 15.4 relies on the following auxiliary result.

Lemma 15.7. *Let f be α-strongly convex and β smooth. Then for any $\boldsymbol{w}, \boldsymbol{v} \in \mathbb{R}^d$,*

$$\langle (\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v})), (\boldsymbol{w} - \boldsymbol{v}) \rangle \geq \frac{\alpha\beta}{\alpha + \beta}\|\boldsymbol{w} - \boldsymbol{v}\|^2 + \frac{1}{\alpha + \beta}\|\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v})\|^2.$$

Proof. Set $\varphi(\boldsymbol{w}) := f(\boldsymbol{w}) - \frac{\alpha}{2}\|\boldsymbol{w}\|^2$. Since f is α-strongly convex, $\varphi(\boldsymbol{w})$ is convex. Moreover, $\nabla\varphi(\boldsymbol{w}) = \nabla f(\boldsymbol{w}) - \alpha\boldsymbol{w}$. We therefore get

$$\begin{aligned}\varphi(\boldsymbol{w}) - \varphi(\boldsymbol{v}) - \langle\nabla\varphi(\boldsymbol{v}), (\boldsymbol{w}-\boldsymbol{v})\rangle &= f(\boldsymbol{w}) - f(\boldsymbol{v}) - \langle\nabla f(\boldsymbol{v}), (\boldsymbol{w}-\boldsymbol{v})\rangle \\ &\quad - \frac{\alpha}{2}(\|\boldsymbol{w}\|^2 - \|\boldsymbol{v}\|^2) + \alpha\langle\boldsymbol{v}, (\boldsymbol{w}-\boldsymbol{v})\rangle \\ &\overset{\text{Lemma 15.2}}{\leq} \frac{\beta}{2}\|\boldsymbol{w}-\boldsymbol{v}\|^2 \\ &\quad - \frac{\alpha}{2}(\|\boldsymbol{w}\|^2 + \|\boldsymbol{v}\|^2 - 2\langle\boldsymbol{v}, \boldsymbol{w}\rangle) \\ &= \frac{\beta-\alpha}{2}\|\boldsymbol{w}-\boldsymbol{v}\|^2.\end{aligned} \tag{15.5}$$

From Lemma 15.2 it follows that φ is β-smooth and satisfies the inequality

$$\langle(\nabla\varphi(\boldsymbol{w}) - \nabla\varphi(\boldsymbol{v})), (\boldsymbol{w}-\boldsymbol{v})\rangle \geq \frac{1}{\beta-\alpha}\|\nabla\varphi(\boldsymbol{w}) - \nabla\varphi(\boldsymbol{v})\|^2.$$

Replacing φ in this expression, we get

$$\langle(\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v}) - \alpha\boldsymbol{w} + \alpha\boldsymbol{v}), (\boldsymbol{w}-\boldsymbol{v})\rangle \geq \frac{1}{\beta-\alpha}\|\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v}) - \alpha\boldsymbol{w} + \alpha\boldsymbol{v}\|^2.$$

The left-hand side of this inequality gives

$$\langle(\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v}) - \alpha\boldsymbol{w} + \alpha\boldsymbol{v}), (\boldsymbol{w}-\boldsymbol{v})\rangle = \langle(\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v})), (\boldsymbol{w}-\boldsymbol{v})\rangle - \alpha\|\boldsymbol{w}-\boldsymbol{v}\|^2. \tag{15.6}$$

The right-hand side, on the other hand, gives

$$\begin{aligned}\frac{1}{\beta-\alpha}\|\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v}) - \alpha\boldsymbol{w} + \alpha\boldsymbol{v}\|^2 = \frac{1}{\beta-\alpha}\big(&\|\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v})\|^2 \\ &+ \alpha^2\|\boldsymbol{w}-\boldsymbol{v}\|^2 \\ &- 2\alpha\langle(\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v})), (\boldsymbol{w}-\boldsymbol{v})\rangle\big).\end{aligned} \tag{15.7}$$

Collecting the terms in (22.2) and (15.7) involving $\langle(\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v})), (\boldsymbol{w}-\boldsymbol{v})\rangle$ on the left, and the terms involving $\|\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v})\|^2$ and $\|\boldsymbol{w}-\boldsymbol{v}\|^2$ on the right, we get

$$\frac{\alpha+\beta}{\beta-\alpha}\langle(\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v})), (\boldsymbol{w}-\boldsymbol{v})\rangle \geq \frac{\alpha\beta}{\beta-\alpha}\|\boldsymbol{w}-\boldsymbol{v}\|^2 + \frac{1}{\beta-\alpha}\|\nabla f(\boldsymbol{w}) - \nabla f(\boldsymbol{v})\|^2.$$

Multiplying this expression with $(\beta-\alpha)/(\alpha+\beta)$ gives the desired inequality. □

Proof of Theorem 15.4. Set $\eta := 2/(\alpha+\beta)$. Since $\nabla f(\boldsymbol{w}^*) = 0$, we will omit this term whenever it appears in the following (so that we can think of $\nabla f(\boldsymbol{w}^k) - \nabla f(\boldsymbol{w}^*)$ whenever we see $\nabla f(\boldsymbol{w}^k)$).

$$\begin{aligned}\|\boldsymbol{w}^{k+1} - \boldsymbol{w}^*\|^2 &= \|\boldsymbol{w}^k - \eta\nabla f(\boldsymbol{w}^k) - \boldsymbol{w}^*\|^2 = \|(\boldsymbol{w}^k - \boldsymbol{w}^*) - \eta\nabla f(\boldsymbol{w}^k)\|^2 \\ &= \|\boldsymbol{w}^k - \boldsymbol{w}^*\|^2 - 2\eta\langle\nabla f(\boldsymbol{w}^k), (\boldsymbol{w}^k - \boldsymbol{w}^*)\rangle + \eta^2\|\nabla f(\boldsymbol{w}^k)\|^2 \\ &\leq \left(\frac{\beta-\alpha}{\beta+\alpha}\right)^2\|\boldsymbol{w}^k - \boldsymbol{w}^*\|^2,\end{aligned}$$

where in the inequality we used Lemma 15.7 to bound the term $\langle\nabla f(\boldsymbol{w}^k), (\boldsymbol{w}^k - \boldsymbol{w}^*)\rangle$, and simplified the resulting expression. The claim now follows by dividing the numerator and the denominator by α. □

Using the bound

$$\left(\frac{\kappa-1}{\kappa+1}\right)^2 = \left(1-\frac{2}{\kappa+1}\right)^2 \leq e^{-4/(\kappa+1)},$$

and the inequality $f(\boldsymbol{w}^k) - f(\boldsymbol{w}^*) \leq (\beta/2)\|\boldsymbol{w}^k - \boldsymbol{w}^*\|^2$ from the β-smoothness assumption, we get the convergence bound

$$f(\boldsymbol{w}^k) - f(\boldsymbol{w}^*) \leq \frac{\beta\|\boldsymbol{w}^0 - \boldsymbol{w}^*\|^2}{2} \cdot e^{-\frac{4}{\kappa+1}k},$$

which is a considerable improvement over the $1/k$ convergence bound that we would get when only assuming β-smoothness. In particular, the number of iterations to reach accuracy ϵ is of order $O(\log(1/\epsilon))$.

Notes

The convergence of gradient descent under various smoothness assumptions is a classic theme in convex optimization. The presentation in this chapter is based on [16]. A standard reference for many of the tools used in the analysis of gradient descent (and a wealth of additional information) [59].

16

Stochastic Gradient Descent

In this lecture we introduce **Stochastic Gradient Descent** (SGD), a probabilistic version of gradient descent that has been around since the 1950s, and that has become popular in the context of data science and machine learning. To motivate the algorithm, consider a set of functions $\mathcal{H} = \{h_{\boldsymbol{w}} \colon \boldsymbol{w} \in \mathbb{R}^d\}$, where each such function depends on d parameters. Also consider a smooth loss functions L, or a smooth approximation of a loss function. Given samples $\{(\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^n \subset \mathcal{X} \times \mathcal{Y}$, define the functions

$$f_i(\boldsymbol{w}) = L(h_{\boldsymbol{w}}(\boldsymbol{x}_i), \boldsymbol{y}_i), \quad i \in \{1, \dots, n\}.$$

The problem of finding functions that minimize the empirical risk is

$$\underset{\boldsymbol{w} \in \mathbb{R}^d}{\text{minimize}} \ \frac{1}{n} \sum_{i=1}^n f_i(\boldsymbol{w}).$$

The f_i are often assumed to be convex and smooth. In addition one often considers a regularization term $R(\boldsymbol{w})$. In what follows, we abstract from the machine learning context and consider purely the associated optimization problem.

Stochastic Gradient Descent

We consider an objective function of the form

$$f(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^n f_i(\boldsymbol{w}). \tag{16.1}$$

In what follows we assume that the functions f_i are convex and differentiable. If n is large, then computing the gradient can be very expensive. However, and considering the machine learning context, where $f(\boldsymbol{w})$ is an estimator of the generalization risk $\mathbb{E}_{\boldsymbol{\xi}}[f_{\boldsymbol{\xi}}(\boldsymbol{w})]$ of a family of functions $f_{\boldsymbol{\xi}}$ parametrized by a random vector $\boldsymbol{\xi}$, we can shift the focus to finding an **unbiased estimator** of the gradient. Quite trivially, choosing an index j uniformly at random and computing the gradient of $f_j(\boldsymbol{w})$ gives such an unbiased estimator *by definition*:

$$\mathbb{E}_U[f_U(\boldsymbol{w})] = \frac{1}{n} \sum_{i=1}^n f_i(\boldsymbol{w}),$$

where $\mathbb{P}\{U = j\} = 1/n$ for $j \in [n] = \{1, \dots, n\}$. The Stochastic Gradient Descent (SGD) algorithm proceeds as follows. Begin with $\boldsymbol{w}^0 \in \mathbb{R}^d$. At each step k, compute an unbiased estimator $\boldsymbol{g}^k$ of the gradient at $\boldsymbol{w}^k$:

$$\mathbb{E}[\boldsymbol{g}^k \mid \boldsymbol{w}^k] = \nabla f(\boldsymbol{w}^k).$$

Next, take a step in direction $\boldsymbol{g}^k$:

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta_k \boldsymbol{g}^k,$$

where η_k is a step length (or **learning rate** in machine learning jargon).

While there are many variants of stochastic gradient descent, we consider the simplest version in which $\boldsymbol{g}^k$ is chosen by picking one of the gradients $\nabla f_i(\boldsymbol{w})$ uniformly at random, and we refer to this as SGD *with uniform sampling*. A commonly used generalization is **mini-batch** sampling, where one chooses a small set of indices $I \subset \{1, \dots, n\}$ at random, instead of only one. We also restrict to the smooth setting without a regularization term; in the non-smooth setting one would apply a proximal operator. Since SGD involves random choices, convergence results are stated in terms of the expected value. Let U be a random variable with distribution $\mathbb{P}\{U = i\} = 1/n$ for $i \in [n]$. Then

$$\mathbb{E}_U[\nabla f_U(\boldsymbol{w})] = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\boldsymbol{w}) = \nabla f(\boldsymbol{w}),$$

so that ∇f_U is an unbiased estimator of ∇f. Assuming that f has a unique minimizer $\boldsymbol{w}^*$, we define the empirical **variance** at the optimal point $\boldsymbol{w}^*$ as

$$\sigma^2 = \mathbb{E}_U[\|\nabla f_U(\boldsymbol{w}^*)\|^2] = \frac{1}{n} \sum_{i=1}^{n} \|\nabla f_i(\boldsymbol{w}^*)\|^2. \tag{16.2}$$

We can now state the convergence result for stochastic gradient descent.

Theorem 16.1. *Assume the function f is α-strongly convex and that the f_i are convex and β-smooth for $i \in [n]$ and $4\beta > \alpha$. Assume f has a unique minimizer $\boldsymbol{w}^*$ and define the variance as in* (16.2)*. Then for any staring point $\boldsymbol{w}^0$, the sequence of iterates $\{\boldsymbol{w}^k\}$ generated by SGD with uniform sampling and step length $\eta = 1/(2\beta)$ satisfies*

$$\mathbb{E}[\|\boldsymbol{w}^k - \boldsymbol{w}^*\|^2] \leq \left(1 - \frac{\alpha}{4\beta}\right)^k \|\boldsymbol{w}^0 - \boldsymbol{w}^*\|^2 + \frac{2\sigma^2}{\alpha\beta}.$$

Proof. As in the analysis of gradient descent, we get

$$\|\boldsymbol{w}^{k+1} - \boldsymbol{w}^*\|^2 = \|\boldsymbol{w}^k - \boldsymbol{w}^*\|^2 - 2\eta\langle \nabla f_U(\boldsymbol{w}^k), (\boldsymbol{w}^k - \boldsymbol{w}^*)\rangle + \eta^2 \|\nabla f_U(\boldsymbol{w}^k)\|^2.$$

Taking the expectation conditional on $\boldsymbol{w}^k$, we get

$$\begin{aligned} \mathbb{E}[\|\boldsymbol{w}^{k+1} - \boldsymbol{w}^*\|^2 \mid \boldsymbol{w}^k] = \|\boldsymbol{w}^k - \boldsymbol{w}^*\|^2 - 2\eta\langle \nabla f(\boldsymbol{w}^k), (\boldsymbol{w}^k - \boldsymbol{w}^*)\rangle \\ + \eta^2 \, \mathbb{E}[\|\nabla f_U(\boldsymbol{w}^k)\|^2 \mid \boldsymbol{w}^k], \end{aligned} \tag{16.3}$$

where we used the fact that the expectation satisfies $\mathbb{E}[\nabla f_U(\boldsymbol{w})] = \nabla f(\boldsymbol{w})$. For the last term we use the bound

$$\begin{aligned} \mathbb{E}[\|\nabla f_U(\boldsymbol{w}^k)\|^2 \mid \boldsymbol{w}^k] &= \mathbb{E}[\|\nabla f_U(\boldsymbol{w}^k) - \nabla f_U(\boldsymbol{w}^*) + \nabla f_U(\boldsymbol{w}^*)\|^2] \\ &\leq 2\,\mathbb{E}[\|\nabla f_U(\boldsymbol{w}^k) - \nabla f_U(\boldsymbol{w}^*)\|^2] + 2\,\mathbb{E}[\|\nabla f_U(\boldsymbol{w}^*)\|^2] \\ &= 2\,\mathbb{E}[\|\nabla f_U(\boldsymbol{w}^k) - \nabla f_U(\boldsymbol{w}^*)\|^2] + 2\sigma^2. \end{aligned}$$

Using the characterization of β smoothness from the previous chapter, namely

$$\frac{1}{\beta}\|\nabla f_i(\boldsymbol{w}) - \nabla f_i(\boldsymbol{v})\|^2 \leq \langle (\nabla f_i(\boldsymbol{w}) - \nabla f_i(\boldsymbol{v})), (\boldsymbol{w} - \boldsymbol{v})\rangle, \tag{16.4}$$

we get that

$$\begin{aligned}\mathbb{E}[\|\nabla f_U(\boldsymbol{w}^k) - \nabla f_U(\boldsymbol{w}^*)\|^2 \mid \boldsymbol{w}^k] &= \frac{1}{n}\sum_{i=1}^n \|\nabla f_i(\boldsymbol{w}^k) - \nabla f_i(\boldsymbol{w}^*)\|^2 \\ &\overset{(16.4)}{\leq} \frac{\beta}{n}\sum_{i=1}^n \langle(\nabla f_i(\boldsymbol{w}^k) - \nabla f_i(\boldsymbol{w}^*)), (\boldsymbol{w}^k - \boldsymbol{w}^*)\rangle \\ &= \frac{\beta}{n}\sum_{i=1}^n \langle \nabla f_i(\boldsymbol{w}^k), (\boldsymbol{w}^k - \boldsymbol{w}^*)\rangle \\ &\qquad - \frac{\beta}{n}\sum_{i=1}^n \langle \nabla f_i(\boldsymbol{w}^*), (\boldsymbol{w}^k - \boldsymbol{w}^*)\rangle \\ &= \beta\langle \nabla f(\boldsymbol{w}), (\boldsymbol{w}^k - \boldsymbol{w}^*)\rangle,\end{aligned}$$

where we used that $\nabla f(\boldsymbol{w}^*) = \mathbf{0}$ for the last equality. Hence, we get the bound

$$\mathbb{E}[\|\nabla f_U(\boldsymbol{w}^k)\|^2 \mid \boldsymbol{w}^k] \leq 2\beta\langle \nabla f(\boldsymbol{w}), (\boldsymbol{w}^k - \boldsymbol{w}^*)\rangle + 2\sigma^2.$$

Plugging this into (16.3), we get

$$\mathbb{E}[\|\boldsymbol{w}^{k+1} - \boldsymbol{w}^*\|^2 \mid \boldsymbol{w}^k] \leq \|\boldsymbol{w}^k - \boldsymbol{w}^*\|^2 - (2\eta - 2\eta^2\beta)\langle \nabla f(\boldsymbol{w}^k), (\boldsymbol{w}^k - \boldsymbol{w}^*)\rangle + 2\eta^2\sigma^2.$$

Using the α-strong convexity, we get the bound

$$\langle \nabla f(\boldsymbol{w}^k), (\boldsymbol{w}^k - \boldsymbol{w}^*)\rangle \geq f(\boldsymbol{w}^k) - f(\boldsymbol{w}^*) + \frac{\alpha}{2}\|\boldsymbol{w}^k - \boldsymbol{w}^*\|^2 \geq \frac{\alpha}{2}\|\boldsymbol{w}^k - \boldsymbol{w}^*\|^2,$$

since $f(\boldsymbol{w}^k) \geq f(\boldsymbol{w}^*)$, so that we get

$$\mathbb{E}[\|\boldsymbol{w}^{k+1} - \boldsymbol{w}^*\|^2 \mid \boldsymbol{w}^k] \leq (1 - \eta\alpha(1 - \eta\beta))\|\boldsymbol{w}^k - \boldsymbol{w}^*\|^2 + 2\eta^2\sigma^2.$$

With step length $\eta = 1/(2\beta)$, and taking the expected value over all previous iterates, we get

$$\mathbb{E}[\|\boldsymbol{w}^{k+1} - \boldsymbol{w}^*\|^2] \leq \left(1 - \frac{\alpha}{4\beta}\right)\mathbb{E}[\|\boldsymbol{w}^k - \boldsymbol{w}^*\|^2] + \frac{\sigma^2}{2\beta^2}.$$

Applying this bound recursively (and moving the index down), we get

$$\begin{aligned}\mathbb{E}[\|\boldsymbol{w}^k - \boldsymbol{w}^*\|^2] &\leq \left(1 - \frac{\alpha}{4\beta}\right)^k \|\boldsymbol{w}^0 - \boldsymbol{w}^*\|^2 + \frac{\sigma^2}{2\beta^2}\sum_{j=0}^{k-1}\left(1 - \frac{\alpha}{4\beta}\right)^j \\ &\leq \left(1 - \frac{\alpha}{4\beta}\right)^k \|\boldsymbol{w}^0 - \boldsymbol{w}^*\|^2 + \frac{2\sigma^2}{\alpha\beta},\end{aligned}$$

where we used that $4\beta > \alpha$. □

Example 16.2. Consider the problem of **logistic regression**, where the aim is to minimize the objective function

$$f(\boldsymbol{w}) = \sum_{i=1}^n \left(\log\left(1 + \exp(\boldsymbol{x}_i^T\boldsymbol{w})\right) - y_i\boldsymbol{x}_i^T\boldsymbol{w}\right)$$

over a vector of weights $\boldsymbol{w}$. This problem arises in the context of a binary classification problem with data pairs $(\boldsymbol{x}_i, y_i)$ and $y_i \in \{0, 1\}$. Setting

$$p := \frac{\mathrm{e}^{\boldsymbol{x}^T \boldsymbol{w}}}{1 + \mathrm{e}^{\boldsymbol{x}^T \boldsymbol{w}}},$$

the resulting classifier is the function

$$h(\boldsymbol{w}) = \begin{cases} 1 & p > 1/2 \\ 0 & p \leq 1/2 \end{cases}$$

The function f is convex, and the gradient is

$$\nabla f(\boldsymbol{w}) = -\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{p}(\boldsymbol{w})),$$

where $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ is the matrix with the $\boldsymbol{x}_i^T$ as rows, $\boldsymbol{y} = (y_1, \ldots, y_n)^T$, and $\boldsymbol{p}(\boldsymbol{w}) \in \mathbb{R}^n$ has coordinates

$$p_i(\boldsymbol{w}) = \frac{\exp(\boldsymbol{x}_i^T \boldsymbol{w})}{1 + \exp(\boldsymbol{x}_i^T \boldsymbol{w})}, \quad 1 \leq i \leq n.$$

We can apply different versions of gradient descent to this problem. Figure 1 shows the typical paths of gradient descent and of stochastic gradient descent for a problem with 100 data points. Note that using a naive approach to computing the gradient, one would need to compute 100 gradients at each step. Stochastic gradient descent, on the other hand, fails to converge due to the variance of the gradient estimator (see Figure 2).

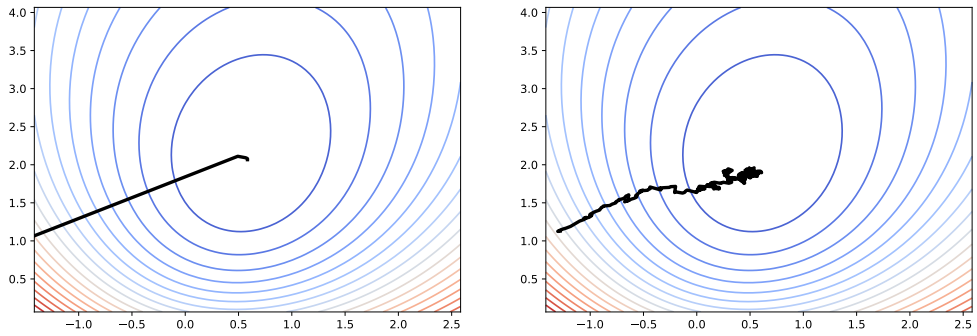


Figure 16.1: The path of gradient descent and of stochastic gradient descent with constant step length.

Extensions

The version of SGD described here is the most basic one. There are many possible extensions to the methods. The include considering different sampling schemes, including **mini-batching** and **importance sampling**. These sampling strategies have the effect of reducing the variance σ^2. In addition, improvements can be made in the step length selection and when dealing with non-smooth function, where the proximal operator comes into play.

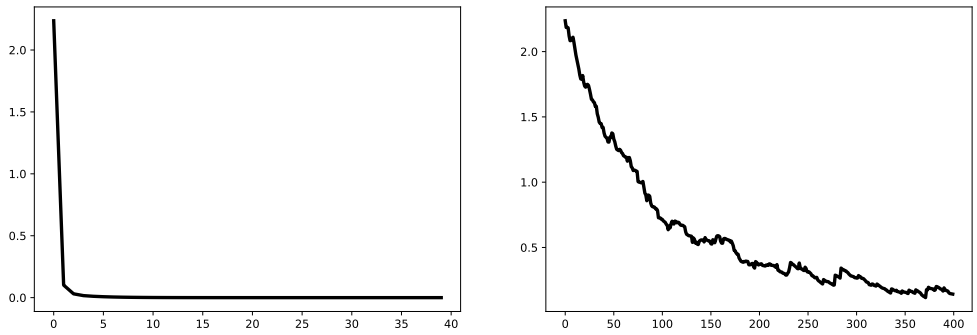


Figure 16.2: Convergence of gradient descent and SGD.

Notes

The origins of stochastic gradient descent go back to the work of Robbins and Monro in 1951 [63]. The algorithm has been rediscovered many times, and gained popularity due to its effectiveness in training deep neural networks on large data sets, where gradient computations are very expensive. Despite its simplicity, a systematic and rigorous analysis has not been available until recently. The presentation in this chapter is based loosely on the papers [32] and [13]. A more general and systematic analysis of SGD that includes non-smooth objectives is given in [31]. These works also discuss general sampling techniques, not just uniform sampling.

Part III

Topics in Deep Learning

17

Neural Networks

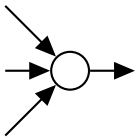
Originally introduced as simplified models of neurons in the brain, (artificial) neural networks are now one of the most widely used models in machine learning and data science. Their popularity owes to their ability to combine generality with computational tractability: while a neural network can approximate virtually any reasonable function to arbitrary accuracy, its structure is still simple enough so that it can be trained efficiently by gradient descent.

Connectivity and Activation

We begin by revisiting the problem of binary classification using a linear function. Given a vector of weights $\boldsymbol{w} \in \mathbb{R}^d$ and a bias term $b \in \mathbb{R}$, define the binary classifier

$$h_{\boldsymbol{w},b}(\boldsymbol{x}) = \mathbf{1}\{\boldsymbol{w}^T\boldsymbol{x} + b > 0\} = \begin{cases} 1 & \boldsymbol{w}^T\boldsymbol{x} + b > 0 \\ 0 & \boldsymbol{w}^T\boldsymbol{x} + b \leq 0 \end{cases}$$

We already encountered this problem when studying linear support vector machines. Visually, we can represent this classifier by a node that takes d inputs $(x_1, \ldots, x_d)$, and outputs 0 or 1:



Such a unit is called a **perceptron**. The node represents a **neuron** that *fires* if a certain linear combination of inputs, $\boldsymbol{w}^T\boldsymbol{x}$, exceeds a threshold $-b$. In order to determine the weights and bias term from data using optimization, it is useful to approximate the indicator with a smooth function such as the sigmoid

$$\sigma(x) = \frac{1}{1 + \mathrm{e}^{-x}}.$$

We can then replace the function $h_{\boldsymbol{w},b}$ with the smooth function $g(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b)$. A convenient property of the sigmoid function is that the derivative has the form

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)),$$

so that the gradient of $\boldsymbol{g}$ can be computed as

$$\nabla g(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b)(1 - \sigma(\boldsymbol{w}^T\boldsymbol{x} + b)) \cdot \boldsymbol{w}.$$

If we drop the requirement that the activator function should approximate the indicator function, then other popular activation functions are the hyperbolic tangent, $\tanh(x)$ (which maps into $[-1, 1]$), and the **rectifiable linear unit** (**ReLU**), $\max\{x, 0\}$ (which maps into $[0, \infty)$ and is not differentiable at 0). Figure 17.1 illustrates these different activations functions.

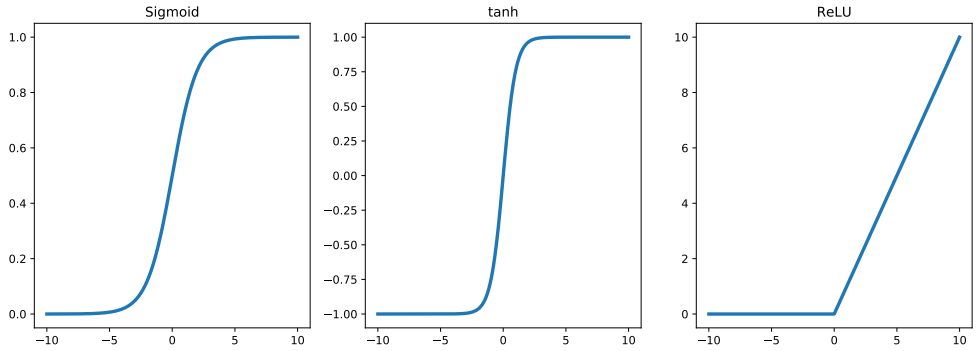


Figure 17.1: Activation functions

A **feedforward neural network** arises by combining various perceptrons by feeding the outputs of a series of perceptrons into new perceptrons, see Figure 17.2.

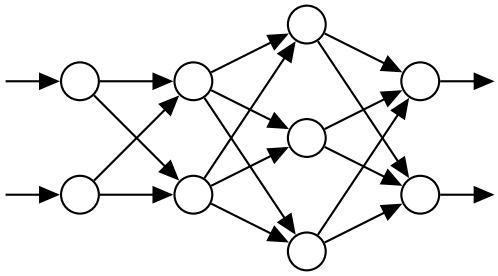


Figure 17.2: A fully connected neural network.

We interpret a neural network as consisting of different **layers**. To the k-th layer we associated a linear map $\boldsymbol{W}^k \colon \mathbb{R}^{d_{k-1}} \to \mathbb{R}^{d_k}$ and a bias vector $\boldsymbol{b}^k \in \mathbb{R}^{d_k}$. Applying the activation function componentwise, we obtain a map

$$\sigma(\boldsymbol{W}^k \boldsymbol{x} + \boldsymbol{b}^k).$$

The first layer is the **input layer**, while the last layer is the **output layer**. Hence, a neural network with ℓ layers is a function of the form $F^\ell(\boldsymbol{x})$, where the F^k are recursively defined as

$$\begin{aligned} F^1(\boldsymbol{x}) &= \sigma(\boldsymbol{W}^1 \boldsymbol{x} + \boldsymbol{b}^1) \\ F^{k+1}(\boldsymbol{x}) &= \sigma(\boldsymbol{W}^{(k+1)} F^k(\boldsymbol{x}) + \boldsymbol{b}^{k+1}), \quad 1 \leq k < \ell, \end{aligned}$$

and $\boldsymbol{W}^k \in \mathbb{R}^{d_k \times d_{k-1}}$, $\boldsymbol{b}^k \in \mathbb{R}^{d_k}$ for $1 \leq k \leq \ell$ (with $d_0 = d$). The layers between the input and output layer are called the **hidden layers**. If we fix the **format**, that is, the vector $\boldsymbol{d} = (d_0, d_1, \ldots, d_\ell)$, where d_i represents the number of nodes in each layer, we get a hypothesis class

$$\mathcal{H}_{\boldsymbol{d}} = \{F \colon \mathbb{R}^d \to \mathbb{R}^{d_\ell} \colon F \text{ is a NN with format } \boldsymbol{d}\}.$$

A neural network $F \in \mathcal{H}_{\boldsymbol{d}}$ can be used to approximate a function $\mathbb{R}^d \to \mathbb{R}^{d_\ell}$, or it can be used for classification tasks, for example by assigning an input $\boldsymbol{x}$ to the class represented by the index of the largest

coordinate of the output, $\arg\max_i F_i(\boldsymbol{x})$. Sometimes, the output is processed through an additional function.

If every node connects to a node in the next layer, then the network is called **fully connected**. We can place further restrictions on the form of individual linear maps at each layer (for example, by requiring them to perform a convolution on subsets of the input) and use different activation functions in each layer. Such a set of specifications defines an **architecture**.

Example 17.1. An elementary function that can be realized by a neural network is the **exclusive or**, or **XOR** function, which can be interpreted as addition in $\mathbb{Z}/2\mathbb{Z}$:

$$\mathrm{XOR}(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} = (1,0) \text{ or } \boldsymbol{x} = (0,1), \\ 0 & \text{if } \boldsymbol{x} = (0,0) \text{ or } \boldsymbol{x} = (1,1) \end{cases}$$

Thinking of the possible inputs as the corners of the unit square $[0,1]^2$ in $\mathbb{R}^2$, one easily verifies that the XOR function cannot be realized by a linear separator: there is no linear function h such that $h(\boldsymbol{x}) > 0$ if $\mathrm{XOR}(\boldsymbol{x}) = 1$ and $h(\boldsymbol{x}) < 0$ if $\mathrm{XOR}(\boldsymbol{x}) = 0$, for $\boldsymbol{x} \in \{0,1\}^2$. One can, however, implement XOR using a fully-connected neural network as

$$\mathrm{XOR}(\boldsymbol{x}) := \boldsymbol{w}^T([\boldsymbol{U}\boldsymbol{x} + \boldsymbol{b}]_+),$$

where

$$\boldsymbol{U} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \qquad \boldsymbol{b} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \qquad \boldsymbol{w} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}.$$

and $[x]_+ = \max\{x, 0\}$ is the ReLU activation function. One easily verifies that $\mathrm{XOR}(\boldsymbol{x}) = 1$ if exactly one of the coordinates of $\boldsymbol{x}$ is 1, and $\mathrm{XOR}(\boldsymbol{x}) = 0$ else.

We can train a neural network on data $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, $i \in \{1, \ldots, n\}$, by minimizing the empirical risk with respect to our favourite (or most suitable) loss function L. A common loss function is the square loss,

$$L(F(\boldsymbol{x}), \boldsymbol{y}) = \frac{1}{2}\|F(\boldsymbol{x}) - \boldsymbol{y}\|^2,$$

where we use the ℓ_2-norm (or Euclidean norm). If the neural network has only one layer and no activation function, then training it with respect to the square loss amounts to solving a least squares problem. Another common option that is used for classification problems with k classes is the **cross-entropy**, or negative **log-likelihood**. We assume that the training data is given by input-output pairs $(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{y} \in \mathbb{R}^k$ is a vector with $y_j = 1$ if $\boldsymbol{x}$ belongs to class j, and 0 else. The output of a neural network for this classification problem is then often a probability distribution, $F(\boldsymbol{x}) = (p_1, \ldots, p_k)$, where p_j is the probability that $\boldsymbol{x}$ belongs to class j. The cross-entropy is defined as

$$L(F(\boldsymbol{x}), y) = -\sum_k \left(y_k \log(p_k) + (1 - y_k)\log(1 - p_k)\right).$$

If the output consists of a vector $F(\boldsymbol{x}) = (v_1, \ldots, v_k)$ instead of a probability distribution, it is often transformed into a probability distribution by setting

$$p_i = \frac{\mathrm{e}^{v_i}}{\sum_j \mathrm{e}^{v_j}}.$$

For the rest of this chapter we will not make use of the specific form of the loss function, and we will get back to it when discussing information theory and applications.

Denote by $\boldsymbol{W}$ and $\boldsymbol{b}$ the concatenation of all the weight matrices and bias vectors. We denote by w_{ij}^k the (i,j)-th entry of the k-th matrix, and by b_i^k the i-th entry of the k-th bias vector. The task is then to minimize the empirical risk

$$f(\boldsymbol{W}, \boldsymbol{b}) := \frac{1}{n} \sum_{i=1}^{n} L(F^\ell(\boldsymbol{x}_i), \boldsymbol{y}_i).$$

If L is differentiable almost everywhere, then the method of choice is gradient descent, using a computational implementation of the chain rule known as **backpropagation**.

Backpropagation

Gradient descent, or stochastic gradient descent, requires evaluating the gradient of a function of the form

$$f_i(\boldsymbol{W}, \boldsymbol{b}) := L(F^\ell(\boldsymbol{x}_i), \boldsymbol{y}_i).$$

In what follows, set $\boldsymbol{x} = \boldsymbol{x}_i$ and $\boldsymbol{y} = \boldsymbol{y}_i$. Also write $\boldsymbol{a}^0 := \boldsymbol{x}$, and for $k \in \{1, \dots, \ell\}$,

$$\boldsymbol{z}^k = \boldsymbol{W}^k \boldsymbol{a}^{k-1} + \boldsymbol{b}^k, \quad \boldsymbol{a}^k = \sigma(\boldsymbol{z}^k). \tag{17.1}$$

In particular, $\boldsymbol{a}^\ell = F^\ell(\boldsymbol{x})$ is the output of the neural network on input $\boldsymbol{x}$. Moreover, set $C = C(\boldsymbol{W}, \boldsymbol{b}) = L(\boldsymbol{a}^\ell, \boldsymbol{y})$ for the loss function.

For every layer k and coordinate $j \in \{1, \dots, d_k\}$, define the sensitivities

$$\delta_j^k := \frac{\partial C}{\partial z_j^k},$$

where z_j^k is the j-th coordinate of $\boldsymbol{z}^k$. Thus δ_j^k measures the sensitivity of the loss function to the input at the j-th node of the k-th layer. Denote by $\boldsymbol{\delta}^k \in \mathbb{R}^{d_k}$ the vector of δ_j^k for $j \in \{1, \dots, d_k\}$. The partial derivatives of C can be computed in terms of these quantities. In what follows, we denote by $\boldsymbol{x} \circ \boldsymbol{y}$ the componentwise product, that is, the vector with entries $x_i y_i$.

Proposition 17.2. *For a neural network with ℓ layers and $k \in \{1, \dots, \ell\}$, we have*

$$\frac{\partial C}{\partial w_{ij}^k} = \delta_i^k a_j^{k-1}, \qquad \frac{\partial C}{\partial b_i^k} = \delta_i^k \tag{17.2}$$

for $i, j \in \{1, \dots, d_k\}$. Moreover, the sensitivities δ_i^k can be computed as follows:

$$\boldsymbol{\delta}^\ell = \sigma'(\boldsymbol{z}^\ell) \circ \nabla_{\boldsymbol{a}^\ell} L(\boldsymbol{a}^\ell, \boldsymbol{y}), \qquad \boldsymbol{\delta}^k = \sigma'(\boldsymbol{z}^k) \circ \left(\boldsymbol{W}^{k+1}\right)^T \boldsymbol{\delta}^{k+1} \tag{17.3}$$

for $k \in \{1, \dots, \ell - 1\}$.

Proof. We begin by showing (17.3). For $\boldsymbol{\delta}^\ell$, note that by the chain rule, we have

$$\delta_i^\ell = \frac{\partial L(\boldsymbol{a}^\ell, \boldsymbol{y})}{\partial z_i^\ell} = \sum_{j=1}^{d_\ell} \frac{\partial L(\boldsymbol{a}^\ell, \boldsymbol{y})}{\partial a_j^\ell} \frac{\partial a_j^\ell}{\partial z_i^\ell} = \frac{\partial L(\boldsymbol{a}^\ell, \boldsymbol{y})}{\partial a_i^\ell} \cdot \sigma'(z_i^\ell).$$

For $k < \ell$, we compute δ_i^k in terms of the δ_j^{k+1} as follows:

$$\delta_i^k = \frac{\partial C}{\partial z_i^k} = \sum_{j=1}^{d_{k+1}} \frac{\partial C}{\partial z_j^{k+1}} \frac{\partial z_j^{k+1}}{\partial z_i^k} = \sum_{j=1}^{d_{k+1}} \delta_j^{k+1} \cdot \frac{\partial z_j^{k+1}}{\partial z_i^k}.$$

For the summands in the last expression we use

$$z_j^{k+1} = \sum_{s=1}^{d_k} w_{js}^{k+1} \sigma(z_s^k) + b_j^{k+1},$$

so that the derivatives evaluate to

$$\frac{\partial z_j^{k+1}}{\partial z_i^k} = w_{ji}^{k+1} \cdot \sigma'(z_i^k).$$

Putting everything together, we arrive at

$$\delta_i^k = \sum_{j=1}^{d_{k+1}} \delta_j^{k+1} \cdot w_{ji}^{k+1} \cdot \sigma'(z_i^k) = \sigma'(z_i^k) \cdot \left((\boldsymbol{W}^{k+1})^T \boldsymbol{\delta}^{k+1} \right)_i .$$

The expressions for the partial derivatives of C with respect to the weights $\boldsymbol{W}$ and the bias $\boldsymbol{b}$ are computed in a straight-forward way using the chain rule. More precisely, at the k-th layer write

$$z_i^k = \sum_{j=1}^{d_{k-1}} w_{ij}^k a_j^{k-1} + b_i^k.$$

The claimed expressions for the derivatives then follow by applying the chain rule,

$$\frac{\partial C}{\partial w_{ij}^k} = \frac{\partial C}{\partial z_i^k} \frac{\partial z_i^k}{\partial w_{ij}^k} = \delta_i^k \cdot a_j^{k-1},$$

and similarly for the derivative with respect to b_i^k. □

For the common quadratic loss function

$$L(\boldsymbol{a}^\ell, \boldsymbol{y}) = \frac{1}{2} \|\boldsymbol{a}^\ell - \boldsymbol{y}\|^2,$$

we get

$$\nabla_{\boldsymbol{a}^\ell} L(\boldsymbol{a}^\ell, \boldsymbol{y}) = \boldsymbol{a}^\ell - \boldsymbol{y},$$

which can be computed easily from the function value $F^\ell(\boldsymbol{x})$ and $\boldsymbol{y}$. Other differentiable loss functions may lead to different terms. Given initial weights $\boldsymbol{W}$ and bias terms $\boldsymbol{b}$, we can compute the values $\boldsymbol{a}^k$ and $\boldsymbol{z}^k$ using a **forward pass**, that is, by applying (17.1) recursively. We can then compute the sensitivities δ_j^k using (17.3), and the partial derivatives of the loss function using (17.2), starting at layer ℓ. This way of computing the gradients is called **backpropagation**. We note that choosing the sigmoid σ as activation function, the computation of the derivative $\sigma'(x)$ is easy. The whole process of computing the gradient of a neural network thus reduces to a simple sequence of matrix-vector product operations and sigmoids.

Example 17.3. Consider the following setting with $n = 10$ points. We train a neural network with four layers and format $\boldsymbol{d} = (d_0, d_1, d_2, d_3) = (2, 2, 3, 2)$ using stochastic gradient descent. The neural network outputs a vector in $\mathbb{R}^2$ and classifies an input point according to whether the first coordinate is greater or smaller than the second coordinate. Figure 17.3 shows the decision boundary by the neural network based on 10 training points, and the display on the right shows the error per iteration of stochastic gradient descent.

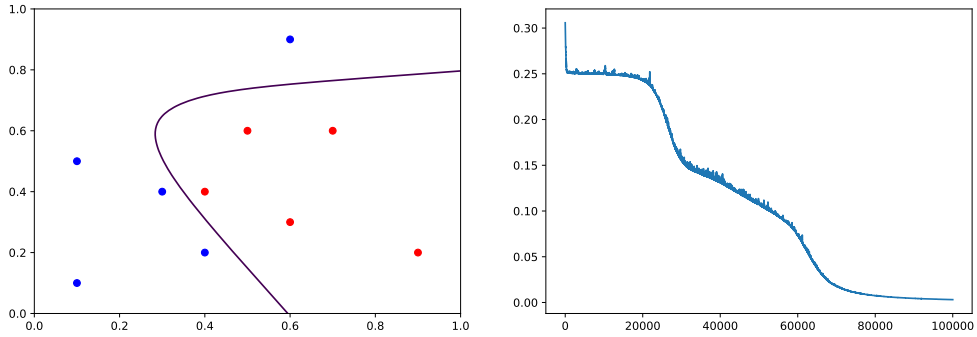


Figure 17.3: Training a neural network for classification and the error of stochastic gradient descent.

Notes

The study of neural networks has its origin in pioneering work by McCulloch and Pitts in the 1940s. Another seminal work in the history of artificial neural networks is the work by Rosenblatt on the Perceptron [64]. The terminology surrounding neural networks is not always consistent. Neural networks, as introduced here, are often referred to as *artificial* neural networks (ANN), to distinguish them from their biological counterparts. A feedforward neural network is also often called a multilayer perceptron (MLP). The idea of backpropagation was explicitly named in [66] and is closely related to **automatic differentiation**. Automatic differentiation has been discovered independently many times, and an interesting overview is given in [33]. Modern machine learning engines such as TensorFlow (`http://www.tensorflow.org`) or PyTorch (`http://www.pytorch.org`) are based on automatic differentiation engines. The content of this chapter is based on the excellent tutorial [37]. A comprehensive modern treatment of the subject can be found in the book [29].

18

Universal Approximation

An important feature of neural networks is their expressive power. In addition to being able to perform complex classification tasks, neural networks have the ability to approximate continuous functions to arbitrary accuracy.

Approximation in one dimension

The idea of approximation is best illustrated in one dimension. Given a function $f \in C([0,1])$, is it possible to find, for every $\epsilon > 0$, a neural network F that approximates f to accuracy ϵ, in the sense that $\|f - F\|_\infty = \sup_{x\in[0,1]} |f(x) - F(x)| < \epsilon$?

To answer this question, we begin by noting that any continuous function on $[0,1]$ can be approximated by step functions, as shown in Figure 18.1

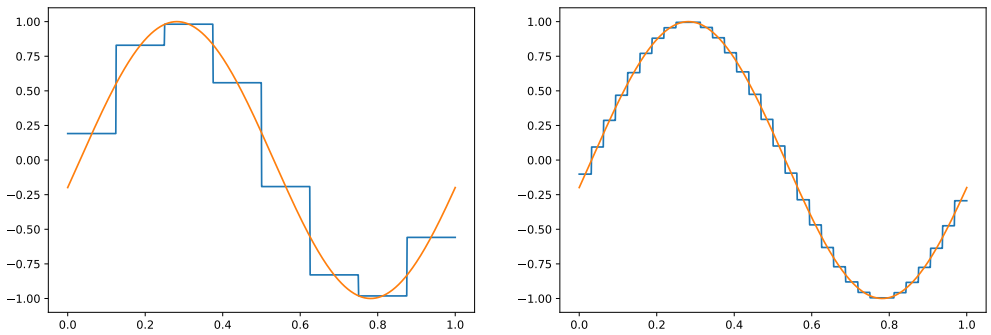


Figure 18.1: Approximation of a continuous function by a step function.

We next observe that any step function can be approximated by a combination of sigmoid functions of the form

$$\sigma(wx + b) = \frac{1}{1 + \mathrm{e}^{-(wx+b)}}.$$

It is convenient to parametrize such as sigmoid in terms of the location where $\sigma(wx + b) = 1/2$, which is given by $x = s := -b/w$. Thus for a given weight w and location s, the function

$$g_{w,s}(x) = \sigma(w(x - s))$$

approximates the step function from 0 to 1 at location s. The approximation improves when increasing w, as seen in Figure 18.2.

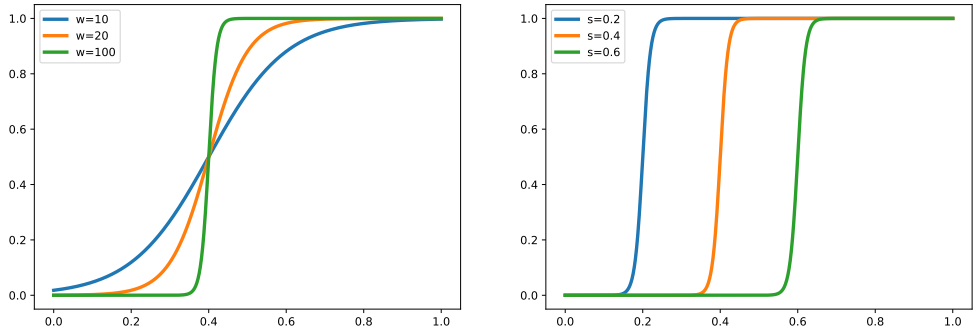


Figure 18.2: Sigmoid approximating a step function at different locations and with different weights.

We can now form linear combinations of such functions to approximate step functions (piecewise constant functions) with multiple steps, as shown in Figure 18.3.

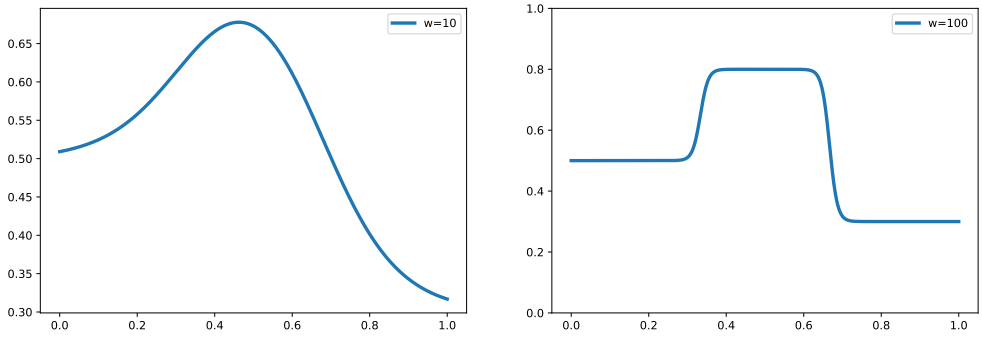


Figure 18.3: The function $0.5g_{w,-0.167} + 0.3g_{w,0.33} - 0.5g_{w,0.67}$ with weights $w = 10$ and $w = 100$.

We can therefore combine the approximation of the step function by sigmoids, and the approximation of an arbitrary continuous function on a closed interval by step functions, to approximate any continuous function on a closed interval by sigmoids; ee Figure 18.4 for an example. There may be some ambiguities on how to approximate a function by step functions. For example, one could take the maximum value, minimum value, or the function value at the midpoint of a function on an interval as the value of the piecewise constant function on that interval. Also, when approximating the piecewise constant functions by sigmoids, one has to take into account the approximation error at the left boundary, possibly by centering the first sigmoids at a negative value. Regardless of how one implements these considerations, in the end one arrives at a function of the form

$$\sum_{i=1}^{m} \alpha_i \sigma_i(w_i x + b_i), \tag{18.1}$$

and even though we have not given a formal proof, it is intuitively clear that such functions can approximate any continuous function on the unit interval to arbitrary accuracy. Such a function is essentially a neural network with one hidden layer, if we drop the sigmoid at the output layer, as shown in Figure 18.5.

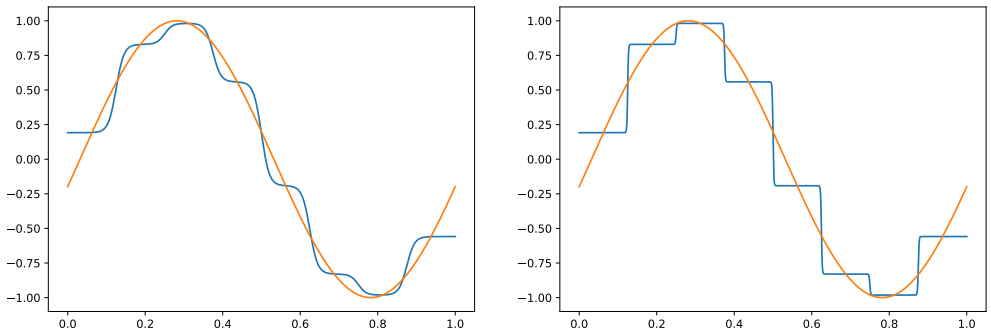


Figure 18.4: Approximation of a function by sigmoids with weights $w = 100$ and $w = 1000$.

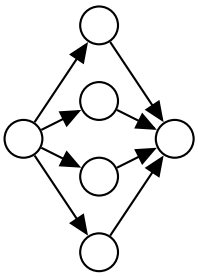


Figure 18.5: A neural network with one hidden layer.

Universal Approximation

The approximation of continuous functions on a closed interval by simple neural networks, which was derived heuristically in the previous section, generalizes to higher dimensions. This is the content of Cybenko's celebrated Universal Approximation Theorem. Let $I^d = [0,1]^d \subset \mathbb{R}^d$ denote the unit cube and let $C(I^d) \subset L^\infty(I^d)$ be the normed vector space of continuous functions on I^d with the norm $\|f\|_\infty = \sup_{\boldsymbol{x} \in I^d} |f(\boldsymbol{x})|$.

Theorem 18.1. *(Cybenko's Universal Approximation Theorem) Let $f \in C(I^d)$. For any $\epsilon > 0$ there exists a function g of the form*

$$g(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i \sigma(\boldsymbol{w}_i^T \boldsymbol{x} + b_i),$$

with weights $\boldsymbol{w}_i \in \mathbb{R}^d$ and $\alpha_i, b_i \in \mathbb{R}$, such that

$$\|f - g\|_\infty < \epsilon$$

The theorem states that the linear subspace of $C(I^d)$ spanned by functions of the form $\sigma(\boldsymbol{w}^T \boldsymbol{x} + b)$ is dense in $C(I^d)$ with respect to the norm $\|\cdot\|_\infty$. The proof is an immediate consequence of some standard results in Measure Theory and Functional Analysis, which we state here[1]. In what follows, if $S \subset X$ is a subset of a normed space, we denote by $\overline{S}$ the *closure* of S in X, i.e., the set of all limit points of elements of S.

[1] If you took Measure Theory or Functional Analysis and ever wondered "what is this good for", here you go!

Theorem 18.2. *((Consequence of) Hahn-Banach) Let $\mathcal{X}$ be a normed vector space, $S \subset \mathcal{X}$ a linear subspace, and $\boldsymbol{x}_0 \in \mathcal{X}$. Then $\boldsymbol{x}_0 \in \overline{S}$ if and only if for all linear functionals L on $\mathcal{X}$, $L(f) = 0$ for all $f \in S$ implies $f(\boldsymbol{x}_0) = 0$.*

In the following theorem, a *signed measure* is defined just like a measure, but without the non-negativity requirement.

Theorem 18.3. *(Riesz Representation Theorem) Let L be a bounded linear functional on $C(I^d)$. Then there exists a signed measure μ on I^d such that for every $f \in C(I^d)$,*

$$L(f) = \int_{I^d} f \, \mathrm{d}\mu(\boldsymbol{x}).$$

In addition to the Hahn-Banach and Riesz Representation Theorem, we need to know that the sigmoid function $\sigma(x)$ has the property that it is *discriminatory*.

Lemma 18.4. *Let $\sigma \in C(\mathbb{R})$ be a continuous function with values in $[0, 1]$, such that $\lim_{x\to\infty} \sigma(x) = 1$ and $\lim_{x\to-\infty} \sigma(x) = 0$. If μ is a signed measure on I^d, and if for every $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ we have*

$$\int_{I^d} \sigma(\boldsymbol{w}^T\boldsymbol{x} + b) \, \mathrm{d}\mu(\boldsymbol{x}) = 0,$$

then $\mu = 0$.

The proof relies on a Fourier Transform argument, which is stated without proof.

Lemma 18.5. *Let μ be a signed measure on I^d. Then*

$$\hat{\mu}(\boldsymbol{w}) := \int_{I^d} \mathrm{e}^{i\boldsymbol{w}^T\boldsymbol{x}} \, \mathrm{d}\mu(\boldsymbol{x}) = 0$$

implies $\mu = 0$.

Proof of Lemma 18.4. Let μ be a signed measure with the stated property. Consider the function $\sigma(s(\boldsymbol{w}^T\boldsymbol{x} + b) + t)$. Then

$$\gamma_t(\boldsymbol{x}) := \lim_{s\to\infty} \sigma(s(\boldsymbol{w}^T\boldsymbol{x} + b) + t) = \begin{cases} 1 & \text{if } \boldsymbol{w}^T\boldsymbol{x} + b > 0 \\ \sigma(t) & \text{if } \boldsymbol{w}^T\boldsymbol{x} + b = 0 \\ 0 & \text{if } \boldsymbol{w}^T\boldsymbol{x} + b < 0 \end{cases}$$

By the Lebesgue Dominated Convergence Theorem, we get

$$\int_{I^d} \gamma_t(\boldsymbol{x}) \, \mathrm{d}\mu(\boldsymbol{x}) = \lim_{s\to\infty} \int_{I^d} \sigma(s(\boldsymbol{w}^T\boldsymbol{x} + b) + t) \, \mathrm{d}\mu(\boldsymbol{x}) = 0,$$

by assumption. But the first integral is

$$\int_{I^d} \gamma_t(\boldsymbol{x}) \, \mathrm{d}\mu(\boldsymbol{x}) = \int_{H_+} \mathrm{d}\mu(\boldsymbol{x}) + \sigma(t) \int_{H_0} \mathrm{d}\mu(\boldsymbol{x}),$$

where $H_+ = \{\boldsymbol{x} \colon \boldsymbol{w}^T\boldsymbol{x} + b > 0\}$ and $H_0 = \{\boldsymbol{x} \colon \boldsymbol{w}^T\boldsymbol{x} + b = 0\}$. Since this is the case for all t, and in particular for the limit as $t \to \infty$ and $t \to -\infty$, it follows that

$$\int_{H_+} \mathrm{d}\mu(\boldsymbol{x}) + \int_{H_0} \mathrm{d}\mu(\boldsymbol{x}) = 0 \quad \text{and} \quad \int_{H_+} \mathrm{d}\mu(\boldsymbol{x}) = 0. \tag{18.2}$$

This establishes that the measure of open and closed half-spaces is zero, but since we are dealing with signed measures, we cannot conclude that $\mu = 0$. To show this, we consider linear functionals on $L^\infty(\mathbb{R})$ given by

$$F(h) = \int_{I^d} h(\boldsymbol{w}^T \boldsymbol{x}) \, \mathrm{d}\mu(\boldsymbol{x}).$$

From (18.2) it follows that $F(h) = 0$ if h is the indicator function of open and closed intervals, and by linearity it follows that $F(h) = 0$ if h is a simple function. Since simple functions are dense in $L^\infty(\mathbb{R})$, it follows that $F = 0$. In particular,

$$0 = \int_{I^d} \mathrm{e}^{i\boldsymbol{w}^T \boldsymbol{x}} \, \mathrm{d}\mu(\boldsymbol{x}) = \hat{\mu}(\boldsymbol{w}),$$

and from Lemma 18.5 we conclude that $\mu = 0$. □

Proof of Theorem 18.1. Let S be the set of functions of the form

$$g(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i \sigma(\boldsymbol{w}_i^T \boldsymbol{x} + b_i)$$

on I^d. Clearly, S is a linear subspace of $C(I^d)$. We will show that S is dense in $C(I^d)$. Assume to the contrary that S is not dense, i.e., that the closure $\overline{S}$ of S is a proper linear subspace of $C(I^d)$. By the Hahn-Banach Theorem 18.2, there exists a non-zero linear functional L on $C(I^d)$ such that $L|_{\overline{S}} = 0$. By the Riesz Reprentation Theorem 18.3, there exists a regular Borel measure μ and a function h such that

$$L(f) = \int_{I^d} f(\boldsymbol{x}) \, \mathrm{d}\mu(\boldsymbol{x})$$

for all $f \in C(I^d)$. This means, however, that for any weights $\boldsymbol{w}$ and bias terms b we have

$$\int_{I^d} \sigma(\boldsymbol{w}^T \boldsymbol{x} + b) \, \mathrm{d}\mu(\boldsymbol{x}) = 0,$$

since $\sigma(\boldsymbol{w}^T \boldsymbol{x} + b) \in L$. By Lemma 18.4, it follows that $\mu = 0$ and hence $L = 0$, contradicting the fact that $L \neq 0$ established earlier. Hence, $\overline{S} = C(I^d)$. □

Note that even though the approximating neural network has only one hidden layer, there is no bound on the size of this layer: it could be arbitrary large. Intuitively, we would expected the complexity of the approximating network to depend on the complexity of the function that we would like to approximate.

Notes

The Universal Approximation Theorem was derived by Cybenko in [23]. Since then, the result has been generalized in several directions, among others also to different activation functions such as ReLU. An instructive visual overview of approximation in one dimension can be found in http://neuralnetworksanddeeplearning.com/chap4.html. A survey of expressivity results for neural networks is [34].

19

Convolutional Neural Networks

The neural networks considered so far are called **fully connected**, since each node in one layer is connected to each node in the following layer. A particularly useful and widespread architecture for image data is the class of **Convolutional Neural Networks**, or **CNN**. In practice, one often considers special types of connectivity. It is also common to use activation functions other than the sigmoid, one example being the Rectified Linear Unit (ReLU). In convolutional neural networks, many of the linear maps in each layer correspond to the mathematical operation of **convolution**. These are then usually combined with ReLU activation functions, pooling layers, and fully connected layers.

Convolutions

In the context of functions, the convolution of two real valued functions f and g is defined as the function

$$f * g(y) = \int_{-\infty}^{\infty} f(x) \cdot g(y - x) \, \mathrm{d}x.$$

In a discrete setting, given a vector $\boldsymbol{x} \in \mathbb{R}^d$ and a sequence $\boldsymbol{g} = (g_{-d+1}, \ldots, g_{d-1})^T$, the convolution is defined as the vector $\boldsymbol{y} = (y_1, \ldots, y_d)^T$, where

$$y_k = \sum_{i=1}^{d} x_i \cdot g_{k-i}, \quad k \in \{1, \ldots, d\}$$

This definition also applies to infinite sequences, but we will not use that here. In terms of matrices,

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_d \end{pmatrix} = \begin{pmatrix} g_0 & g_{-1} & \cdots & g_{-d+1} \\ g_1 & g_0 & \cdots & g_{-d+2} \\ \vdots & \vdots & \ddots & \vdots \\ g_{d-1} & g_{d-2} & \cdots & g_0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

The sequence $\boldsymbol{g}$ is called a **filter** in signal processing. Typically, multiplying a $d \times d$ matrix with a d-dimensional vector requires d^2 multiplications. If, however, the matrix has a special structure, then it is possible to perform this multiplication much faster, without having to explicitly represent the matrix as two-dimensional array in memory.

Example 19.1. The **cyclic convolution** corresponds to multiplication with a **circulant matrix**

$$\boldsymbol{C} = \begin{pmatrix} c_0 & c_1 & \cdots & c_{d-1} \\ c_{d-1} & c_0 & \cdots & c_{d-2} \\ \vdots & \vdots & \ddots & \vdots \\ c_1 & c_2 & \cdots & c_0 \end{pmatrix}$$

This is the same as the convolution with a sequence $\boldsymbol{g}$ as above, with $c_i = g_{-i}$ and $g_i = g_{d-1}$ for $i \in \{1, \ldots, d-1\}$. A special case of a circulant matrix is the **Discrete Fourier Transform** (DFT), in which

$$c_{jk} = \exp(2\pi i \cdot jk/n).$$

The DFT plays an important role in signal and image processing, and versions of it, such as the two-dimensional Discrete Cosine Transform, are the basis of the image compression standard JPEG. One of the most influential and widely used algorithms, the Fast Fourier Transform (FFT), computes a DFT with $O(d \log(d))$ operations, and this is essentially optimal. FFT-based algorithms can also be applied to carry out cyclic convolutions, since circulants are diagonalized by DFT matrices.

Example 19.2. In many applications, only a few of the entries of $\boldsymbol{g}$ will be non-zero. Perhaps the easiest and most illustrative example is the difference matrix

$$\boldsymbol{D} = \begin{pmatrix} -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \cdots & 0 \\ 0 & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -1 \end{pmatrix}.$$

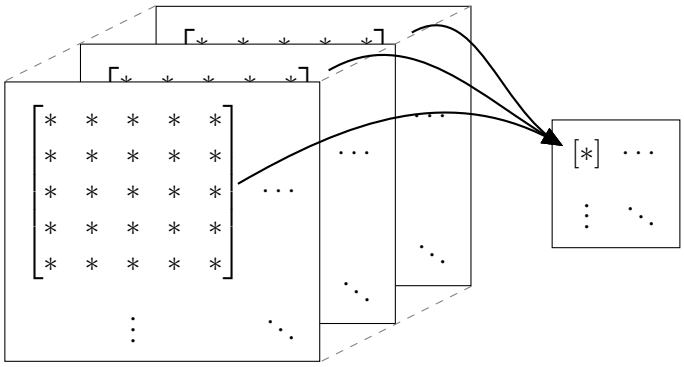
This matrix maps a vector $\boldsymbol{x}$ to the vector of differences $x_{i+1} - x_i$. This transformation detects *changes* in the vector, and a vector that is constant over a large range of coordinates is mapped to a **sparse** vector. Again, computing $\boldsymbol{y} = \boldsymbol{D}\boldsymbol{x}$ requires just $d-1$ subtractions and no multiplications, and it is not necessary to store $\boldsymbol{D}$ as matrix in memory.

Given a convolution $\boldsymbol{G}$, one often only considers a subset of the rows, such as every second or every third row. If only every k-th row is considered, then we say that the filter has **stride** length k. If the support (the number of non-zero entries) of $\boldsymbol{g}$ is small, then one can interpret the convolution operation as taking the inner product of $\boldsymbol{g}$ with a particular *section* of $\boldsymbol{x}$, and then moving this section, or "window", by k entries if the stride length is k. Each such operation can be seen as extracting certain information from a part of $\boldsymbol{x}$.

Convolution Layers and Image Data

In the context of colour image data, the vector $\boldsymbol{x}$ will not be seen as a single vector in some $\mathbb{R}^d$, but as a **tensor** with format $N \times M \times 3$. Here, the image is considered to be a $N \times M$ matrix, with each entry corresponding to a pixel, and each pixel is represented by a 3-tuple consisting of the red, blue and green components. A convolution filter will therefore operate on a subset of this tensor, typically a $n \times m \times 3$ window. Typical parameters are $N = M = 32$ and $n = m = 5$.

Each $5 \times 5 \times 3$ block is mapped linearly to an entry of a 28×28 matrix in the same way (that is, applying the same filter). The filter applied is called a **kernel** and the resulting map is interpreted as



representing a **feature**. Specifically, instead of writing it as a vector $\boldsymbol{g}$, we can write it as a matrix $\boldsymbol{K}$, and then have, if we denote the input image by $\boldsymbol{X}$,

$$\boldsymbol{Y} = \boldsymbol{X} * \boldsymbol{K}, \quad Y_{ij} = \sum_{k,\ell} X_{k\ell} \cdot K_{i-k,j-\ell}.$$

In a convolutional layer, various different kernels or convolutions are applied to the image. This can be seen as extracting several features from an image. Figure 19.1 shows and example from a seminal paper by Krizhevsky et. al. that classified the 1.2 million images from the ImageNet LSVRC-2010 database to record accuracy.

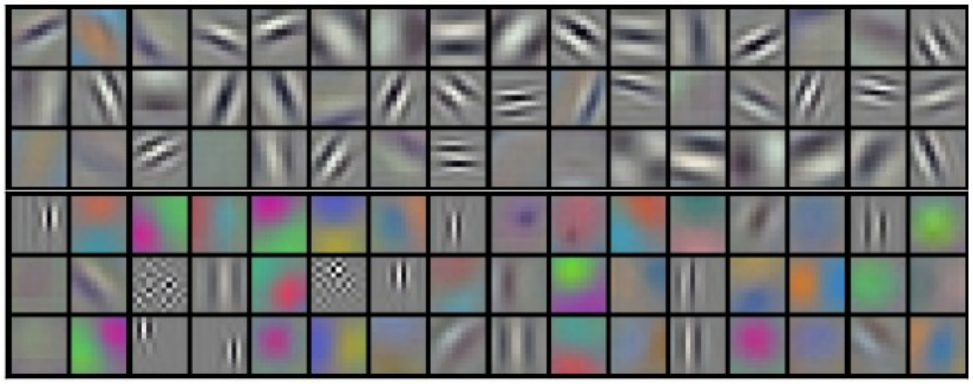


Figure 19.1: Filters for a convolutional layer from an image classification tasks, Krizhevsky et. al.

Convolutional networks are inspired by biology, where the brain scans different parts of an image and extracts certain types of structure from local patches, and then combines the pieces.

Softmax

For a classification problem with k classes, it is common to have k output and pick the class corresponding to the largest entry. A transformation that is often used on the output data is the softmax operation. Suppose the output consists of k entries $v_1, \ldots, v_k$. Then one computes vector with entries

$$-\log\left(\frac{\mathrm{e}^{v_i}}{\sum_{j=1}^{k} \mathrm{e}^{v_j}}\right).$$

Example

Convolutional layers are often combined with **pooling layers**. In a pooling layer, a few entries (typically a 4×4 submatrix) are combined into one entry, either my taking the maximum (max pooling) or by averaging. This operation reduces the size of the data. In addition, towards the end one often adds a fully connected layer. A prototypical example is the influential LeNet-5 architecture introduced by Yann LeCun and coauthors in 1998 and applied to digit recognition, see Figure 19.2. The following code shows an

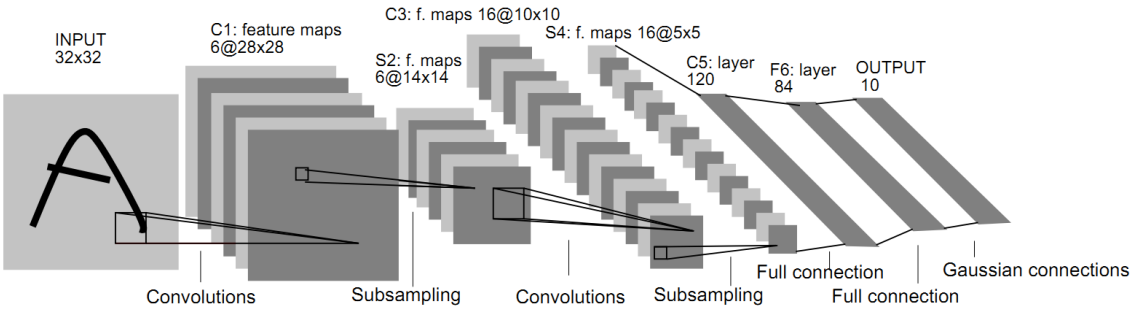


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Figure 19.2: Architecture of LeNet-5 from LeCunn et. al. (1998)

implementation of the LeNet-5 architecture in Python using TensorFlow and the Keras frontend.

In [1]:
```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# Adjust format to get 60000 x 28 x 28 x 1 tensor for training
#  and 10000 x 28 x 28 x 1 tensor for testing
X_train = X_train[..., np.newaxis].astype('float32') / 255
X_test = X_test[..., np.newaxis].astype('float32') / 255
# Change output data type to categorial / 10 classes
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
# Build up CNN according to LeNet-5
model = Sequential()
model.add(Conv2D(6, (5, 5), activation='relu', padding='same',
                 input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(16, (5, 5), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(120, (5, 5), activation='relu'))
model.add(Flatten())
model.add(Dense(84, activation='relu'))
model.add(Dense(10, activation='softmax'))
# Now the magic happens...
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
model.fit(X_train, y_train,
          batch_size=128,
          epochs=20,
          validation_split=0.10)
# Evaluate output
output = model.evaluate(X_test, y_test)
print('Accuracy: {0:.2f}'.format(output[1]))
```

Out [1]:
```
10000/10000 [==============================] - 3s 318us/step
Accuracy: 0.9855
```

Notes

There are various excellent sources and tutorials on convolutional neural networks. One example is Chapter 9 of [29] and Chapter 7 of the survey [37] which also contains a worked through example in MATLAB. Two seminal papers in the field are [48] and [46].

20

Sequence Models

Just as convolutional neural networks are designed for image classification tasks, **Recurrent Neural Networks** (**RNN**) are a class of neural networks designed to operate on sequences. The defining feature of a RNN is memory: when processing an element from a sequence, the network takes into account conclusions drawn from previous elements in the same sequence. Applications of recurrent neural networks include speech recognition, handwriting recognition, text-to-speech synthesis, machine translation, sentiment analysis, and generating literature or music.

Recurrence

Recurrent neural networks operate on sequences

$$\boldsymbol{X} = \{\boldsymbol{x}_t\}_{t\geq 1},$$

where $\boldsymbol{x}_t \in \mathbb{R}^m$ for $t \geq 1$. It is important to note that every sequence $\boldsymbol{X}$ is a data point in its own right, and not a collection of data points. Training data thus consists of a collection of sequences $\{\boldsymbol{X}_i\}_{i=1}^n$, and the network treats each such sequence as an individual input. In practice, such a sequence has finite length, but we do not need to bound the length explicitly here. The index t is suggestive of time, and will usually be referred to as such, but the data need not be a time series.

A neural network operating on sequences typically maps a sequence to another sequence, just as a CNN operates on "image-like" data during the convolutional stages. It does so by computing a linear map $\boldsymbol{X} \mapsto \boldsymbol{Y}$, followed by a non-linear activation function ρ. The first observation when dealing with long sequences is that each element of the input, $\boldsymbol{x}_t$, should be transformed into an element of the output, $\boldsymbol{y}_t = \rho(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{b})$, in exactly the same way. We can therefore write the linear map associated to one layer as follows:

$$\begin{bmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} \boldsymbol{W} & \boldsymbol{0} & \cdot \\ \boldsymbol{0} & \boldsymbol{W} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \end{bmatrix} + \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{b} \\ \vdots \end{bmatrix}.$$

The next important observation is that, while the absolute position t is not important, the *context* of an element and the *ordering* of elements is: the output $\boldsymbol{z}_t$ should depend not only on $\boldsymbol{x}_t$, but also on information extracted from previous elements of the sequence. Perhaps the easiest way of incorporating

previous information is by setting

$$\begin{aligned} \boldsymbol{z}_t &= \boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{V}\boldsymbol{z}_{t-1} + \boldsymbol{b}, \\ \boldsymbol{y}_t &= \rho(\boldsymbol{z}_t), \end{aligned} \tag{20.1}$$

for $t \in \{1, 2, \dots\}$, where ρ is an activation function. The first line in (20.1) is reminiscent of **vector autoregression** in time series analysis. In general, information obtained from earlier in the sequence is encoded in a **state variable** $\boldsymbol{h}_t$ that is computed from the outputs $\boldsymbol{y}_t$ and the previous state $\boldsymbol{h}_{t-1}$. A layer of a general RNN can thus be described as

$$\begin{aligned} \boldsymbol{z}_t &= \boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{V}\boldsymbol{h}_{t-1} + \boldsymbol{b}, \\ \boldsymbol{y}_t &= \rho(\boldsymbol{z}_t), \\ \boldsymbol{h}_t &= f(\boldsymbol{y}_t, \boldsymbol{h}_{t-1}), \end{aligned} \tag{20.2}$$

where f is a function. Figure 20.1 shows two ways of visualising a layer of an RNN.

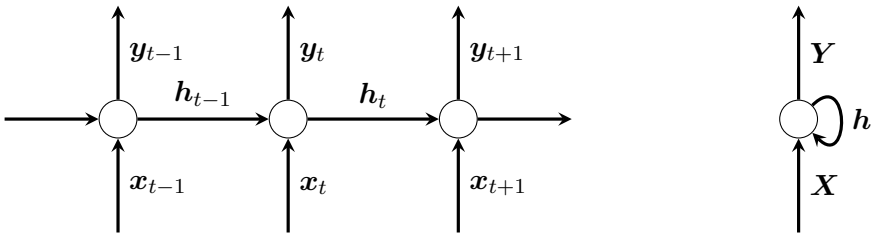


Figure 20.1: A layer in a recurrent neural network, in unfolded form and with a loop.

The output of an RNN could consist of a sequence (for example, in machine translation), an element of a sequence, or a probability vector for classification (for example, in sentiment analysis). The structure of an RNN thus consists of one or more layers of the form (20.2), possibly followed by an output function. RNN layers are typically combined with other layers, such as feed-forward or convolutional layers, and the nodes in Figure 20.1 can also be replaced by multi-layer networks, leading to more expressive architectures.

Example 20.1. A simple example that illustrates how a recurrent neural network works is addition. We consider the special case of binary addition of sequences over the alphabet $\{0, 1\}$. For example, $01011 + 01101 = 11000$. The input data is coded as a sequence of column vectors of size 2, and the output consists of a single sequence:

$$\left[\begin{array}{c|c|c|c|c} 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \end{array}\right] \quad \Rightarrow \quad 11000$$

We take the right-most column/number as the first element of the sequence. Adding two digits corresponds to the XOR (exclusive or) operation, which can be implemented by a fully-connected neural network as

$$\mathrm{XOR}(\boldsymbol{x}) := \boldsymbol{w}^T([\boldsymbol{U}\boldsymbol{x} + \boldsymbol{b}]_+),$$

where

$$\boldsymbol{U} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \qquad \boldsymbol{b} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \qquad \boldsymbol{w} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}.$$

and $[x]_+ = \max\{x, 0\}$ is the ReLU activation function. One easily verifies that $\mathrm{XOR}(\boldsymbol{x}) = 1$ if exactly one of the coordinates of $\boldsymbol{x}$ is 1, and $\mathrm{XOR}(\boldsymbol{x}) = 0$ else. In order to add two sequences, when processing

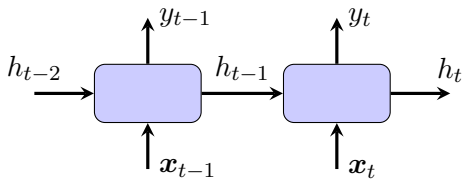


Figure 20.2: Binary addition. Each element $\boldsymbol{x}_t$ of the input sequence consists of two coordinates that are added together, and h_t denotes the carry bit from the previous addition. Thus $h_t = 1$ if and only if $\boldsymbol{x}_{t-1} = (1, 1)^T$ or one of the coordinates of $\boldsymbol{x}_{t-1}$ is 1 and $h_{t-1} = 1$.

each element the network needs to *remember* whether the addition of the previous digits produced a carry bit. Schematically, we can can visualize such as network as in Figure 20.2. We can consider the boxes in Figure 20.2 as black boxes, each taking the two input bits and the carry bit and outputting the binary sum. Such a black box can itself consist of more than one layer. In our case, one way of implementing such a black box would be

$$\begin{aligned}
\boldsymbol{c}_t &= [\boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{b}]_+ \\
\boldsymbol{a}_t &= [\boldsymbol{W}\boldsymbol{c}_t + \boldsymbol{V}h_{t-1} + \boldsymbol{d}]_+ \\
y_t &= \boldsymbol{w}^T\boldsymbol{a}_t \\
h_t &= \boldsymbol{e}_2^T(\boldsymbol{a}_t + \boldsymbol{c}_t),
\end{aligned}$$

where $\boldsymbol{U}, \boldsymbol{w}$ and $\boldsymbol{b}$ are as in the XOR implementation, and

$$\boldsymbol{W} = \begin{pmatrix} 1 & -2 \\ 1 & -2 \end{pmatrix}, \qquad \boldsymbol{V} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad \boldsymbol{e}_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \qquad \boldsymbol{d} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}.$$

Each box thus consists of one non-recurrent ReLU layer that computes $\boldsymbol{c}_t$ and one recurrent layer that computes the output y_t and the carry h_t for the next element.

Backpropagation through time

For training a RNN, we require a version of backpropagation that takes into account the time dependence. The resulting modification of backpropagation is called **backpropagation through time** (**BPTT**).

For simplicity we consider a single layer RNN of the form (20.2). Backpropagation Through Time then follows simply by "unrolling" the RNN and interpreting each time t as a layer of a deep feedforward network. Let $F(\boldsymbol{X}; \boldsymbol{W}, \boldsymbol{V}, \boldsymbol{b})$ be the sequence computed by the RNN on input $\boldsymbol{X}$. Let

$$C = C(\boldsymbol{W}, \boldsymbol{V}, \boldsymbol{b}) := L(F(\boldsymbol{X}'), \boldsymbol{Y}')$$

be the value of a suitable loss function on a fixed pair $(\boldsymbol{X}', \boldsymbol{Y}')$ if input and output sequences from the training data. Write

$$\boldsymbol{\delta}_t := \nabla_{\boldsymbol{z}_t} C$$

for the sensitivity of the objective function to the output of the linear transformation at time t. The cost function itself will usually be of the form

$$C = \sum_{t \geq 1} C_t, \qquad C_t := L(\boldsymbol{y}_t, \boldsymbol{y}_t'),$$

where each $\boldsymbol{y}_t = F(\boldsymbol{X}')_t$. Each C_t only consists of information up to time t. In what follows we will work with C, but the same reasoning applies for any C_t. From (20.2) we get the following characterization of the gradients of C.

Lemma 20.2. *The gradients of the cost function C with respect to $\boldsymbol{W}$, $\boldsymbol{V}$, and $\boldsymbol{b}$ are*

$$\nabla_{\boldsymbol{W}} C = \sum_{t\geq 1} \boldsymbol{\delta}_t \cdot \boldsymbol{x}_t^\intercal, \quad \nabla_{\boldsymbol{V}} C = \sum_{t\geq 1} \boldsymbol{\delta}_t \cdot \boldsymbol{h}_{t-1}^\intercal, \quad \nabla_{\boldsymbol{b}} C = \sum_{t\geq 1} \boldsymbol{\delta}_t,$$

where $\nabla_{\boldsymbol{W}} C$ is the matrix with entries $\partial C/\partial w_{ij}$ (and similar for $\nabla_{\boldsymbol{V}} C$).

Proof. We derive the expression for $\nabla_{\boldsymbol{W}} C$, the other cases are similar. The weights $\boldsymbol{W}$ occur within C in the form $\boldsymbol{a}_t := \boldsymbol{W}\boldsymbol{x}_t$. Hence,

$$\frac{\partial C}{\partial w_{ij}} = \sum_{t\geq 1} \langle \nabla_{\boldsymbol{a}_t} C, \frac{\partial \boldsymbol{a}_t}{\partial w_{ij}} \rangle = \sum_{t\geq 1} (\nabla_{\boldsymbol{a}_t} C)_i \cdot x_j.$$

Using the chain rule and the fact that $\boldsymbol{z}_t = \boldsymbol{a}_t + \boldsymbol{V}\boldsymbol{h}_{t-1} + \boldsymbol{b}$, we get

$$\nabla_{\boldsymbol{a}_t} C = \nabla_{\boldsymbol{z}_t} C = \boldsymbol{\delta}_t.$$

The claim follows. □

We have thus reduced the gradient computation to the computation of the sensitivities $\boldsymbol{\delta}_t$. As in the case of feedforward neural networks, backpropagation consists of computing the $\boldsymbol{\delta}_t$ recursively, only that here we do this along time instead of depth.

Lemma 20.3. *For a sequence of length T, the sensitivities $\boldsymbol{\delta}_t$ satisfy*

$$\boldsymbol{\delta}_T = \rho'(\boldsymbol{z}_T) \odot \nabla_{\boldsymbol{y}_T} L(\boldsymbol{Y}, \boldsymbol{Y}')$$

and for $t \in \{2, \ldots, T\}$,

$$\boldsymbol{\delta}_{t-1} = \rho'(\boldsymbol{z}_{t-1}) \odot \nabla_{\boldsymbol{y}_{t-1}} f(\boldsymbol{y}_{t-1}, \boldsymbol{h}_{t-2}) \cdot \boldsymbol{V}^T \boldsymbol{\delta}_t,$$

where $\boldsymbol{h}_0$ is the initial state.

Proof. For the partial derivatives with respect to $\boldsymbol{z}_t$ and $\boldsymbol{y}_t$ we get

$$\boldsymbol{\delta}_t = \rho'(\boldsymbol{z}_t) \odot \nabla_{\boldsymbol{y}_t} C, \qquad \nabla_{\boldsymbol{y}_t} C = \nabla_{\boldsymbol{y}_t} \boldsymbol{h}_t \cdot \nabla_{\boldsymbol{h}_t} C = \nabla_{\boldsymbol{y}_t} f(\boldsymbol{y}_t, \boldsymbol{h}_{t-1}) \cdot \nabla_{\boldsymbol{h}_t} C, \tag{20.3}$$

where by $\nabla_{\boldsymbol{y}_t} \boldsymbol{h}_t$ we denote the matrix with ij-th entry $\partial(\boldsymbol{h}_t)_j / \partial(\boldsymbol{y}_t)_i$. From the expression (20.2) we see that

$$\nabla_{\boldsymbol{h}_{t-1}} C = \nabla_{\boldsymbol{h}_{t-1}} \boldsymbol{z}_t \cdot \nabla_{\boldsymbol{z}_t} C = \boldsymbol{V}^T \boldsymbol{\delta}_t.$$

Putting everything together, we get for $t \in \{2, \ldots, T\}$,

$$\begin{aligned} \boldsymbol{\delta}_{t-1} &= \rho'(\boldsymbol{z}_{t-1}) \odot \nabla_{\boldsymbol{y}_{t-1}} C \\ &= \rho'(\boldsymbol{z}_{t-1}) \odot \nabla_{\boldsymbol{y}_{t-1}} f(\boldsymbol{y}_{t-1}, \boldsymbol{h}_{t-2}) \cdot \nabla_{\boldsymbol{h}_{t-1}} C \\ &= \rho'(\boldsymbol{z}_{t-1}) \odot \nabla_{\boldsymbol{y}_{t-1}} f(\boldsymbol{y}_{t-1}, \boldsymbol{h}_{t-2}) \cdot \boldsymbol{V}^T \boldsymbol{\delta}_t. \end{aligned}$$

The expression for $\boldsymbol{\delta}_T$ follows from the first equation in (20.3). □

input : RNN parameters $\boldsymbol{W}$, $\boldsymbol{V}$, $\boldsymbol{b}$
output : Gradients $\nabla_{\boldsymbol{W}} C$, $\nabla_{\boldsymbol{V}} C$, $\nabla_{\boldsymbol{b}} C$
for t *from* 1 *to* T
  $\boldsymbol{z}_t \leftarrow \boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{V}\boldsymbol{h}_{t-1} + \boldsymbol{b}$;
  $\boldsymbol{y}_t \leftarrow \rho(\boldsymbol{z}_t)$;
  $\boldsymbol{h}_t \leftarrow f(\boldsymbol{y}_t, \boldsymbol{h}_{t-1})$;
$\boldsymbol{\delta}_T \leftarrow \rho'(\boldsymbol{z}_T) \odot \nabla_{\boldsymbol{y}_T} C$;
$\nabla_{\boldsymbol{W}} C \leftarrow \boldsymbol{0}$, $\nabla_{\boldsymbol{V}} C \leftarrow \boldsymbol{0}$, $\nabla_{\boldsymbol{b}} C \leftarrow \boldsymbol{0}$;
for t *from* T *to* 2
  $\nabla_{\boldsymbol{W}} C \leftarrow \nabla_{\boldsymbol{W}} C + \boldsymbol{\delta}_t \cdot \boldsymbol{x}_t^\intercal$;
  $\nabla_{\boldsymbol{V}} C \leftarrow \nabla_{\boldsymbol{V}} C + \boldsymbol{\delta}_t \cdot \boldsymbol{h}_{t-1}^\intercal$;
  $\nabla_{\boldsymbol{b}} C \leftarrow \nabla_{\boldsymbol{b}} C + \boldsymbol{\delta}_t$;
  $\boldsymbol{\delta}_{t-1} \leftarrow \rho'(\boldsymbol{z}_{t-1}) \odot \nabla_{\boldsymbol{y}_{t-1}} f(\boldsymbol{y}_{t-1}, \boldsymbol{h}_{t-2}) \cdot \boldsymbol{V}^\intercal \boldsymbol{\delta}_t$;
$\nabla_{\boldsymbol{W}} C \leftarrow \nabla_{\boldsymbol{W}} C + \boldsymbol{\delta}_1 \cdot \boldsymbol{x}_1^\intercal$;
$\nabla_{\boldsymbol{V}} C \leftarrow \nabla_{\boldsymbol{V}} C + \boldsymbol{\delta}_1 \cdot \boldsymbol{h}_0^\intercal$;
$\nabla_{\boldsymbol{b}} C \leftarrow \nabla_{\boldsymbol{b}} C + \boldsymbol{\delta}_1$;

Algorithm 1: Gradient computation in a RNN.

Algorithm 1 describes describes an implementation of BPTT for computing a single gradient. This algorithm can be used as as part of a stochastic gradient descent implementation to train a RNN. Note that BPTT is easily extended to multilayer RNNs. Suppose we have ℓ layers, and $\boldsymbol{a}_t^k$ denotes the output at time t and layer k, with $\boldsymbol{a}_t^0$ being the input sequence and $\boldsymbol{y}_t = \boldsymbol{a}^\ell$ the output. Let $\boldsymbol{W}^k$, $\boldsymbol{V}^k$, $\boldsymbol{b}^k$ denote the parameters at layer k and $\boldsymbol{\delta}_t^k = \partial C / \partial \boldsymbol{z}_t^k$ the sensitivity with respect to time t and layer k. Then

$$\boldsymbol{\delta}_T^\ell = \rho'(\boldsymbol{z}_T^\ell) \odot \nabla_{\boldsymbol{y}_T} L(\boldsymbol{Y}, \boldsymbol{Y}'), \qquad \boldsymbol{\delta}_t^{k-1} = \rho'(\boldsymbol{z}_t^{k-1}) \odot \boldsymbol{W}^k \boldsymbol{\delta}_t^k.$$

Starting with $\boldsymbol{\delta}_T^\ell$, which is computed from a forward pass, we obtain the $\boldsymbol{\delta}_t^k$ for $k < \ell$ and $t < T$ by horizontal backpropagation (for t) and vertical backpropagation (for k).

The vanishing gradient problem

While superficially BPTT does not differ much from traditional backpropagation, the length of sequences can be significantly larger than the typical depth of a feedforward neural network, and this can cause problems that do not surface in non-recurrent networks. The most obvious issue is computational complexity. A more subtle issue is the **vanishing gradient problem**, and somewhat related, the **exploding gradient problem**. Consider, for example, the simple version of a simple one-layer RNN with $f(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}$, that is,

$$\begin{aligned} \boldsymbol{z}_t &= \boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{V}\boldsymbol{x}_{t-1} + \boldsymbol{b}, \\ \boldsymbol{y}_t &= \rho(\boldsymbol{z}_t). \end{aligned}$$

The backpropagation equations read as

$$\boldsymbol{\delta}_t = \rho'(\boldsymbol{z}_t) \odot \boldsymbol{V}^\intercal \boldsymbol{\delta}_{t+1} = \left(\prod_{i=1}^{T-t} \operatorname{diag}(\rho'(\boldsymbol{z}_{t+i-1})) \boldsymbol{V}^\intercal \right) \boldsymbol{\delta}_T.$$

One problem with activation functions ρ such as the logistic sigmoid and $\tanh$ is that the derivative ρ' can become arbitrary small for large arguments (this problem does not occur when using ReLU). But even for small values of $\boldsymbol{z}_t$ the gradients can become small. If we assume that $\rho'(x)$ is bounded by γ (as is the case for the logistic sigmoid, $\tanh$, and ReLU, where $\gamma = 1/2$), then we can bound the 2-norm of $\boldsymbol{\delta}_t$ as

$$\|\boldsymbol{\delta}_t\|_2 \leq \gamma \cdot \|\boldsymbol{V}^\intercal\|_2 \cdot \|\boldsymbol{\delta}_{t+1}\|_2 \leq \gamma^{T-t} \cdot \|\boldsymbol{V}^\intercal\|_2^{T-t} \cdot \|\boldsymbol{\delta}_T\|_2.$$

Thus if the largest singular value of $\boldsymbol{V}^\intercal$ satisfies $\|\boldsymbol{V}^\intercal\|_2 \leq c/\gamma$ for some $c < 1$, then

$$\|\boldsymbol{\delta}_t\|_2 \leq c^{T-t}\|\boldsymbol{\delta}_T\|_2,$$

and the contribution of $\boldsymbol{\delta}_t$ to the gradient can become negligible for small t and large values of T. Conversely, similar inequalities in the other direction using the smallest singular value show that the gradient can become very large. One common strategy is to use truncated backpropagation, which limits how far back in time the algorithm looks, but this obviously also limits the long-term memory of the RNN.

Example 20.4. Consider the problem of determining whether a binary sequence $\boldsymbol{X} = \{x_t\}_{t=1}^T$ has an odd number of 1s. We consider a simple RNN of the form

$$h_t = \sigma(wx_t + vh_{t-1}), \qquad t \geq 1,$$

where σ is the logistic sigmoid. The output of the RNN is $y = h_T$. As loss function we use the log-loss function, or cross-entropy. Even when gradient descent converges well for training this problem, the resulting RNN can fail to correctly classify the test sequences. Figure 20.3 shows the decrease in magnitude of the $\boldsymbol{\delta}_t$ for different values of w and v, based on 20 sample sequences of length 10.

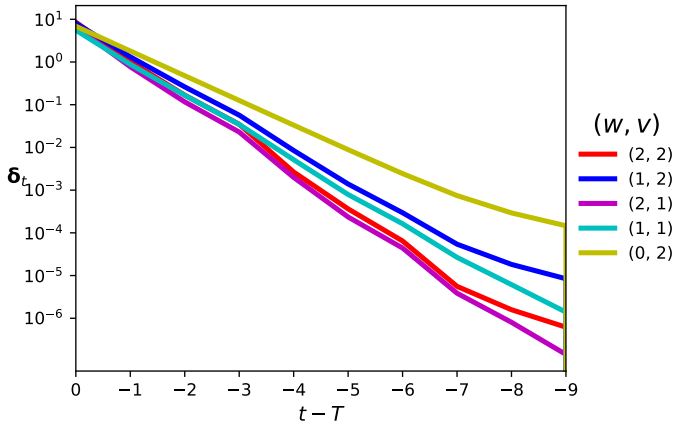


Figure 20.3: Decay of the sensitivities to changes at position $T - t$.

This illustrates that information from the first few digits is not accounted for, and therefore this specific form of a RNN is not suitable for the task of determining whether the number of 1s is odd or even. A simple way of solving the problem in this specific example is by using the XOR function, as used in Example 20.1.

Long Short-Term Memory Networks

One of the problems with RNNs trained using backpropagation in time is that they can have problems with long-range dependencies, as discussed in the previous section. Long Short-Term Memory networks (LSTMs) address this specific issue. A common implementation of the Long Short-Term Memory network makes use of two types of activation functions, a sigmoid σ and the hyperbolic tangent $\tanh$. [1] A layer of an LSTM is then specified by the following operations:

[1] As with all types of neural network architecture, there are small variations in the precise implementation, which vary from source to source.

$$
\begin{aligned}
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_f \boldsymbol{x}_t + \boldsymbol{V}_f \boldsymbol{h}_{t-1} + \boldsymbol{b}_f) \\
\boldsymbol{\iota}_t &= \sigma(\boldsymbol{W}_\iota \boldsymbol{x}_t + \boldsymbol{V}_\iota \boldsymbol{h}_{t-1} + \boldsymbol{b}_\iota) \\
\boldsymbol{g}_t &= \tanh(\boldsymbol{W}_g \boldsymbol{x}_t + \boldsymbol{V}_g \boldsymbol{h}_{t-1} + \boldsymbol{b}_g) \\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_o \boldsymbol{x}_t + \boldsymbol{V}_o \boldsymbol{h}_{t-1} + \boldsymbol{b}_o) \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{\iota}_t \odot \boldsymbol{g}_t \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)
\end{aligned}
$$

One can visualize an LSTM as in Figure 20.4.

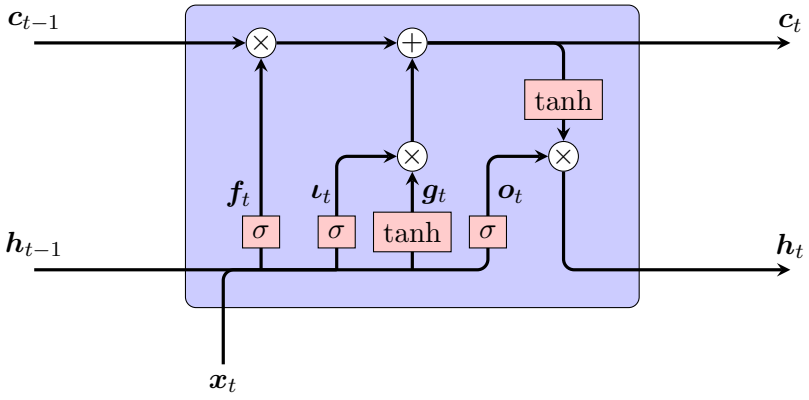


Figure 20.4: LSTM Architecture.

The **cell state** $\boldsymbol{c}_t$ represents the relevant "memory" of the network. The vector $\boldsymbol{f}_t$ decides which information to keep from $\boldsymbol{c}_{t-1}$. The vector $\boldsymbol{\iota}_t$ decides which information to update in the cell state, and the information to be updated is provided by $\boldsymbol{g}_t$. The cell is then updated accordingly. The vector $\boldsymbol{o}_t$ decides which parts of the cell state should be used as output, and is multiplied componentwise to $\tanh(\boldsymbol{c}_t)$, where the $\tanh$ forces the output into the interval $[-1, 1]$.

Backpropagation for LSTMs is derived in a similar way as for generic RNNs. For this, it is convenient to rewrite the LSTM as follows:

$$
\begin{aligned}
\boldsymbol{z}_t &= \boldsymbol{W} \boldsymbol{x}_t + \boldsymbol{V} \boldsymbol{h}_{t-1} + \boldsymbol{b} \\
\boldsymbol{y}_t &= \rho(\boldsymbol{z}_t) \\
\boldsymbol{c}_t &= g(\boldsymbol{y}_t, \boldsymbol{c}_{t-1}) \\
\boldsymbol{h}_t &= f(\boldsymbol{y}_t, \boldsymbol{c}_t),
\end{aligned}
\tag{20.4}
$$

where the $\boldsymbol{f}_t$, $\boldsymbol{\iota}_t$, $\boldsymbol{g}_t$, and $\boldsymbol{o}_t$ have been combined into a single $\boldsymbol{y}_t$, and ρ stands for the combined activation function that operates as $\tanh$ on $\boldsymbol{g}_t$ and σ on the other coordinates. This form is reminiscent of (20.2), the difference being that the recursion is for the cell states $\boldsymbol{c}_t$, and $\boldsymbol{h}_t$ is computed from these.

Notes

The concept of a recurrent neural networks and backpropagation through time was introduced by Rumelhart, Hinton and Williams [65]. There are many different ways in which a RNN can be specified, but the common feature is a recurrence relation that updates a current state based on not only the input, but also on the value of previous states.

The vanishing gradient problem was observed and studies in [38] and [8], and analysed further in [61]. Long short-term memory networks were introduced by Hochreiter and Schmidhuber [39].

21

Attention and the Transformer

The attention mechanism is a powerful concept that first arose in the context of machine translation, and has since become an important ingredient in many applications of deep learning. One motivation for the attention mechanism is how we pay attention to the relationship between certain words in a sentence in order to make sense of the whole sentence. Before introducing the attention model and the transformer, we set the stage by discussing sequence to sequence models in the context of natural language processing.

Language Models

In what follows, when convenient we use the shorthand notation $\boldsymbol{x}_{1:n}$ to denote a sequence $\{\boldsymbol{x}_i\}_{i=1}^n$. A language model is a probability distribution over a set of words. More precisely, given a sequence $\{\boldsymbol{x}_i\}_{i=1}^n$, we consider the joint probability that factors over conditional probabilities

$$p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = p(\boldsymbol{x}_n \mid \boldsymbol{x}_{1:n-1}) \cdots p(\boldsymbol{x}_2 \mid \boldsymbol{x}_1) \cdot p(\boldsymbol{x}_1).$$

In applications such as text generation and machine translation, the next word $\boldsymbol{x}_{n+1}$ in a sequence is chosen as the word that is the most likely given the previous words for a particular learned distribution:

$$\boldsymbol{x}_{n+1} = \arg\max_{\boldsymbol{x}} p(\boldsymbol{x} \mid \boldsymbol{x}_{1:n}).$$

In practice, words are encoded using a dictionary that assigns to each word a unique number. One common way of implementing such an encoding is using one-hot encoding. To describe this, let $\mathcal{D}$ be a dictionary of words and assume $|\mathcal{D}| = N$. Then a one-hot encoding is an injective map

$$\varphi \colon \mathcal{D} \to \{0, 1\}^N,$$

such that for each word $\boldsymbol{x}$, $\varphi(\boldsymbol{x})$ has exactly one non-zero entry. Thus each word is represented using a binary vector of length equal to the size of the dictionary, and the location of the non-zero entry uniquely identifies the word. The map $\boldsymbol{x} \mapsto p(\boldsymbol{x} \mid \boldsymbol{x}_{1:n})$ can be represented as a vector $\boldsymbol{p}^{1:n} := (p_1, \ldots, p_n) \in \mathbb{R}^N$, and we chose $\boldsymbol{x}_{n+1}$ as the word corresponding to the largest entry via the one-hot encoding:

$$\boldsymbol{x}_{n+1} = \arg\max_{\boldsymbol{x} \in \mathcal{D}} \varphi(\boldsymbol{x})^T \boldsymbol{p}^{1:n}.$$

Sequence to sequence

In machine translation, the task is to transform one sequence of words into another sequence in a meaningful way. A natural setting for this problem is that of recurrent neural networks (RNN). Recall the

general structure of a RNN,

$$\boldsymbol{h}_t = f(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}),$$

where $\{\boldsymbol{x}_t\}_{t\geq 1}$ is an input sequence and $\{\boldsymbol{h}_t\}_{t\geq 0}$ is a sequence of states. The function f can involve various intermediate steps, including linear maps and various activation functions. It could, for example, represent the state sequence of an LSTM.

One approach to machine translation is to use a pair of RNNs, one for encoding the input sequence into a context vector, and one for generating the output sequence. An encoder RNN takes as input a sequence $\{\boldsymbol{x}_t\}_{t\geq 1}$ and outputs a fixed-size context vector $\boldsymbol{c}$ as a function of the sequence $\{\boldsymbol{h}_t\}_{t\geq 0}$ (usually, a function of the last entry $\boldsymbol{h}_T$). A decoder RNN is an RNN of the form $\boldsymbol{s}_t = f(\boldsymbol{y}_{t-1}, \boldsymbol{s}_{t-1}; \boldsymbol{c})$ that generates a probability conditional distribution on the next entry of an output sequence:

$$p(\cdot \mid \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{t-1}; \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) = g(\boldsymbol{y}_{t-1}, \boldsymbol{s}_t; \boldsymbol{c}). \tag{21.1}$$

In the context of machine translation, the probability is represented as a discrete probability vector over a dictionary of words or sometimes characters. As next $\boldsymbol{y}_t$, we pick the word or character with the highest probability. Note that the dependence on the input sequence $\{\boldsymbol{x}_t\}_{t\geq 1}$ occurs via the context vector $\boldsymbol{c}$, while the dependence on the previous output entries $\boldsymbol{y}_{t-j}$ is via the state variable $\boldsymbol{s}_t$. In practice, it is common to implement the encoder and decoder using LSTM networks. By multiplying the conditional probabilities in (21.1), we get the conditional probability for every sentence $\{\boldsymbol{y}_t\}_{t\geq 1}$, given as

$$p(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{T'} \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) = \prod_{t=1}^{T'} p(\boldsymbol{y}_t \mid \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{t-1}; \boldsymbol{x_1}, \ldots, \boldsymbol{x}_T).$$

Intuitively, int he context of machine translation, this represents the probability that $\{\boldsymbol{y}_i\}_{t\geq 1}$ is the correct translation of $\{\boldsymbol{x}_t\}_{t\geq 1}$. Training data then consists of pairs $(\boldsymbol{X}, \boldsymbol{Y})$, where $\boldsymbol{X}$ is a sentence in one language and $\boldsymbol{Y}$ is a sentence in another language. The objective function is then the conditional log-likelihood

$$\frac{1}{N} \sum_{(\boldsymbol{X}^i, \boldsymbol{Y}^i)} \log p(\boldsymbol{Y}^i \mid \boldsymbol{X}^i),$$

where N is the size of the training dataset, which is maximized over the parameters of the encoder and decoder networks that combine to parametrize the probability p.

One challenge related to this form of sequence to sequence models is the fixed length of the vector $\boldsymbol{c}$ that is passed to the decoder. Another, more substantial limitation is the inability to model alignment between parts of the input sentence and parts of the output sentence. Both these limitations are addressed by the attention mechanism.

The Attention Mechanism

Rather than considering a fixed context vector $\boldsymbol{c}$ that parametrizes the decoder, attention models consider for each state $\boldsymbol{s}_i$ of the output network a vector of the form

$$\boldsymbol{c}_i = \sum_{j=1}^{T} \alpha_{ji} \boldsymbol{h}_j,$$

that is, a linear combination of the states of the encoder network. As before, the distribution on the output dictionary is then constructed as

$$p(\cdot \mid \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{t-1}; \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) = g(\boldsymbol{y}_{t-1}, \boldsymbol{s}_t; \boldsymbol{c}_t). \tag{21.2}$$

To construct the alignment weights, we first construct an alignment function $a(\boldsymbol{s}_{j-1}, \boldsymbol{h}_i)$ that assigns a score that quantifies how relevant the hidden encoder state $\boldsymbol{h}_i$ is to the decoder state $\boldsymbol{s}_{j-1}$:

$$e_{ji} = a(\boldsymbol{s}_{j-1}, \boldsymbol{h}_i).$$

An intuitive measure of how much one vector is aligned with another is the dot product, hence one popular choice is to use a scaled version of $a(\boldsymbol{s}, \boldsymbol{h}) = \boldsymbol{s}^T \boldsymbol{h}$, but taking other function, such as a one-layer fully connected neural network with $\tanh$ activation, is also possible. These scores are then transformed into a probability distribution using the softmax function

$$\alpha_{ji} = \rho([e_{j1}, \ldots, e_{jT}]) := \frac{\exp(e_{ji})}{\sum_i \exp(e_{ji})}.$$

While at first sight this just seems to add complexity to the problem, the key observation is that the alignment weights α_{ij} can be learned by using an addition feedforward neural network. The following example shows alignment scores between a French and an English sentence computed by an attention model.

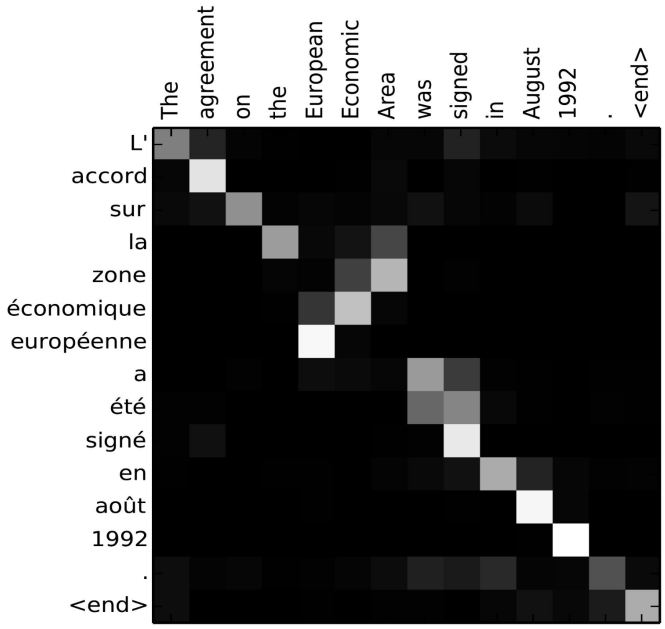


Figure 21.1: Alignment scores between a French and English sentence. Source: [2]

To define the attention mechanism in more generality, we define matrices of **keys**, **queries** and **values** as follows:

$$\begin{aligned}
\boldsymbol{K} &= [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_{T_k}]^T \in \mathbb{R}^{T_k \times d}, \\
\boldsymbol{Q} &= [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{T_q}]^T \in \mathbb{R}^{T_q \times d}, \\
\boldsymbol{V} &= [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{T_v}]^T \in \mathbb{R}^{T_v \times d_v}.
\end{aligned}$$

The attention mechanism combines the keys, queries and values into a matrix of outputs as

$$\text{attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \rho\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V},$$

where ρ is the softmax function. In the most basic examples of scaled dot product attention, the queries are given by $\boldsymbol{q}_i = \boldsymbol{s}_{i-1}$ and the keys and values by $\boldsymbol{k}_j = \boldsymbol{v}_j = \boldsymbol{h}_j$, so that the transpose of the j-th row of the output matrix is given by

$$\sum_{i=1}^{d} \rho\left(\frac{\boldsymbol{s}_{j-1}^T \boldsymbol{h}_i}{\sqrt{d}}\right) \boldsymbol{h}_i.$$

The Transformer

The transformer is a particular implementation of the attention mechanism that does not rely on the use of recurrent encoder-decoder networks. Specifically, a transformer uses the concept of **stacked self-attention** and fully connected encoder and decoder networks. Here, self-attention means that the attention mechanism is based on the same input and output sequence, that is $\boldsymbol{s}_i = \boldsymbol{h}_i$. The encoder is composed of N identical layers. Each of these layers consists of two sublayers, one of them being a multi-head attention mechanism and one a fully connected sublayer. These are connected using the idea of residual connections, layer normalization and dropout. The decoder similarly consists of N identical layers.

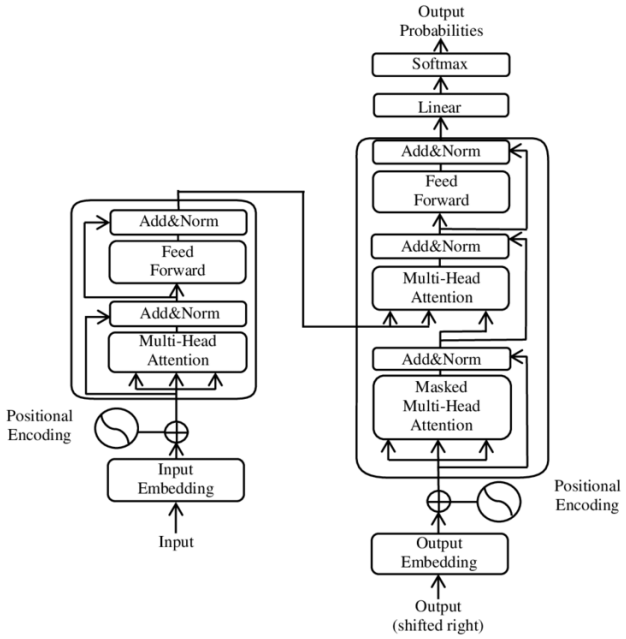


Figure 21.2: The basic transformer architecture. Source: [82]

Multi-head Attention

Multi-head attention allows the model to attend to information different representation subspaces. Specifically, we introduce projection matrices $\boldsymbol{W}_i^Q$, $\boldsymbol{W}_i^K$ and $\boldsymbol{W}_i^V$, and set

$$\text{MultiAtt}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = [\text{head}_1, \ldots, \text{head}_h]\boldsymbol{W}^o$$

for an output matrix $\boldsymbol{W}^o$, where

$$\text{head}_i = \text{attention}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V).$$

While we now employ multiple attention functions, the dimension on which each one operates is smaller, leading to a comparable complexity.

Notes

Sequence to sequence models, as discussed here, were introduced in [19, 75]. Attention was introduced in [2], see also [51]. The seminal paper [82] introduced the concept of a transformer, and has become one of the most influential papers in machine learning of the past decade. Examples of applications that use the transformer technology include OpenAI's GPT-3 and ChatGPT language models (GPT stands for Generative Pre-trained Transformer), as well as DeepMinds AlphaFold system for predicting the structure of proteins from amino acid sequences [43].

22

Robustness

There is no unique way to construct a classifier. A good classifier is expected to have small generalization error. In addition, we expect a good classifier to be **robust**: by this, we mean that a small perturbation of an object should not move it to a different class. In this lecture we discuss how to determine the smallest perturbation that would make an object change class, discuss a method of generating such **adversarial perturbations**, and derive some theoretical limits on robustness.

Adversarial Perturbations

Let $h\colon \mathbb{R}^d \to \mathcal{Y}$ be any classifier. Define the smallest perturbation that moves a data point into a different class as

$$\Delta_h(\boldsymbol{x}) = \inf_{\boldsymbol{r}} \{\|\boldsymbol{r}\|\colon h(\boldsymbol{x}+\boldsymbol{r}) \neq h(\boldsymbol{x})\}. \tag{22.1}$$

Given a probability distribution on the input spaces, define the **robustness** as

$$\rho(h) = \mathbb{E}\left[\frac{\Delta_h(X)}{\|X\|}\right].$$

We begin by discussing the special case of binary classification using a hyperplane, which we already saw previously in the context of linear support vector machines (Figure 22.1).

The aim was to find a hyperplane that separates two sets of points and has *maximal margin*. Assume we are given linearly separable points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$ with labels $y_i \in \{-1, 1\}$ for $i \in \{1, \ldots, n\}$. A separating hyperplane is defined as a hyperplane

$$H = \{\boldsymbol{x} \in \mathbb{R}^d\colon \boldsymbol{w}^T\boldsymbol{x} + b = 0\},$$

where $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ are such that for all $i \in \{1, \ldots, d\}$,

$$\begin{aligned} \boldsymbol{w}^T\boldsymbol{x}_i + b > 0 \text{ if } y_i = 1 \\ \boldsymbol{w}^T\boldsymbol{x}_i + b < 0 \text{ if } y_i = -1. \end{aligned}$$

Define the classifier $h(\boldsymbol{x}) = \operatorname{sign}(\boldsymbol{w}^T\boldsymbol{x} + b)$, and let H_+ and H_- be the open half-spaces where $h(\boldsymbol{x}) = 1$ and $h(\boldsymbol{x}) = -1$. Given any point $\boldsymbol{x} \notin H$, we define the *smallest perturbation* and the *distance* to H as

$$\boldsymbol{r}^* = \operatorname{argmin} \frac{1}{2}\|\boldsymbol{r}\|^2 \qquad \text{subject to} \quad \boldsymbol{w}^T(\boldsymbol{x}+\boldsymbol{r}) + b = 0.$$

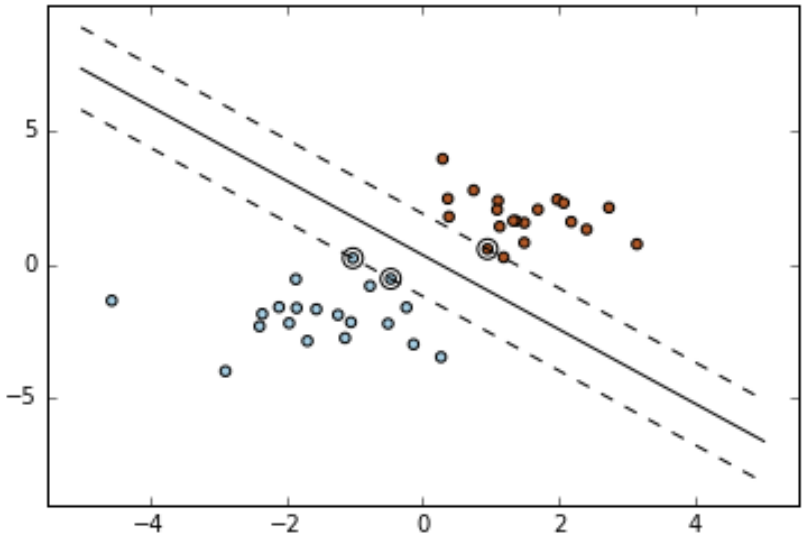


Figure 22.1: A hyperplane separating two sets of points with margin and support vectors.

We can obtain a closed form solution either geometrically, or by considering the Lagrangian, since the function to be minimized is convex:

$$\mathcal{L}(\boldsymbol{r}, \lambda) = \frac{1}{2}\|\boldsymbol{r}\|^2 + \lambda(\boldsymbol{w}^T(x + \boldsymbol{r}) + b).$$

Taking the gradient with respect to r, we get the relation

$$\nabla_{\boldsymbol{r}}\mathcal{L}(\boldsymbol{r}, \lambda) = \boldsymbol{r} + \lambda\boldsymbol{w} = \boldsymbol{0} \;\Rightarrow\; \boldsymbol{r}^* = -\lambda^*\boldsymbol{w}. \tag{22.2}$$

To determine the Lagrange multiplier, we take the inner product with $\boldsymbol{w}$ and obtain

$$\boldsymbol{w}^T\boldsymbol{r}^* = -\lambda^*\|\boldsymbol{w}\|^2 \;\Rightarrow\; \lambda^* = -\frac{\boldsymbol{w}^T\boldsymbol{r}^*}{\|\boldsymbol{w}\|^2}.$$

Using the fact that $\boldsymbol{w}^T\boldsymbol{r}^* = -(\boldsymbol{w}^T\boldsymbol{x} + b)$, and replacing λ^* in (22.2), we get

$$\boldsymbol{r}^* = -\frac{\boldsymbol{w}^T\boldsymbol{x} + b}{\|\boldsymbol{w}\|^2} \cdot \boldsymbol{w}, \quad \Delta(\boldsymbol{x}) = \frac{|\boldsymbol{w}^T\boldsymbol{x} + b|}{\|\boldsymbol{w}\|}.$$

Note that this result is compatible with the one derived in the context of SVM, where we normalized $\boldsymbol{w}$ and b such that $\boldsymbol{w}^T\boldsymbol{x} + b = 1$. It follows that in order to change a data point $\boldsymbol{x}$ from being classified as 1 to being classified as -1, we just have to add $(1 + \epsilon)\boldsymbol{r}^*$ for a small value of ϵ.

The setting generalizes to classification with more than one class. Assume that we have k linear functions $f_1, \ldots, f_k$, with $f_i(\boldsymbol{x}) = \boldsymbol{w}_i^T\boldsymbol{x} + b_i$, and a classifier $h\colon \mathbb{R}^d \to \{1, \ldots, k\}$ that assigns to each $\boldsymbol{x}$ the index j of the largest value $f_j(\boldsymbol{x})$ (this corresponds to the one-to-many setting for multiple classification). Let $\boldsymbol{x}$ be such that $\max_j f_j(\boldsymbol{x}) = f_k(\boldsymbol{x})$ and define the linear functions

$$g_i(\boldsymbol{x}) := f_i(\boldsymbol{x}) - f_k(\boldsymbol{x}) = (\boldsymbol{w}_i - \boldsymbol{w}_k)^T\boldsymbol{x} + (b_i - b_k), \quad i \in \{1, \ldots, k-1\}.$$

Then

$$\boldsymbol{x} \in \bigcap_{1 \le i \le k-1} \{\boldsymbol{y} \colon g_i(\boldsymbol{y}) < 0\}.$$

The intersection of half-spaces is a **polyhedron**, and $\boldsymbol{x}$ is in the interior of the polyhedron P delineated by the hyperplanes $H_i = \{\boldsymbol{y} \colon g_i(\boldsymbol{x}) = 0\}$ (see Figure 22.2).

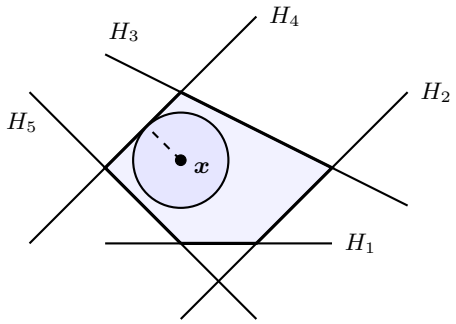


Figure 22.2: The **distance to misclassification** is the radius of the largest enclosed ball in a polyhedron P.

A perturbation $\boldsymbol{x} + \boldsymbol{r}$ ceases to be in class k as soon as $g_j(\boldsymbol{x} + \boldsymbol{r}) > 0$ for some j, and the smallest length of such an r equals the radius of the largest enclosed ball in the polyhedron. Formally, noting that $\boldsymbol{w}_j - \boldsymbol{w}_k = \nabla g_j(\boldsymbol{x})$, we get

$$\hat{j} := \arg\min_j \frac{|g_j(\boldsymbol{x})|}{\|\nabla g_j(\boldsymbol{w})\|}, \quad \boldsymbol{r}^* = -\frac{g_{\hat{j}}(\boldsymbol{x})}{\|\nabla g_{\hat{j}}(\boldsymbol{x})\|^2} \cdot \nabla g_{\hat{j}}(\boldsymbol{x}). \tag{22.3}$$

The formulation (22.3) suggests how to approach the case where f_i are non-linear functions. In this case, we work with the first-order approximation at a point $\boldsymbol{x}$:

$$f_j(\boldsymbol{y}) \approx f_j(\boldsymbol{x}) + \nabla f_j(\boldsymbol{x})^T (\boldsymbol{y} - \boldsymbol{x}).$$

The **DeepFool** Algorithm is an iterative greedy algorithm that starts with a point $\boldsymbol{x}_0$, and then for each $i \ge 0$, determines $\boldsymbol{x}_{i+1}$ as the closest point in the boundary of the polyhedron defined by the linearizations of the f_j around $\boldsymbol{x}_i$. Once we have determined class in which $\boldsymbol{x}$ lives, we can also use any other suitable algorithm for solving the constrained optimization problem. Figure 22.3 shows an image of a handwritten "4" from the MNIST digits database. A CNN that was trained to accuracy 98% on 10000 test samples correctly identifies the digit, but the perturbed image, while still clearly depicting a 4, is mistaken for an 9 by the same network. Overall, the accuracy of the network drops to 67% when perturbing the whole test data by the same magnitude as that of the perturbation in Figure 22.3.

Fundamental Limits

To study the effect of *random* perturbations, we consider a generative model for our data. Let $\mathcal{Z} = \mathbb{R}^m$ and let Z be a Gaussian random variable with mean 0 and unit covariance on $\mathcal{Z}$. Consider a data generating process $g \colon \mathcal{Z} \to \mathcal{X}$, where $\mathcal{X} = \mathbb{R}^d$ is the data space. Given a classifier $f \colon \mathcal{X} \to \{1, \dots, k\}$, denote by $C_i = f^{-1}(i)$ the region of $\mathcal{X}$ corresponding to class i, $i \in \{1, \dots, k\}$. Moreover, denote by $h = f \circ g \colon \mathcal{Z} \to \{1, \dots, k\}$ the composite map. In addition to the error $\Delta(\boldsymbol{x})$, defined in (22.1), we define the **in-distribution error** for $\boldsymbol{x} = g(\boldsymbol{z})$ as

$$\hat{\Delta}(\boldsymbol{x}) = \inf\{\|g(\boldsymbol{z} + \boldsymbol{r}) - g(\boldsymbol{z})\| \,:\, \boldsymbol{r} \in \mathbb{R}^m, h(\boldsymbol{z} + \boldsymbol{r}) \neq h(\boldsymbol{z})\}.$$

Figure 22.3: A correctly identified 4 and a perturbed image, identified as a 9.

Since $\hat{\Delta}$ is based on the distance between $\boldsymbol{x}$ *in the image* of g, we clearly have $\Delta(\boldsymbol{x}) \leq \hat{\Delta}(\boldsymbol{x})$. In the following, we assume that $X = g(Z)$, that is, X is distributed on $\mathcal{X}$ with the push-forward measure of the Gaussian on $\mathcal{Z}$ under g.

Theorem 22.1. *(Fawzi[3]) Let $f\colon \mathbb{R}^d \to \{1, \dots, k\}$ be any classifier, and let $g\colon \mathbb{R}^m \to \mathbb{R}^d$ be L-Lipschitz continuous. Then for all $\epsilon > 0$,*

$$\mathbb{P}\{\hat{\Delta}(X) \leq \epsilon\} \geq 1 - \sqrt{\frac{\pi}{2}} \mathrm{e}^{-\frac{\epsilon^2}{2L^2}}.$$

A consequence of this result is that when the Lipschitz constant L is small with respect to ϵ, then with high probability one will be close to a boundary between classes.

Notes

The content of this lecture is based on the two papers [58] and [27]. Additional material can be found in [57].

23

Generative Adversarial Networks

In Machine Learning, one distinguishes between **discrimitative** and **generative models**. Given an input space $\mathcal{X}$, an output space $\mathcal{Y}$, and a probability distribution on $\mathcal{X} \times \mathcal{Y}$, the discriminative setting is concerned with predicting and output for a given input, or more specifically, with the *regression function*

$$f(x) = \mathbb{E}[Y \mid X = x].$$

More generally, the goal is to learn the distribution of Y conditioned on X. A generative model, on the other hand, seeks to understand the joint distribution $\rho_{X,Y}$ of inputs and outputs. A typical example of such a model is the problem of estimating the parameters of a mixture distribution, where the observed data is assumed to come from one of two or more distributions. Understanding the distribution that gave rise to a large set of training data may allow to *sample* from that distribution, and generate additional "realistic" data that shares many features with the original training data. In this lecture we introduce a framework that has allowed to generate realistic, synthetic data based on neural networks.

Generative Adversarial Nets

We consider a data space $\mathcal{X}$ with probability distribution ρ_X. A **generator** is a measurable map $G\colon \mathcal{Z} \to \mathcal{X}$, where $\mathcal{Z}$ is the **latent space** with probability measure ρ_Z. Thus the generator *generates* data in the input space. The push-forward measure of ρ_Z on $\mathcal{X}$ (i.e., the density of the random variable $G(Z)$, where Z is distributed according to ρ_Z) is denoted by ρ_G, and typically differs from ρ_X. A **discriminator** is a map $D\colon \mathcal{X} \to [0,1]$. A **Generative Adversarial Net** consists of a generator-discriminator pair (G, D), where both G and D are represented by neural networks that are trained according to complementary objectives. The discriminator aims to distinguish training data, sampled from the distribution ρ_X on $\mathcal{X}$, from data generated by G, or equivalently, sampled from the distribution ρ_G on $\mathcal{X}$. The generator G aims to make it impossible for D to distinguish between the data it generated and data sampled from ρ_X.

The conflicting goals of D and G can be captured using a cost function

$$V(D,G) = \mathbb{E}_X[\log(D(X))] + \mathbb{E}_Z[\log(1 - D(G(Z)))], \tag{23.1}$$

which is a function of the parameters that define D and G. The goal of D is thus to *maximize* $V(D,G)$, while the goal of G is to *minimize* this function.

For a fixed generator G, maximizing the expectation (23.1) amounts to maximizing the function

$$\int_{\mathcal{X}} [\log(D(x))\rho_X(x) + \log(1 - D(x))\rho_G(x)] \, \mathrm{d}x.$$

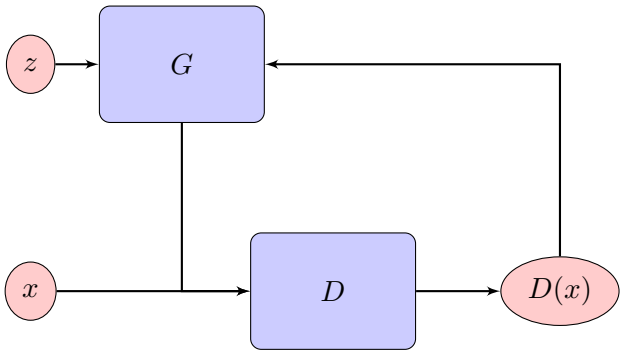


Figure 23.1: Structure of a GAN

A short calculation shows that the function $\log(y)p + \log(1-y)q$ is maximized at $y = p/(p+q)$, and hence the optimal discriminator is given by

$$D^*(x) = \frac{\rho_X(x)}{\rho_X(x) + \rho_G(x)} = 1 - \frac{\rho_G(x)}{\rho_X(x) + \rho_G(x)}. \tag{23.2}$$

Remark 23.1. We can interpret the setting as follows. Let X_1, X_2 be random variables with values in $\mathcal{X}$, where X_1 is distributed according to ρ_X and X_2 is distributed according to ρ_G. Moreover, let Y be a Bernoulli random variable with values in $\mathcal{Y} = \{0, 1\}$ and probability $\mathbb{P}\{Y = 1\} = 1/2$. Consider then the random variable (X', Y) on $\mathcal{X} \times \mathcal{Y}$, where $X' = Y \cdot X_1 + (1-Y) \cdot X_2$. We can interpret sampling a point $x \in \mathcal{X}$ as sampling from the distribution ρ_X or ρ_G with equal probability. The conditional densities of X' given the values of Y are

$$\rho_{X'|Y=1}(x) = \rho_X(x), \quad \rho_{X'|Y=0}(x) = \rho_G(x).$$

Hence, by Bayes rule we get for the regression function

$$\begin{aligned}
\mathbb{E}[Y \mid X = x] &= \mathbb{P}\{Y = 1 \mid X = x\} \\
&= \frac{\mathbb{P}\{X = x \mid Y = 1\}\mathbb{P}\{Y = 1\}}{\mathbb{P}\{X = x \mid Y = 1\}\mathbb{P}\{Y = 1\} + \mathbb{P}\{X = x \mid Y = 0\}\mathbb{P}\{Y = 0\}} \\
&= \frac{\rho_X(x)}{\rho_X(x) + \rho_G(x)} = D^*(x).
\end{aligned}$$

The function $D^*(x)$ thus corresponds to the Bayes classifier,

$$h(x) = \begin{cases} 1 & D^*(x) > 1/2 \\ 0 & D^*(x) \leq 1/2. \end{cases}$$

In practice, we are only given data samples $x_1, \ldots, x_n$ and an indicator y_i, where $y_i = 1$ if x_i has been sampled from ρ_X and $y_i = 0$ if it has been sampled from ρ_G. The *empirical risk* to be minimized thus becomes

$$-\frac{1}{n}\sum_{i=1}^{n} y_i \log(D(x_i)) + (1-y_i)\log(1 - D(x_i)), \tag{23.3}$$

which is precisely the *log-loss* function applied to a classifier that assigns to input data a probability p of coming from ρ_X.

Example 23.2. Let $\mathcal{Z} = \mathcal{X} = \mathbb{R}$. On $\mathcal{Z}$ we take the Gaussian density $\rho_Z(z) = (2\pi)^{-1/2}\exp(-z^2/2)$, and on $\mathcal{X}$ a scaled and shifted normal distribution $\rho_X(x) = (2\pi\sigma^2)^{-1/2}\exp(-(x-m)^2/2\sigma^2)$. As generator we consider a linear function $G(z) = ax + b$. Assuming a fixed generator G, the goal of the discriminator is to distinguish between samples coming from the two distributions on $\mathcal{X}$:

$$\rho_G(x) = \frac{1}{\sqrt{2\pi}a}\mathrm{e}^{-\frac{(x-b)^2}{2a^2}}, \quad \rho_X(x) = \frac{1}{\sqrt{2\pi}\sigma}\mathrm{e}^{-\frac{(x-m)^2}{2\sigma^2}}.$$

In other words, we aim to determine the parameters of a **Gaussian mixture distribution**, see Figure 23.2. In this example, the function of the form (23.2) can be realized within the class of neural networks with

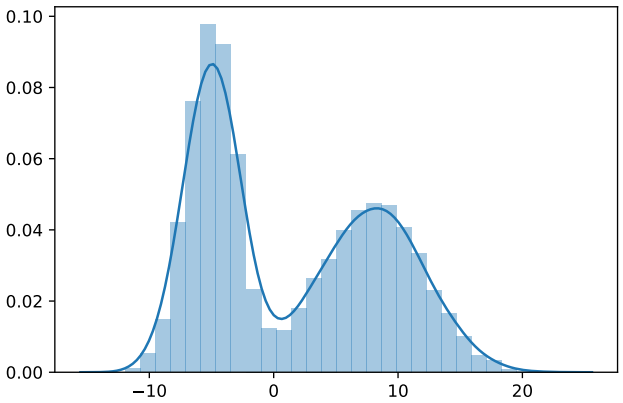


Figure 23.2: A Gaussian mixture distribution with histogram.

one hidden layer consisting of two nodes, and normalized radial basis function activation.

The optimal discriminator D^* depends on the generator G. We next aim to minimize $V(D^*, G)$ over all generators.

Theorem 23.3. *The function $V(D^*, G)$, where D^* is defined as in (23.1), satisfies*

$$V(D^*, G) \geq -\log(4),$$

with equality if $\rho_G = \rho_X$.

Proof. If $\rho_G = \rho_X$, then $V(D^*, G) = -\log(4)$ follows from a direct calculation. To see that this is optimal, write

$$\begin{aligned}
V(D^*, G) &= \int_{\mathcal{X}} \log(D^*(x))\rho_X(x)\,\mathrm{d}x + \int_{\mathcal{X}} \log(1 - D^*(x))\rho_G(x)\,\mathrm{d}x \\
&= \int_{\mathcal{X}} \log\left(\frac{\rho_X(x)}{\rho_X(x) + \rho_G(x)}\right)\rho_X(x)\,\mathrm{d}x \\
&\quad + \int_{\mathcal{X}} \log\left(\frac{\rho_G(x)}{\rho_X(x) + \rho_G(x)}\right)\rho_G(x)\,\mathrm{d}x \\
&= \int_{\mathcal{X}} \log\left(\frac{2\rho_X(x)}{\rho_X(x) + \rho_G(x)}\right)\rho_X(x)\,\mathrm{d}x \\
&\quad + \int_{\mathcal{X}} \log\left(\frac{2\rho_G(x)}{\rho_X(x) + \rho_G(x)}\right)\rho_G(x)\,\mathrm{d}x - \log(4) \\
&= D_{\mathrm{KL}}(\rho_X \,\|\, (\rho_X + \rho_G)/2) + D_{\mathrm{KL}}(\rho_G \| (\rho_X + \rho_G)/2) - \log(4),
\end{aligned}$$

where D_{KL} denotes the **Kullback-Leibler divergence** of one measure with respect to another. Using Jensen's Inequality, we see that

$$\begin{aligned}
-D_{\mathrm{KL}}(\rho_X \parallel (\rho_X + \rho_G)/2) &= \int_{\mathcal{X}} \log\left(\frac{\rho_X(x) + \rho_G(x)}{2\rho_X(x)}\right) \rho_X(x)\,\mathrm{d}x \\
&\leq \log\left(\int_{\mathcal{X}} \left(\frac{\rho_X(x) + \rho_G(x)}{2\rho_X(x)}\right) \rho_X(x)\,\mathrm{d}x\right) \\
&= \log\left(\frac{1}{2}\int_{\mathcal{X}} (\rho_X(x) + \rho_G(x))\,\mathrm{d}x\right) \\
&= \log(1) = 0,
\end{aligned}$$

and similarly with $-D_{\mathrm{KL}}(\rho_G \parallel (\rho_X + \rho_G)/2)$. It follows that

$$D_{\mathrm{KL}}(\rho_X \parallel (\rho_X + \rho_G)/2) + D_{\mathrm{KL}}(\rho_G \parallel (\rho_X + \rho_G)/2) \geq 0,$$

which completes the proof. □

In the GAN setting, $D(x; \boldsymbol{w})$ and $G(z; \boldsymbol{v})$ are assumed to be neural networks with parameters $\boldsymbol{w}$ and $\boldsymbol{v}$. The training data for D consists of samples $\{x_i\}_{i=1}^n \subset \mathcal{X}$ and $\{G(z_j)\}_{j=1}^n \subset \mathcal{X}$ for random samples $z_j \in \mathcal{Z}$, and D is trained using stochastic gradient descent applied to the log-loss function (23.3). The training data for G consists of the samples z_j, and it is trained using stochastic gradient descent for the loss function

$$\frac{1}{n}\sum_{i=1}^{n} \log(1 - D(G(z_i))).$$

Both networks are trained simultaneously, as shown in Figure 23.3.

input : training data $\{x_i\}_{i=1}^n$
output : discriminator D and generator G
$i = 0$, choose $\boldsymbol{w}^0$, $\boldsymbol{v}^0$;
while *stopping criteria not met*
  for j *from* 0 *to* $k-1$
    $\boldsymbol{w}_0^i = \boldsymbol{w}^i$;
    sample $z_1, \ldots, z_m$ from prior ρ_Z on $\mathcal{Z}$;
    sample $x_{i_1}, \ldots, x_{i_m}$ from training data;
    $\boldsymbol{g} = \frac{1}{m}\sum_{\ell=1}^m \nabla_{\boldsymbol{w}}(\log(D(x_{i_\ell}; \boldsymbol{w}_j^i)) + \log(1 - D(G(z_\ell; \boldsymbol{v}^i); \boldsymbol{w}_j^i)))$;
    update $\boldsymbol{w}_{j+1}^i = \boldsymbol{w}_j^i + \eta_j^i \boldsymbol{g}$;
  $\boldsymbol{w}^{i+1} = \boldsymbol{w}_k^i$;
  sample $z_1, \ldots, z_m$ from prior ρ_Z on $\mathcal{Z}$;
  $\boldsymbol{v}^{i+1} = \boldsymbol{v}^i - \gamma^i \frac{1}{m}\sum_{\ell=1}^m \nabla_{\boldsymbol{v}} \log(1 - D(G(z_\ell; \boldsymbol{v}^i); \boldsymbol{w}^{i+1}))$;

Figure 23.3: Training a GAN

The choice of learning rates η_j^i and γ^i, as well as the number of iterates k for training D are hyperparameters. Assuming that when given G, at every step it is possible to reach the optimal D^*, then it can be shown that the algorithm converges to an optimal G, with $\rho_G = \rho_X$. In practice, several additional considerations have to be taken into account. For example, one has to decide on a suitable *architecture* for D and G. In typical imaging applications, one chooses a CNN for D and G. Figure 23.4 shows the result of training such a GAN to generate realistic digits.

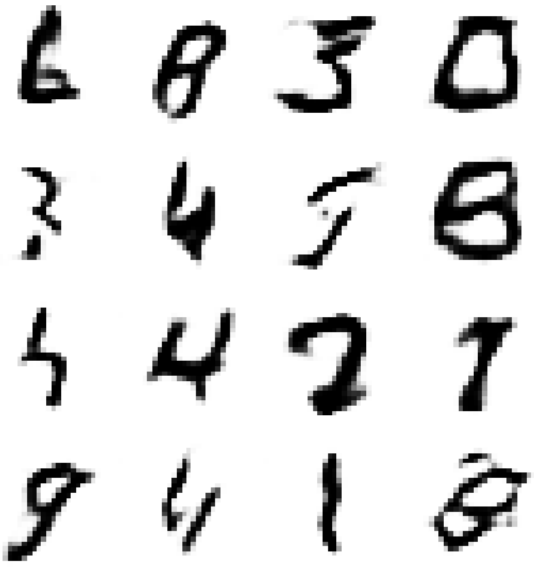


Figure 23.4: Digits generated by a GAN trained on MNIST digits for 2500 epochs with batch size $m = 128$.

Notes

Generative adversarial nets, in the way described here, were introduced by Ian Goodfellow et al in [30]. The GAN setup is also related to the concept of Predictability Minimization, introduced by Schmidhuber in the 1990s. For an overview, see [70]. A wealth of practical information on implementing GANs, as well as other information, can be found under

- https://machinelearningmastery.com/start-here/#gans

The Magenta project by Google AI provides a framework to generate music and art based, in part, on GANs:

- https://magenta.tensorflow.org/

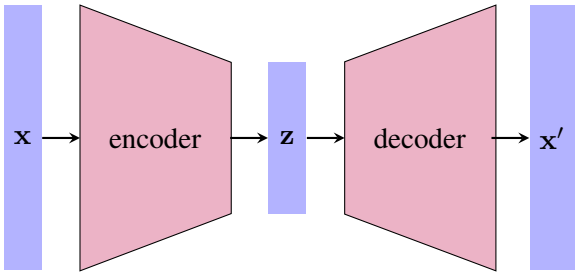
24

Variational Autoencoders

One of the strengths of neural networks is their ability to learn efficient representations of data. An efficient representation is one that represents the features of data that are relevant to a certain task in a compact way. An elementary example is image and sound compression based on the Discrete Cosine Transform (DCT), as in the JPEG or MP3 standards. Here, the relevant task is to have the image or piece of music recognizable by a human without perceivable loss in quality. An efficient representation for this task can be achieved by transforming the image or sound into the frequency domain, which often reveals that a signal can be well-approximated using only very few frequencies, and hence only these need to be stored. Another example is dimension reduction using Principal Component Analysis (PCA), with applications to object recognition tasks. Computing an efficient representation if often referred to as encoding, while reconstructing an object from this representation is called decoding. Methods such as DCT or PCA are unsupervised approaches that rely on domain knowledge. In contrast, autoencoders are encoder-decoder pairs, where the encoding and decoding procedures are implemented using neural networks whose parameters are learned from training data. Variational Autoencoders (VAE) take a probabilistic approach to the problem. The distribution of the observed data $\boldsymbol{x}$ is assumed to depend on latent parameters $\boldsymbol{z}$, and rather than computing an encoding directly from data, the aim to learn the joint distribution of $\boldsymbol{x}$ and $\boldsymbol{z}$. As such, VAE are directed probabilistic graphical models.

Autoencoders and dimension reduction

An **encoder** takes an input $\boldsymbol{x}$ and produces a **representation** $\boldsymbol{z} = f(\boldsymbol{x})$ of the data. This representation is usually of lower complexity, and the process of reducing the complexity of data while keeping all its essential features is known as **dimensionality reduction**. A **decoder** reconstructs an approximation $g(\boldsymbol{z}) = \boldsymbol{x}' \approx \boldsymbol{x}$.



If we denote the encoder by E, the decoder by D, and if L denotes a loss function, then finding an

optimal encoder-decoder pair for a given data set $\{\boldsymbol{x}_i\}_{i=1}^n$ can be cast as an optimization problem:

$$\underset{D,E}{\text{minimize}} \frac{1}{n} \sum_{i=1}^{n} L(\boldsymbol{x}_i, D(E(\boldsymbol{x}_i))).$$

The encoder and decoder can be linear or nonlinear, the manner in which these are represented and parametrized depends on the application.

Example 24.1. A common example is principal component analysis (PCA). Assume the data space is $\mathbb{R}^d$, and we consider an encoder of the form $E(\boldsymbol{x}) = \boldsymbol{Q}^T\boldsymbol{x}$, where $\boldsymbol{Q} \in \mathbb{R}^{d\times m}$ consists of the first m columns of an orthogonal matrix, and the decoder has the form $D(\boldsymbol{x}) = \boldsymbol{Q}\boldsymbol{z}$ for $\boldsymbol{z} \in \mathbb{R}^m$. Then the optimization problem of finding the optimal encoder-decoder pair of the given form is equivalent to

$$\underset{\boldsymbol{Q}}{\text{minimize}} \frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{x}_i\|_2^2,$$

where the optimization is over all matrices in $\mathbb{R}^{d\times m}$ with orthogonal columns. For each data point $\boldsymbol{x}_i$, we can think of $\boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{x}_i$ as projecting $\boldsymbol{x}_i$ orthogonally to the subspace of $\mathbb{R}^d$ spanned the orthogonal columns of $\boldsymbol{Q}$. The problem is then equivalent to finding an m-dimensional linear subspace that *best explains the data*, in the sense that the squared difference between the data points and their projections is as small as possible. Denote by $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ the matrix whose rows are the n data points $\boldsymbol{x}_i$. The feature covariance matrix is then the symmetric matrix $\boldsymbol{X}^T\boldsymbol{X} \in \mathbb{R}^{d\times d}$, and it can be shown that the optimal $\boldsymbol{Q}$ has as columns the normalized eigenvectors corresponding to the m largest eigenvalues of $\boldsymbol{X}^T\boldsymbol{X}$ (recall that a symmetric matrix has real eigenvalues, and can be diagonalized by an orthogonal matrix).

An **autoencoder** is an encoder-decoder pair, where the encoder and decoder are implemented using neural networks. In this case, the optimization problem is solved using (stochastic) gradient descent. While autoencoders can potentially find better low-dimensional representations of the data than linear projections such as PCA, they are prone to overfitting. If the encoder and decoder have sufficient expressive power, they could, in principle, encode the training data in a way that achieves zero reconstruction error. For example, one could think of an encoder that represents every training data point as a point in $\mathbb{R}$, and a decoder that contains all the information for reconstructing every training data point, and merely uses the "code" $z \in \mathbb{R}$ as an index, to signal which training data point to reconstruct. This would defy the purpose of dimensionality reduction, as one would like to retain essential features of the data after encoding.

Variational Autoencoders

A **variational autoencoder** is based on a probabilistic model for the joint distribution on the product of the data space $\mathcal{X}$ and the latent space $\mathcal{Z}$,

$$\rho(\boldsymbol{x}, \boldsymbol{z}) = \rho(\boldsymbol{x}|\boldsymbol{z}) \cdot \rho(\boldsymbol{z}).$$

Given a data point $\boldsymbol{x}$, the representation $\boldsymbol{z} \in \mathcal{Z}$ is not directly computed by an encoding map, but sampled from the conditional distribution with density

$$\rho(\boldsymbol{z}|\boldsymbol{x}) = \frac{\rho(\boldsymbol{x}|\boldsymbol{z})\rho(\boldsymbol{z})}{\rho(\boldsymbol{x})}.$$

Even if we know $\rho(\boldsymbol{x}|\boldsymbol{z})$ and $\rho(\boldsymbol{z})$, computing the marginal $\rho(\boldsymbol{x}) = \int_{\mathbb{R}} \rho(\boldsymbol{x}, \boldsymbol{z})\, \mathrm{d}\boldsymbol{z}$, also called the *evidence*, is typically not tractable. In practice, the density $\rho(\boldsymbol{z}|\boldsymbol{x})$ is *approximated* by a density $q_\theta(\boldsymbol{z}|\boldsymbol{x})$,

parametrized by a set of parameters Θ. In order to make sense of what it means to approximate one density by another one, we need a notion of distance between densities. A convenient notion turns out to be the **Kullback Leibler divergence** (KL), defined as

$$\mathrm{D}_{\mathrm{KL}}(q \mid p) = \mathbb{E}_q[\log(q/p)]$$

for two densities p and q. The KL divergence has some convenient properties that justify its use in machine learning. The proof of the following lemma is left as an exercise.

Lemma 24.2. *Let p and q be two probability densities over the same space $\mathcal{X}$. Then the KL divergence is non-negative, with $\mathrm{D}_{\mathrm{KL}}(q \mid p) = 0$ if and only if $q = p$ almost everywhere.*

We can find a suitable approximation q_θ by solving the optimization problem

$$\underset{\theta \in \Theta}{\text{minimize}}\, \mathrm{D}_{\mathrm{KL}}(q_\theta \mid \rho(\cdot|\boldsymbol{x})). \tag{24.1}$$

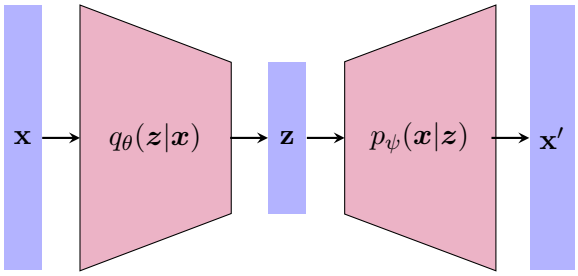


Figure 24.1: A variational autoencoder

The objective is not tractable as it is, since it involves the marginal density $\rho(\boldsymbol{x})$. This term, however, does not affect the minimizer and can therefore be removed, leading to what is known as the *evidence lower bound*. This is defined as

$$\begin{aligned}\mathrm{ELBO}(q_\theta, \boldsymbol{x}) &:= \log \rho(\boldsymbol{x}) - \mathrm{D}_{\mathrm{KL}}(q_\theta \mid \rho(\cdot|\boldsymbol{x})) \\ &= \mathbb{E}_{q_\theta}[\log \rho(\boldsymbol{x}, Z)] - \mathbb{E}_{q_\theta}[\log q_\theta(Z|\boldsymbol{x})] \\ &= \mathbb{E}_{q_\theta}[\log \rho(\boldsymbol{x}|Z)] - \mathrm{D}_{\mathrm{KL}}(q_\theta \mid \rho)\end{aligned}$$

Since $\mathrm{D}_{\mathrm{KL}}(q_\theta \mid \rho(\cdot|\boldsymbol{x})) \geq 0$, the ELBO function gives a lower bound on the "evidence" $\rho(\boldsymbol{x})$,

$$\log \rho(\boldsymbol{x}) \geq \mathrm{ELBO}(q_\theta, \boldsymbol{x}),$$

with equality if $q_\theta = \rho(\cdot|\boldsymbol{x})$. The optimization problem (24.1) is thus equivalent to the problem of maximizing the evidence lower bound. Note that $\mathrm{ELBO}(q_\theta, \boldsymbol{x})$, as defined above, depends on the data point x. The term $\mathrm{D}_{\mathrm{KL}}(q_\theta \mid \rho)$ can be seen as a regularization term.

We can parametrize the density $\rho(x|z) =: p_\psi(x|z)$ by a set of parameters ψ. In terms of the parameters (θ, ψ), the ELBO function has the form

$$\mathrm{ELBO}(\psi, \theta, \boldsymbol{x}) := \mathbb{E}_{q_\theta}[\log p_\psi(\boldsymbol{x}|Z)] - \mathrm{D}_{\mathrm{KL}}(q_\theta \mid \rho). \tag{24.2}$$

Given data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, define $\ell_i(\psi, \theta) := -\mathrm{ELBO}(\psi, \theta, \boldsymbol{x}_i)$. This gives rise to the loss function

$$L(\psi, \theta) = \sum_{i=1}^{n} \ell_i(\psi, \theta).$$

The parameters that specify the distribution q are the *variational parameters*, while the parameters (ψ, θ) are the *model parameters*. Variational inference refers to maximizing the ELBO with respect to the q (or its variational parameters), while variational EM uses the parameters (ψ, θ). These are often the parameters of a neural network.

One problem with an expression of the form (24.2) is that in involves the expected value with respect to a distribution that depends on parameters that we would like to optimize over using gradient descent. The density $\rho(\boldsymbol{z})$, as well as the conditional densities, are often taken to be Gaussian. Parametrize the encoding and decoding random variables with densities $q_\theta(\boldsymbol{z}|\boldsymbol{x})$ and $p_\psi(\boldsymbol{x}|\boldsymbol{z})$, respectively, as

$$Z = \mu_\theta(\boldsymbol{x}) + \sigma_\theta(\boldsymbol{x}) \cdot E,$$
$$X = \mu_\psi(\boldsymbol{z}) + \sigma_\psi(\boldsymbol{z}) \cdot E,$$

where E is a standard normal (Gaussian) random variable. The ELBO then takes the form

$$\mathrm{ELBO}(\psi, \theta, \boldsymbol{x}) = \mathbb{E}_E[\log(\mu_\psi(\boldsymbol{x}) + \sigma_\theta E)]$$

To implement a VAE, one needs to define the encoder and decoder architectures, as well as a loss function and an optimization algorithm. Here is a brief outline of the steps:

1. Define the encoder and decoder architectures. The encoder should take in the input and map it to a latent representation, while the decoder should take in the latent representation and map it back to the original input space. Any neural network architecture can be used for the encoder and decoder, such as fully connected layers or convolutional layers.

2. Define a loss function that measures the reconstruction error between the original input and the reconstructed output. This loss function should be minimized during training.

3. Define a prior distribution over the latent space, such as a standard normal distribution.

4. During training, pass the input through the encoder to obtain a latent representation, and then pass this latent representation through the decoder to obtain a reconstructed output. Calculate the reconstruction error using the loss function, and also calculate the KL divergence between the latent distribution learned by the VAE and the prior distribution.

5. Minimize the sum of the reconstruction error and the KL divergence using an optimization algorithm such as stochastic gradient descent (SGD).

Example 24.3. When applying variational autoencoders with digits from the MNIST database, then every digit corresponds to a conditional distribution in latent space, and sampling from this distribution one can generate realistic digits. Figure 24.2 shows the result of training a variational autoencoder for digits, and a two-dimensional projection of the distributions corresponding to the individual digits in latent space. The sample principles can be applied to generate faces or other images, and to generate sequences such as pieces of music.

Notes

Variational autoencoders have their origin in the papers [44] and [62]. The related idea of variational inference is discussed in detail in in [84] and [11]. A comprehensive account of variational autoencoders can be found in [45].

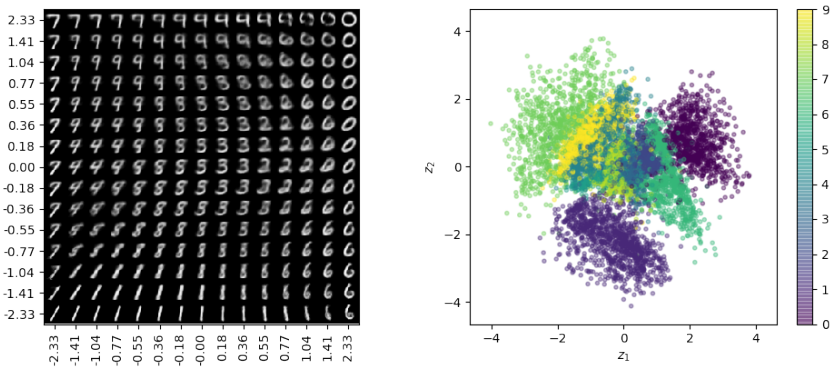


Figure 24.2: Variational autoencoder applied to digits.

25

Reinforcement Learning

If supervised learning is learning by instruction, reinforcement learning is learning by experience. The conceptual setting for reinforcement learning is that of an agent interacting with an environment. The environment is represented through a set of states. The agent takes actions and receives feedback from the environment in terms of a reward function. The goal of the agent is to learn the best action in each state in order to maximize the expected cumulative reward in the long run. Reinforcement learning is formalized using the theory of discrete-time stochastic control, also known as Markov Decision Processes (MDP).

Markov Decision Processes

A (discrete) **Markov chain** is a discrete stochastic process $\{X_t\}_{t\in\mathbb{N}}$, where each X_t depends only on the previous random variable. Formally,

$$\mathbb{P}(X_{t+1} = s_{t+1} \mid X_t = s_t, \ldots, X_1 = s_1) = \mathbb{P}(X_{t+1} = s_{t+1} \mid X_t = s_t).$$

If the random variables in the Markov chain take values in a **finite state space** $\mathcal{S} = \{1, \ldots, k\}$, then for each t we have a $k \times k$ transition matrix $\boldsymbol{P}_t$, whose values p_{ij}^t are given by

$$p_{ij}^t = \mathbb{P}(X_{t+1} = j \mid X_t = i).$$

A Markov chain is called *stationary* if this transition matrix is independent of t. Markov chains can model the changes in an environment over time. The only restriction is that the next state only depends on the current state and not the previous states; the future depends only on the present, and not the past. In Markov decision processes, the transition probability is not only determined by the previous state, but also by the action of an agent. In what follows, we denote by $\Delta(\mathcal{S})$ the set of probability distributions on a set $\mathcal{S}$.

Definition 25.1. A finite **Markov Decision Process** is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where

1. $\mathcal{S}$ is a finite set of **states**;
2. $\mathcal{A}$ is a finite set of **actions**;
3. $\mathcal{P}$ is a **transition kernel** $\mathcal{P}\colon \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$;
4. $\mathcal{R}$ is a **reward function** $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$;
5. $\gamma \in [0, 1]$ is a **discount factor**.

As is customary, we write $p(s' \mid s, a) := \mathcal{P}(s, a)(s')$ to suggest conditional dependence. Given a state, an action $a \in \mathcal{A}$ is typically chosen according to a stochastic **policy** $\pi \colon \mathcal{S} \to \Delta(\mathcal{A})$, and we write $\pi(a \mid s)$ for the probability of policy a given state s. Given an initial distribution $\mu_0 \in \Delta(\mathcal{S})$, an initial action $A_0 \in \mathcal{A}$ and a policy π, we thus get stochastic processes S_t and A_t that satisfy

$$\mathbb{P}(S_t = s' \mid S_{t-1} = s, A_{t-1} = a) = P(s' \mid s, a)$$
$$\mathbb{P}(A_t = a \mid S_t = s) = \pi(a \mid s)$$

for $t \geq 1$. In addition, we get a reward process

$$R_t = \mathcal{R}(S_{t-1}, A_{t-1}, S_t)$$

and an expected reward given states s and a,

$$r(s, a) := \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] = \sum_{s' \in \mathcal{S}} p(s' \mid s, a)\mathcal{R}(s, a, s'). \tag{25.1}$$

The interpretation is that at each point in time we are in a given state S_t, select an action A_t and move to a new state S_{t+1}, obtaining a reward R_{t+1}. Schematically, we can represent an MDP as in Figure (25.1).

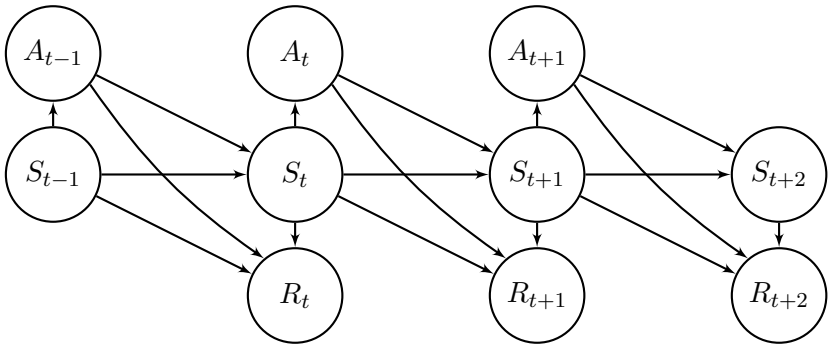


Figure 25.1: A Markov Decision Process as a graphical model

So far we looked at the reward function at one point in time, but in applications we want to maximize the cumulative reward, given by

$$G_t := \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

where $\gamma \in [0, 1]$ is the discounting factor introduced in the definition of a Markov decision model. Note that we have the recursion

$$G_t = R_{t+1} + \gamma G_{t+1}.$$

The discount γ represents the present value of future rewards, and is a common feature in finance. There are also practical benefits of including a discount factor, as it avoids infinite sums in cyclic Markov chains. We set $G := G_0$ and will often without loss of generality assume $t = 0$. Sometimes we consider an upper limit T for the sum. A segment of agent-environment interaction of finite length is called an *episode*. After an episode, we may reset the state to a staring configuration and start again.

Given a policy π, an initial state s and action a at time $t = 0$, we can define the *state value* function $v^\pi(s)$ and the *value-action* (or q) function q^π with respect to a policy as

$$v^\pi(s) = \mathbb{E}^\pi[G \mid S_0 = s]$$
$$q^\pi(s, a) = \mathbb{E}^\pi[G \mid S_0 = s, A_0 = a]$$

where by $\mathbb{E}^\pi$ we denote the probability distribution arising from choosing a policy π. Thus v^π measures the value of state s, while q^π measures the value of taking action a while in state s. Note that these quantities are related as follows:

$$v^\pi(s) = \mathbb{E}^\pi_{a\sim\pi(s)}[q^\pi(s,a)]. \tag{25.2}$$

An important property of value functions is that they satisfy recursive relationships. These are the well known **Bellman Equations**.

Theorem 25.2. *(Bellman equations) Given an MDP with policy π, a state s and an action a, the value functions satisfy the relations*

$$\begin{aligned} v^\pi(s) &= \mathbb{E}^\pi_{A\sim\pi(s)}[r(s,A) + \mathbb{E}_{S\sim\mathcal{P}(s,A)}[\gamma v^\pi(S)]] \\ q^\pi(s,a) &= r(s,a) + \gamma\,\mathbb{E}_{S\sim\mathcal{P}(s,a)}[\mathbb{E}^\pi_{A\sim\pi(S)}[q^\pi(S,A)]], \end{aligned}$$

where $r(s,a)$ is the expected reward (25.1)*.*

The Bellman equations for the state value function expresses that the value of the present state is equal to the expected value of the next state, plus the reward expected along the way.

Proof. We prove the theorem for $q^\pi(s,a)$, the other identities follow along the lines. Note that

$$\begin{aligned} q^\pi(s,a) &= \mathbb{E}^\pi[R_1 + \gamma G_1 \mid S_0 = s, A_0 = a] \\ &\overset{(1)}{=} \mathbb{E}_{S\sim\mathcal{P}(s,a)}\,\mathbb{E}^\pi[\mathcal{R}(s,a,S) + \gamma G_1 \mid S_0 = s] \\ &= \mathbb{E}_{S\sim\mathcal{P}(s,a)}[\mathcal{R}(s,a,S) + \gamma\,\mathbb{E}^\pi[G_1 \mid S_0 = s]] \\ &= \mathbb{E}_{S\sim\mathcal{P}(s,a)}[\mathcal{R}(s,a,S) + \gamma v^\pi(S)] \\ &= r(s,a) + \gamma\,\mathbb{E}_{S\sim\mathcal{P}(s,a)}[\mathbb{E}^\pi_{A\sim\pi(S)}[q^\pi(S,A)]], \end{aligned}$$

where for (1) we used the fact that a only enters into the first summand of G. The claim now follows from (25.2). □

Note that since we assumed that the state and action spaces are finite, all the expected values encountered can be written out as sums. In particular, the Bellman equations for the value functions can be expressed as

$$\begin{aligned} v^\pi(s) &= \sum_{a\in\mathcal{A}} \pi(a \mid s) \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s' \mid s,a) v^\pi(s') \right) \\ &= \sum_{a\in\mathcal{A}} \pi(a \mid s) \sum_{s'\in\mathcal{S}} p(s' \mid s,a) \left(\mathcal{R}(s,a,s') + \gamma v^\pi(s') \right), \\ q^\pi(s,a) &= \sum_{s'\in\mathcal{S}} p(s' \mid s,a) \left(\mathcal{R}(s,a,s') + \gamma \sum_{a'\in\mathcal{A}} \pi(a \mid s) q^\pi(s',a') \right). \end{aligned}$$

Assume $\mathcal{S} = \{1,\dots,n\}$ and set $v_i = v^\pi(i)$ and $\boldsymbol{v}^\pi = (v_1,\dots,v_n)^T$. Define the matrix $\boldsymbol{P}^\pi$ and vector $\boldsymbol{r}^\pi$ as

$$\begin{aligned} \boldsymbol{P}^\pi = (p_{ij}), \quad p_{ij} &= \sum_{a\in\mathcal{A}} \pi(a \mid i) \cdot p(j \mid i,a) \\ \boldsymbol{r}^\pi = (r_i), \quad r_i &= \sum_{a\in\mathcal{A}} \pi(a \mid i) \cdot r(s,i). \end{aligned}$$

Then the Bellman equation for the $v^\pi(s)$ can be written in terms of linear algebra and solved as

$$\boldsymbol{v}^\pi = \boldsymbol{r}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{v} \quad \Rightarrow \quad \boldsymbol{v}^\pi = (\mathbf{1} - \gamma \boldsymbol{P}^\pi)^{-1} \boldsymbol{r}^\pi.$$

We can get a similar characterization for the Bellman equations for $q^\pi(s, a)$.

The goal in reinforcement learning is to find a policy that maximises these functions, i.e., to find

$$q^*(s, a) = \max_\pi q^\pi(s, a), \quad v^*(s) = \max_\pi v^\pi(s),$$

where the maximum is over a suitable set of policies. We denote an optimal policy by π^*. It can be shown that such a policy always exists, but it is not obvious how to find one. It turns out that the optimal values also satisfy a system of equations, the Bellman optimality conditions.

Theorem 25.3. *(Bellman Optimality Conditions) The optimal values satisfy*

$$v^*(s) = \max_{a \in \mathcal{A}} r(s, a) + \gamma \, \mathbb{E}_{S \sim \mathcal{P}(s,a)}[v^*(S)]$$
$$q^*(s, a) = r(s, a) + \gamma \, \mathbb{E}_{S \sim \mathcal{P}(s,a)}[\max_{a'} q^*(S, a)]$$

While we were able to solve the Bellman equations for the state-value and the value-action functions exactly, this is no longer possible for the optimal values.

Dynamic Programming

We first consider the problem of evaluating a policy π, that is, determining v^π for all states s. The general idea is simple: begin with a value vector $\boldsymbol{v}_0$ and then iterate according to the Bellman equation,

$$\boldsymbol{v}^{k+1} = T^\pi(\boldsymbol{v}^k) := \boldsymbol{r}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{v}^k$$

for $k \geq 0$. The operator T^π is known as the **Bellman operator** for **value iteration**. The information needed for this procedure are the policy values $\pi(a \mid i)$, the transition probabilities $p(j \mid i, a)$ and the expected rewards $r(s, i)$. These parameters are often known by design. We know show that this sequence of vectors converges to $\boldsymbol{v}^\pi$. This is based on a fundamental result known as the Banach Fixed-Point Theorem. We call a continuous map T from a metric space to itself a γ-contraction, if $\|T(\boldsymbol{u}) - T(\boldsymbol{v})\| \leq \gamma \|\boldsymbol{u} - \boldsymbol{v}\|$ for all $\boldsymbol{u}, \boldsymbol{v}$ in that space.

Theorem 25.4. *(Banach Fixed-Point Theorem) Let V be a Banach space and let $T \colon V \to V$ be a γ-contraction for some $\gamma \in [0, 1)$. Then there exists a fixed point $\boldsymbol{v}^* \in V$ with $T(\boldsymbol{v}^*) = \boldsymbol{v}^*$. Moreover, for any $\boldsymbol{v}^0 \in V$, the sequence $\{\boldsymbol{v}^k\}_k$ defined by*

$$\boldsymbol{v}^{k+1} = T(\boldsymbol{v}^k)$$

for $k \geq 0$ converges to $\boldsymbol{v}^*$.

Proposition 25.5. *For an MDP and a policy π, the Bellman operator for value iteration is a γ-contraction,*

$$\|T^\pi(\boldsymbol{u}) - T^\pi(\boldsymbol{v})\|_\infty \leq \gamma \|\boldsymbol{u} - \boldsymbol{v}\|_\infty.$$

In particular, for $\gamma \in [0, 1)$ the sequence $\{\boldsymbol{v}^k\}_k$ defined by iterating the Bellman operator converges to $\boldsymbol{v}^\pi$.

Proof. Note that

$$\begin{aligned}\|T^\pi(\boldsymbol{u}) - T^\pi(\boldsymbol{v})\|_\infty &= \|\boldsymbol{r}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{u} - \boldsymbol{r}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{v}\|_\infty \\ &= \gamma \|\boldsymbol{P}^\pi(\boldsymbol{u} - \boldsymbol{v})\|_\infty \\ &\leq \gamma \|\boldsymbol{P}^\pi\|_\infty \|\boldsymbol{u} - \boldsymbol{v}\|_\infty \leq \gamma \|\boldsymbol{u} - \boldsymbol{v}\|_\infty,\end{aligned}$$

since the matrix $\boldsymbol{P}^\pi$ has entries bounded by 1. The second claim follows from the Banach Fixed-Point Theorem 25.4. □

Once the value of a policy is determined, we aim to change the policy in order to improve the value. We can find a better policy by maximizing the state-action function,

$$\pi'(s) \in \arg\max_{a \in \mathcal{A}} q^\pi(s, a).$$

If we assume a deterministic policy, that is, $a = \pi(s)$, then it can be shown that

$$v^\pi(s) = q^\pi(s, \pi(s)) \leq q^\pi(s, \pi'(s)) \leq v^{\pi'}(s)$$

and we get an improvement in value, with equality if and only if $v^\pi(s) = v^*(s)$ is the optimal value. This gives and algorithm for finding an optimal policy π for an MDP.

Example 25.6. Consider the game Gridworld. The playing field is an 4×4 grid and each of the 16 cells represents a state. On each cell (state), four actions are possible: north, south, east, west. These actions cause the state to change to one adjacent to it according to the direction specified. Actions that would take the agent off the grid result in a reward of -1 and leave the state unchanged. Suppose we label two cells as special terminal states, say A and B. Starting at any state, the agent moves around the board according to a policy and wins if it ends up in A, while it loses if it ends up in B. The goal is to find a policy that will guide the agent to A while avoiding B.

There are similar approaches to approximating the optimal value function v^* and the optimal value function q^* by iterating the corresponding Bellman equations.

Notes

The theory of Markov Decision Processes was developed in pioneering work by Richard Bellman [6, 7] and Ronald Howard [41]. The continuous analogue of the theory is known as stochastic optimal control, and has found applicants in many areas, including engineering and finance. A comprehensive reference for the modern theory of reinforcement learning is the textbook [76]. Deep reinforcement learning, that is the combination of reinforcement learning with deep neural networks, has become popular through the work by DeepMind on groundbreaking applications such as AlphaGo, AlphaZero [74] and AlphaTensor [26].

A

Mathematical Background

"It's not possible to know too much. It's not possible to have too much technique. I'm a big believer in the idea of headroom."

— Guthrie Govan

This appendix contains background material from various field of mathematics. Most of the material should be familiar to advanced undergraduate students in mathematics. The section on finite-precision arithmetic covers material that is not commonly encountered in standard undergraduate courses. This part may be read lightly: its purpose is merely to raise awareness about the intricacies and subtleties of numerical computation (after all, most machine learning algorithms employ numerical, as opposed to symbolic, computation). Important concepts are **highlighted**.

1 Asymptotic notation

An **algorithm**[1] is a sequence of instructions carried out by a computer. Important features of an algorithm are computation time (measured as the number of operations or the number of iterations needed to reach a solution) and accuracy. One is mainly interested in the **orders of magnitude** of these quantities, and not so much in their exact values. A convenient notation for this purpose is the asymptotic O-notation.

Let $f, g\colon \mathbb{R} \to \mathbb{R}$ be two functions. Then:

- $f(n) \in O(g(n))$ as $n \to \infty$ if there exists a constant $C > 0$ and an integer n_0 such that $|f(n)| \leq C|g(n)|$ for $n > n_0$;
- $f(x) \in O(g(x))$ as $x \to 0$ if there exists a constant $C > 0$ and a real number $\varepsilon > 0$ such that $|f(x)| \leq C|g(x)|$ for $|x| < \varepsilon$.

We omit "$n \to \infty$" or "$x \to 0$" when it is clear from the context. One often finds statements such as $f(n) = O(g(n))$ or $f(x) = 1 + x + O(x^2)$; the first is equivalent to $f(n) \in O(g(n))$, while the second means $f(x) = 1 + x + g(x)$ for a function $g(x) \in O(x^2)$. The following examples illustrate this:

- $\sqrt{n} + n^2 \in O(n^2)$
- $10^{100} \in O(1)$
- $\sin(x) \in O(x)$
- $n^5 \in O(e^n)$
- $x^3 \in O(x^2)$
- $e^x = 1 + x + O(x^2)$

[1] For our purposes it is not necessary to get into the formal definition in terms of Turing machines.

The notation $f(n) \in \Omega(g(n))$ as $n \to \infty$ means that $g(n) \in O(f(n))$ as $n \to \infty$, and $f(n) \in o(g(n))$ as $n \to \infty$ means that for *all* $C > 0$ there exists n_0 such that $|f(n)| < C|g(n)|$ for $n > n_0$. If $g(n) \neq 0$ for sufficiently large n, this is equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. One defines $\Omega(g(x))$ and $o(g(x))$ as $x \to 0$ analogously.

2 Linear Algebra

We restrict to linear algebra over the field of real numbers $\mathbb{R}$, as this is the setting that is of most interest in optimization. A **vector** in $\mathbb{R}^n$ and its **transpose** are written as

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = (x_1, \ldots, x_n)^\intercal, \quad \boldsymbol{x}^\intercal = (x_1, \ldots, x_n),$$

with **coordinates** $x_i \in \mathbb{R}$ for $1 \leq i \leq n$. The zero vector is denoted by $\mathbf{0}$, while $\boldsymbol{e}$ is the vector with every coordinate equal to 1. If $\lambda_1, \lambda_2 \in \mathbb{R}$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, then $\lambda_1\boldsymbol{x} + \lambda_2\boldsymbol{y}$ is the vector with coordinates $\lambda_1 x_i + \lambda_2 y_i$ for $1 \leq i \leq n$.

In $\mathbb{R}^n$ we have the Euclidean (or standard) **inner product** (or scalar product)

$$\boldsymbol{x}^\intercal \boldsymbol{y} = \sum_{i=1}^{n} x_i y_i.$$

The Euclidean inner product satisfies $\langle \boldsymbol{x}, \boldsymbol{x} \rangle \geq 0$, with equality if and only if $\boldsymbol{x} = \mathbf{0}$. Moreover, it is **symmetric** ($\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \langle \boldsymbol{y}, \boldsymbol{x} \rangle$) and **bilinear**: for $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^n$ and $\alpha, \beta \in \mathbb{R}$,

$$\langle \alpha\boldsymbol{x}_1 + \beta\boldsymbol{x}_2, \boldsymbol{y} \rangle = \alpha\langle \boldsymbol{x}_1, \boldsymbol{y} \rangle + \beta\langle \boldsymbol{x}_2, \boldsymbol{y} \rangle.$$

Two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are called **orthogonal**, if $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = 0$.

Example A.1. The vectors $(1, 1)^\intercal$ and $(1, -1)^\intercal$ are orthogonal in $\mathbb{R}^2$, while $(1, 1)^\intercal$ and $(2, -1)^\intercal$ are not.

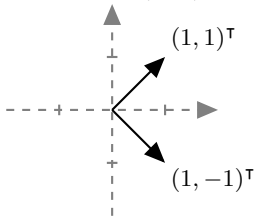


Figure A.1: Orthogonal vectors

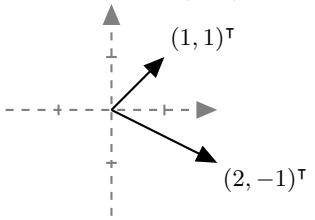


Figure A.2: Non-orthogonal vectors

Linear subspaces

A **linear subspace** is a subset $V \subseteq \mathbb{R}^n$ such that for any $\boldsymbol{x}, \boldsymbol{y} \in V$ and for all $\alpha, \beta \in \mathbb{R}$, $\alpha\boldsymbol{x} + \beta\boldsymbol{y} \in V$. In particular, the sets $\{\mathbf{0}\}$ and $\mathbb{R}^n$ are linear subspaces.

Example A.2. The linear subspaces of $\mathbb{R}^2$ are $\{\mathbf{0}\}$, lines through the origin, and $\mathbb{R}^2$. The linear subspaces of $\mathbb{R}^3$ are $\{\mathbf{0}\}$, lines and planes through the origin, and $\mathbb{R}^3$.

A **linear combination** of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k \in \mathbb{R}^n$ is an expression of the form $\boldsymbol{x} = \sum_{i=1}^{k} \lambda_i \boldsymbol{x}_i$, where $\lambda_i \in \mathbb{R}$ for $1 \leq i \leq k$. The set of linear combinations

$$V = \operatorname{span}\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\} := \left\{ \sum_{i=1}^{k} \lambda_i \boldsymbol{x}_i \mid \lambda_i \in \mathbb{R} \right\}$$

forms a linear subspace of $\mathbb{R}^n$. It is the intersection of all linear subspaces that contain $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$. The vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ are **linearly independent** if $\sum_{i=1}^k \lambda_i \boldsymbol{x}_i = 0$ implies $\lambda_1 = \cdots = \lambda_k = 0$. A minimal set of vectors that span a linear subspace V is called a **basis** of this subspace, and the number of elements in a basis is the **dimension** of the linear subspace. The elements of a basis are always linearly independent, and a maximal linearly independent set in a vector subspace V is a basis. If $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k\}$ is a basis of a subspace V, then every $\boldsymbol{x} \in V$ has a *unique* representation $\boldsymbol{x} = \sum_{i=1}^k \lambda_i \boldsymbol{b}_i$. A basis is **orthogonal** if $\langle \boldsymbol{b}_i, \boldsymbol{b}_j \rangle = 0$ for $i \neq j$, and **orthonormal** if in addition $\langle \boldsymbol{b}_i, \boldsymbol{b}_i \rangle = 1$ for $1 \leq i \leq k$. The unique expression of $\boldsymbol{x} \in V$ as linear combination of an orthonormal basis $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k\}$ of V is given by

$$\boldsymbol{x} = \sum_{i=1}^{k} \langle \boldsymbol{x}, \boldsymbol{b}_i \rangle \boldsymbol{b}_i.$$

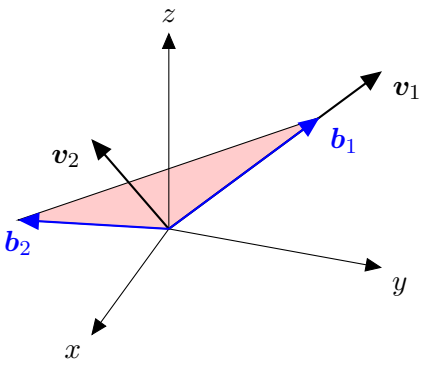
The **standard basis** of $\mathbb{R}^n$ is the orthonormal basis $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n\}$, where $\boldsymbol{e}_i$ has a 1 in the i-th coordinate and 0 elsewhere.

Example A.3. The vectors $\boldsymbol{v}_1 = (0, 1, 1)^\intercal$ and $\boldsymbol{v}_2 = (1, 0, 1)^\intercal$ span a linear subspace of $\mathbb{R}^3$ of dimension 2, but they are not orthogonal. The vectors

$$\boldsymbol{b}_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \boldsymbol{b}_2 = \frac{1}{\sqrt{3}} \begin{pmatrix} \sqrt{2} \\ -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

form an orthonormal basis of V. The vector $\boldsymbol{x} = (1, 1, 2)^\intercal$ lives in V, and its representation in terms of $\{\boldsymbol{b}_1, \boldsymbol{b}_2\}$ is

$$\boldsymbol{x} = \frac{3}{\sqrt{2}} \boldsymbol{b}_1 + \sqrt{\frac{3}{2}} \boldsymbol{b}_2.$$



The **direct sum** of two vector subspaces $V, W \subset \mathbb{R}^n$ with $V \cap W = \{\boldsymbol{0}\}$ is

$$V \oplus W = \{\boldsymbol{v} + \boldsymbol{w} \mid \boldsymbol{v} \in V, \boldsymbol{w} \in W\}.$$

The **orthogonal complement** of a subspace $V \subseteq \mathbb{R}^n$ is the set

$$V^\perp = \{\boldsymbol{x} \in \mathbb{R}^n \mid \forall \boldsymbol{y} \in V \colon \langle \boldsymbol{x}, \boldsymbol{y} \rangle = 0\}.$$

The vector space $\mathbb{R}^n$ is the direct sum of V and its orthogonal complement,

$$\mathbb{R}^n = V \oplus V^\perp. \tag{A.1}$$

If $V = \operatorname{span}\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$, then $V^\perp = \{\boldsymbol{y} \mid \langle \boldsymbol{x}_1, \boldsymbol{y}\rangle = \cdots \langle \boldsymbol{x}_k, \boldsymbol{y}\rangle = 0\}$; to check whether a vector $\boldsymbol{y}$ is in the orthogonal complement of V we therefore only need to check whether $\boldsymbol{y}$ is orthogonal to a spanning set (for example, a basis) of V.

Example A.4. The vector $\boldsymbol{x} = (-1, -1, 1)^\intercal$ is orthogonal to the basis $\{\boldsymbol{b}_1, \boldsymbol{b}_2\}$ from Example A.3. It is therefore orthogonal to the whole plane $V = \operatorname{span}\{\boldsymbol{b}_1, \boldsymbol{b}_2\}$ spanned by these vectors. The orthogonal complement of V is the line $\{\lambda \boldsymbol{x} \mid \lambda \in \mathbb{R}\}$.

The **direct product** of two vector subspaces $V \subseteq \mathbb{R}^n$, $W \subseteq \mathbb{R}^m$ is defined as

$$V \times W = \{(\boldsymbol{v}, \boldsymbol{w}) \in \mathbb{R}^{n+m} \mid \boldsymbol{v} \in V, \boldsymbol{w} \in W\}.$$

where $(\boldsymbol{v}, \boldsymbol{w})$ is the vector whose first n coordinates coincide with $\boldsymbol{v}$, and the last m coordinates coincide with $\boldsymbol{w}$. In particular, $\mathbb{R}^n \times \mathbb{R}^m = \mathbb{R}^{n+m}$.

Linear maps

An $m \times n$ **matrix**

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix},$$

represents a **linear map** from $\mathbb{R}^n$ to $\mathbb{R}^m$ by means of

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}, \quad y_i = \sum_{j=1}^{n} a_{ij} x_j.$$

For example,

$$\begin{pmatrix} 2 & 1 & 0 \\ 1 & 0 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}.$$

The columns of a matrix are vectors, and we sometimes write

$$\boldsymbol{A} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)$$

for the matrix whose columns are given by the vectors $\boldsymbol{a}_i$. If $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ and $n = m + k$, then $\boldsymbol{A}$ can be written as **block matrix**,

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{pmatrix},$$

with $\boldsymbol{A}_{11} \in \mathbb{R}^{m \times m}$, $\boldsymbol{A}_{22} \in \mathbb{R}^{k \times k}$, $\boldsymbol{A}_{12} \in \mathbb{R}^{m \times k}$ and $\boldsymbol{A}_{21} \in \mathbb{R}^{k \times m}$. The sum and difference of matrices of the same size are defined component-wise.

The $n \times n$ matrix $\mathbf{1}$ is the matrix with 1 on the diagonal and 0 elsewhere, while $\mathbf{0}$ is the matrix consisting of only zeros. A matrix is **diagonal** if all the off-diagonal elements are 0, **lower-triangular** if all the elements above the diagonal are 0, and **upper-triangular** if all the elements below the diagonal are 0. A **block-diagonal** matrix is a block matrix, with all blocks outside the main diagonal consisting of zero-matrices $\mathbf{0}$.

The **transpose** $\boldsymbol{A}^\intercal$ is the matrix with entries $a'_{ij} := a_{ji}$. It is the matrix $\boldsymbol{A}$ mirrored on the diagonal from top left to bottom right. A matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is called **symmetric** if $\boldsymbol{A}^\intercal = \boldsymbol{A}$. The set of symmetric matrices in $\mathbb{R}^{n \times n}$ is denoted by $\mathcal{S}^n$.

The **product** of an $m \times p$ matrix $\boldsymbol{A}$ with a $p \times n$ matrix $\boldsymbol{B}$,

$$\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B},$$

is the $m \times n$ matrix $\boldsymbol{C}$ whose (i, j)-th entry is given by

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}.$$

It represents a composition of maps $\mathbb{R}^n \to \mathbb{R}^p \to \mathbb{R}^m$. The number of columns of $\boldsymbol{A}$ has to equal the number of rows of $\boldsymbol{B}$ for this definition to make sense. Products of block matrices or of block matrices with vectors can be carried out block-wise. If, for example, $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)^\intercal$ with $\boldsymbol{x}_1 \in \mathbb{R}^{1\times m}$ and $\boldsymbol{x}_2 \in \mathbb{R}^{1\times k}$, then

$$\begin{pmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_{11}\boldsymbol{x}_1 + \boldsymbol{A}_{12}\boldsymbol{x}_2 \\ \boldsymbol{A}_{21}\boldsymbol{x}_1 + \boldsymbol{A}_{22}\boldsymbol{x}_2 \end{pmatrix}.$$

The matrix $\mathbf{1}$ satisfies $\mathbf{1}\boldsymbol{A} = \boldsymbol{A}$ and $\boldsymbol{A}\mathbf{1} = \boldsymbol{A}$, whenever the dimensions are such that this is defined. In general, even if $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$, $\boldsymbol{A}\boldsymbol{B} \neq \boldsymbol{B}\boldsymbol{A}$.

Example A.5. Let

$$\boldsymbol{A} = \begin{pmatrix} 1 & 2 \\ 1 & 4 \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} 2 & 3 \\ 3 & 2 \end{pmatrix}.$$

Then

$$\boldsymbol{A}\boldsymbol{B} = \begin{pmatrix} 8 & 7 \\ 14 & 11 \end{pmatrix}, \quad \boldsymbol{B}\boldsymbol{A} = \begin{pmatrix} 5 & 16 \\ 5 & 14 \end{pmatrix}.$$

If we consider a vector $\boldsymbol{x} \in \mathbb{R}^n$ as an $n \times 1$ matrix and the transpose as an $1 \times n$ matrix, then for $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n \cong \mathbb{R}^{n\times 1}$ we have

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x}^\intercal \boldsymbol{y}.$$

The transpose of a product satisfies $(\boldsymbol{A}\boldsymbol{B})^\intercal = \boldsymbol{B}^\intercal \boldsymbol{A}^\intercal$. From this it follows that for any matrix, $\boldsymbol{A}^\intercal \boldsymbol{A}$ is symmetric. For any matrix $\boldsymbol{A}$ we have

$$\langle \boldsymbol{x}, \boldsymbol{A}\boldsymbol{x} \rangle = \boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{x} = (\boldsymbol{A}^\intercal \boldsymbol{x})^\intercal \boldsymbol{x} = \langle \boldsymbol{A}^\intercal \boldsymbol{x}, \boldsymbol{x} \rangle.$$

It follows from this that if a matrix is symmetric, then it is also self-adjoint, which means that $\langle \boldsymbol{x}, \boldsymbol{A}\boldsymbol{x} \rangle = \langle \boldsymbol{A}\boldsymbol{x}, \boldsymbol{x} \rangle$.

The **rank** of a matrix $\boldsymbol{A}$, $\mathrm{rk}(\boldsymbol{A})$, is the maximum number of linearly independent rows or columns of $\boldsymbol{A}$. The **kernel** and **image** of $\boldsymbol{A}$ are the linear subspaces

$$\ker \boldsymbol{A} := \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{x} = 0\}, \quad \operatorname{im} \boldsymbol{A} = \{\boldsymbol{A}\boldsymbol{x} \mid \boldsymbol{x} \in \mathbb{R}^n\}.$$

The dimensions are given by $\dim \ker \boldsymbol{A} = n - \mathrm{rk}(\boldsymbol{A})$ and $\dim \operatorname{im} \boldsymbol{A} = \mathrm{rk}(\boldsymbol{A})$. While $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ represents a linear map from $\mathbb{R}^n$ to $\mathbb{R}^m$, the transpose $\boldsymbol{A}^\intercal$ represents a map in the other direction, and the image of $\boldsymbol{A}^\intercal$ coincides with the orthogonal complement of the kernel of $\boldsymbol{A}$, $(\ker \boldsymbol{A})^\perp = \operatorname{im} \boldsymbol{A}^\intercal$. In particular, in view of (A.1) we have the direct sum decomposition

$$\mathbb{R}^n = \ker \boldsymbol{A} \oplus \operatorname{im} \boldsymbol{A}^\intercal.$$

A **system of linear equations**

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1n}x_n &= b_1 \\ \vdots \qquad\qquad \vdots \qquad\quad &\vdots \\ a_{m1}x_1 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

is written as a matrix vector product

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{A.2}$$

where the $m \times n$ matrix $\boldsymbol{A}$ is defined as above, and $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{b} \in \mathbb{R}^m$. If the columns of $\boldsymbol{A}$ are linearly independent, then the system of equations can have at most one solution, and otherwise it has infinitely many solutions (this is the case if $n > m$). If $n = m$, then the system (A.2) has a unique solution if and only if the matrix $\boldsymbol{A}$ is **invertible** or **non-singular**. This is the case if the rows of $\boldsymbol{A}$ (or equivalently, the columns of $\boldsymbol{A}$) are linearly independent. If $\boldsymbol{A}$ is not invertible, it is called **singular**.

If $\boldsymbol{A}$ is invertible, there exists a matrix $\boldsymbol{A}^{-1}$ (the **inverse**) such that

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \mathbf{1}.$$

The solution of (A.2) is then given by $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$. The following conditions on a matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ are equivalent:

1. $\boldsymbol{A}$ is invertible,
2. $\operatorname{rk}(\boldsymbol{A}) = n$,
3. $\ker \boldsymbol{A} = \{\mathbf{0}\}$,
4. $\operatorname{im} \boldsymbol{A} = \mathbb{R}^n$,
5. the rows of $\boldsymbol{A}$ are linearly independent,
6. the columns of $\boldsymbol{A}$ are linearly independent,
7. $\det(\boldsymbol{A}) \neq 0$,

where the determinant is

$$\det(\boldsymbol{A}) = \sum_{\sigma\in S_n} \operatorname{sgn}(\sigma) a_{1\sigma(1)} \cdots a_{n\sigma(n)},$$

and S_n is the group of permutations of $[n] = \{1, \dots, n\}$, with $\operatorname{sgn}(\sigma)$ the sign of the permutation (parity of the number of inversions).

Example A.6. For two- and three-dimensional matrices

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix},$$

the determinants are

$$\begin{aligned} \det(\boldsymbol{A}) &= a_{11}a_{22} - a_{12}a_{21}, \\ \det(\boldsymbol{B}) &= a_{11}(a_{22}a_{33} - a_{23}a_{32}) - a_{12}(a_{21}a_{33} - a_{23}a_{31}) + a_{13}(a_{21}a_{32} - a_{22}a_{31}). \end{aligned}$$

A matrix $\boldsymbol{Q}$ is **orthogonal** if $\boldsymbol{Q} = (\boldsymbol{q}_1, \dots, \boldsymbol{q}_n)$, with $\langle \boldsymbol{q}_i, \boldsymbol{q}_j \rangle = \delta_{ij}$, and

$$\delta_{ij} = \begin{cases} 0 & \text{if } i \neq j, \\ 1 & \text{if } i = j. \end{cases}$$

As the (i, j)-th entry of $\boldsymbol{Q}^\intercal\boldsymbol{Q}$ is given by $\langle \boldsymbol{q}_i, \boldsymbol{q}_j \rangle$, the orthogonality of $\boldsymbol{Q}$ can succinctly be characterized by the requirement $\boldsymbol{Q}^\intercal\boldsymbol{Q} = \mathbf{1}$. In particular, $\boldsymbol{Q}^\intercal = \boldsymbol{Q}^{-1}$, and the columns (and rows) of an orthogonal

matrix form an orthonormal basis of $\mathbb{R}^n$. Orthogonal matrices have the property that $\langle \boldsymbol{Q}\boldsymbol{x}, \boldsymbol{Q}\boldsymbol{y}\rangle = \langle \boldsymbol{x}, \boldsymbol{y}\rangle$. From this it follows that orthogonality of vectors is preserved under orthogonal transformations. The determinant of an orthogonal matrix is $\det(\boldsymbol{Q}) = 1$ (but $\det(\boldsymbol{A}) = 1$ does not imply that $\boldsymbol{A}$ is orthogonal). As the product of orthogonal matrices is again orthogonal, the set of orthogonal $n \times n$ matrices forms a group, commonly denoted by $O(n)$.

Example A.7. Consider the three matrices,

$$\boldsymbol{A} = \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} 2 & 3 \\ 3 & 2 \end{pmatrix}, \quad \boldsymbol{C} = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}.$$

The matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are symmetric, while $\boldsymbol{C}$ is not. The matrices $\boldsymbol{B}$ and $\boldsymbol{C}$ are invertible, with inverse

$$\boldsymbol{B}^{-1} = \begin{pmatrix} -0.4 & 0.6 \\ 0.6 & -0.4 \end{pmatrix}, \quad \boldsymbol{C}^{-1} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}.$$

The matrix $\boldsymbol{A}$ is not invertible, since the second column is a multiple of the first. The kernel of $\boldsymbol{A}$ the linear span of $(-2, 1)^\intercal$. The matrix $\boldsymbol{C}$ is orthogonal (this can be seen by checking that the columns or rows are orthonormal, or looking at the expression of the inverse above).

Eigenvalues

A vector $\boldsymbol{u} \neq \boldsymbol{0}$ is an **eigenvector** of $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, if there exists a $\lambda \in \mathbb{C}$ such that

$$\boldsymbol{A}\boldsymbol{u} = \lambda \boldsymbol{u}.$$

Such a number λ is called an **eigenvalue** of $\boldsymbol{A}$. Note that the eigenvectors are only defined up to scaling: if $\boldsymbol{u}$ is an eigenvector, then so is $\lambda\boldsymbol{u}$ for any non-zero $\lambda \in \mathbb{R}$.

From the definition of the determinant, the function $\lambda \mapsto \det(\lambda\boldsymbol{1} - \boldsymbol{A})$ is a polynomial of degree at most n, called the **characteristic polynomial** of $\boldsymbol{A}$. The eigenvalues are the roots of this polynomial,

$$\det(\lambda\boldsymbol{1} - \boldsymbol{A}) = 0.$$

The eigenvalues can be complex numbers, and appear in complex conjugate pairs. If the matrix $\boldsymbol{A}$ is symmetric, then the eigenvalues are all real numbers. Two important quantities, the **determinant** and the **trace** of a matrix (corresponding, up to sign, to the highest and lowest coefficient of the characteristic polynomial) can be expressed in terms of the eigenvalues:

$$\det(\boldsymbol{A}) = \lambda_1 \cdots \lambda_n, \quad \operatorname{trace}(\boldsymbol{A}) := a_{11} + \cdots + a_{nn} = \lambda_1 + \cdots + \lambda_n.$$

A matrix has a zero eigenvalue if and only if it is singular. Eigenvalues may occur with multiplicity.

Norms

A **norm** in $\mathbb{R}^n$ is a function $\|\cdot\|$ that satisfies the following three properties

1. $\|\boldsymbol{x}\| \geq 0$ for all $\boldsymbol{x} \in \mathbb{R}^n$, and $\boldsymbol{x} = 0$ if and only if $\boldsymbol{x} = \boldsymbol{0}$;
2. $\|\lambda\boldsymbol{x}\| = |\lambda|\|\boldsymbol{x}\|$ for $\lambda \in \mathbb{R}$ and $\boldsymbol{x} \in \mathbb{R}^n$;
3. $\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$ for $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$.

Three important examples of norms are the following:

1. The 1-norm: $\|\boldsymbol{x}\|_1 = \sum_{i=1}^n |x_i|$;
2. The 2-norm: $\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$;
3. The ∞-norm: $\|\boldsymbol{x}\|_\infty = \max_{1\leq i\leq n} |x_i|$.

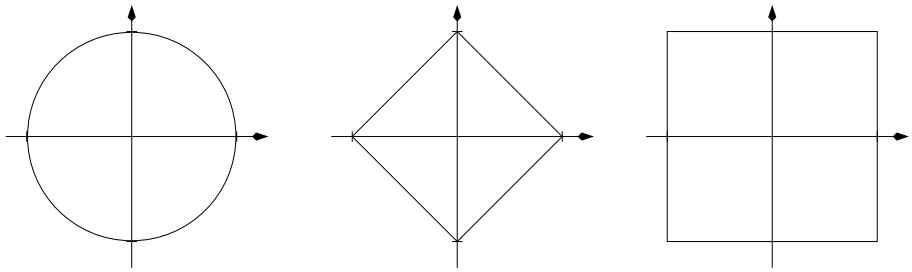
Example A.8. Let $\boldsymbol{x} = (2, -3, 4)^\intercal$. The $\|\boldsymbol{x}\|_1 = 9$, $\|\boldsymbol{x}\|_2 = \sqrt{29}$, and $\|\boldsymbol{x}\|_\infty = 4$.

Note that the 2-norm, also called **Euclidean norm**, can be defined as

$$\|\boldsymbol{x}\|_2^2 = \boldsymbol{x}^\intercal \boldsymbol{x} = \langle \boldsymbol{x}, \boldsymbol{x}\rangle,$$

that is, it is the norm induced by the Euclidean inner product. From this it follows that the 2-norm does not change under orthogonal transformations: if $\boldsymbol{Q} \in O(n)$ and $\boldsymbol{x} \in \mathbb{R}^n$, then $\|\boldsymbol{Q}\boldsymbol{x}\|_2 = \|\boldsymbol{x}\|_2$. Orthogonal transformations in $\mathbb{R}^2$ and $\mathbb{R}^3$ correspond to rotations and reflections, so it is intuitively clear that these don't change distances.

The **unit sphere** with respect to a norm is the set $\{\boldsymbol{x} \in \mathbb{R}^n \mid \|\boldsymbol{x}\| = 1\}$, and the (closed) **unit ball** is the set $\{\boldsymbol{x} \in \mathbb{R}^n \mid \|\boldsymbol{x}\| \leq 1\}$. The unit spheres with respect to the 2-norm, the 1-norm and the ∞-norm in $\mathbb{R}^2$ are shown in the following diagram.



The unit sphere with respect to the 2-norm in $\mathbb{R}^n$ is usually denoted by S^{n-1}.

The 1-, 2- and ∞-norms are equivalent, in the sense that they can be bounded in terms of each other. In particular,

$$\|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{x}\|_2 \leq \sqrt{n}\|\boldsymbol{x}\|_\infty, \quad \|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{x}\|_1 \leq n\|\boldsymbol{x}\|_\infty. \tag{A.3}$$

The inner product and the 2-norm are related the **Cauchy-Schwarz** inequality,

$$|\langle \boldsymbol{x}, \boldsymbol{y}\rangle| \leq \|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2,$$

with equality if and only if $\boldsymbol{x}$ and $\boldsymbol{y}$ are linearly dependent. As a consequence of the Cauchy-Schwarz inequality we get

$$-1 \leq \frac{\langle \boldsymbol{x}, \boldsymbol{y}\rangle}{\|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2} \leq 1.$$

The **angle** between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is the number $\theta \in [0, 2\pi)$ such that

$$\cos(\theta) = \frac{\langle \boldsymbol{x}, \boldsymbol{y}\rangle}{\|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2}.$$

If $\boldsymbol{x}$ and $\boldsymbol{y}$ are orthogonal, then $\cos(\theta) = 0$ and $\theta \in \{\pi/2, 3\pi/2\}$.

Norms are an important device to measure the size of vectors. In order to measure the amount by which a linear transformation (matrix) distorts vectors, we need the concept of **matrix norms**. A matrix norm is a function on the set of matrices that is a norm when considering a matrix as a vector, and in addition satisfies the condition

$$\|\boldsymbol{A}\boldsymbol{B}\| \leq \|\boldsymbol{A}\|\|\boldsymbol{B}\|.$$

The most important examples are given by the **operator norms**. Given a vector norm $\|\boldsymbol{x}\|$, the associated operator norm is defined as

$$\|\boldsymbol{A}\| = \max_{\boldsymbol{x}\neq\boldsymbol{0}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|} = \max_{\boldsymbol{x}\colon \|\boldsymbol{x}\|\leq 1} \|\boldsymbol{A}\boldsymbol{x}\|.$$

The matrix norms $\|\boldsymbol{A}\|_1, \|\boldsymbol{A}\|_2, \|\boldsymbol{A}\|_\infty$ are the operator norms that arise when using the 1-, 2- and ∞-norms. They can conveniently characterized as follows

- $\|\boldsymbol{A}\|_1 = \max_j \sum_{i=1}^n |a_{ij}|$;
- $\|\boldsymbol{A}\|_\infty = \max_i \sum_{j=1}^n |a_{ij}|$;
- $\|\boldsymbol{A}\|_2 = \sqrt{\lambda_{\max}(\boldsymbol{A}^\intercal\boldsymbol{A})}$.

Here, $\lambda_{\max}$ denotes the largest eigenvalue of the symmetric matrix $\boldsymbol{A}^\intercal\boldsymbol{A}$. If $\boldsymbol{A}$ is symmetric, then $\boldsymbol{A}^\intercal\boldsymbol{A} = \boldsymbol{A}^2$, and the eigenvalues of $\boldsymbol{A}^2$ are the squares of the eigenvalues of $\boldsymbol{A}$. It follows that for symmetric $\boldsymbol{A}$, $\|\boldsymbol{A}\|_2 = \lambda_{\max}(\boldsymbol{A})$.

Example A.9. Let

$$\boldsymbol{A} = \begin{pmatrix} 1 & -2 \\ -2 & 1 \end{pmatrix}.$$

Then $\|\boldsymbol{A}\|_1 = \|\boldsymbol{A}\|_\infty = 3$ and $\|\boldsymbol{A}\|_2 = 3$.

A special case is the **dual norm** of a vector norm: to a given norm is defined as

$$\|\boldsymbol{x}\|^* = \max_{\boldsymbol{y}\colon \|\boldsymbol{y}\|\leq 1} \langle \boldsymbol{x}, \boldsymbol{y}\rangle.$$

This is the operator norm of $\boldsymbol{x}^\intercal$, considered as a $1 \times n$ matrix. The dual of the dual norm is the norm itself. The dual norm of the 2-norm is again the 2-norm, while the 1-norm and the ∞-norm are dual to each other.

In addition to the operator norms, an important matrix norm is the **Frobenius norm** of a matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$,

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i,j=1}^n a_{ij}^2}.$$

This is just the 2-norm of $\boldsymbol{A}$ interpreted as a vector in $\mathbb{R}^{n^2}$. The 2-norm and the Frobenius norm have the important property of being **orthogonal invariant**, which means that for any $\boldsymbol{Q} \in O(n)$,

$$\|\boldsymbol{Q}\boldsymbol{A}\|_2 = \|\boldsymbol{A}\boldsymbol{Q}\|_2 = \|\boldsymbol{A}\|_2, \quad \|\boldsymbol{Q}\boldsymbol{A}\|_F = \|\boldsymbol{A}\boldsymbol{Q}\|_F = \|\boldsymbol{A}\|_F.$$

Orthogonal invariance allows to simplify a matrix without changing the norm.

Positive semidefinite matrices

If $\boldsymbol{A} \in \mathcal{S}^n$ is a symmetric matrix and $\boldsymbol{u} \in \mathbb{R}^n$ an eigenvector with $\|u\|_2 = 1$ and corresponding eigenvalue λ, then $\boldsymbol{u}^\intercal \boldsymbol{A}\boldsymbol{u} = \lambda \boldsymbol{u}^\intercal \boldsymbol{u} = \lambda$. In particular, the largest and smallest values of an eigenvalue are given by

$$\lambda_1 = \max_{\boldsymbol{u}\colon \|\boldsymbol{u}\|_2 = 1} \boldsymbol{u}^\intercal \boldsymbol{A}\boldsymbol{u}, \quad \lambda_n = \min_{\boldsymbol{u}\colon \|\boldsymbol{u}\|_2 = 1} \boldsymbol{u}^\intercal \boldsymbol{A}\boldsymbol{u}.$$

A symmetric matrix $\boldsymbol{A}$ is called **positive semidefinite**, written $\boldsymbol{A} \succeq \boldsymbol{0}$, if for all non-zero $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{x} \geq 0$, and **positive definite**, written $\boldsymbol{A} \succ \boldsymbol{0}$, if $\boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{x} > 0$ for all $\boldsymbol{x} \neq \boldsymbol{0}$. Equivalently, a symmetric matrix is positive semidefinite if all its eigenvalues are non-negative, and positive definite if they are all positive. The set of positive semidefinite symmetric matrices in $\mathbb{R}^{n\times n}$ is denoted by $\mathcal{S}^n_+$, while the set of positive definite matrices is $\mathcal{S}^n_{++}$.

An **inner product** (or scalar product) on $\mathbb{R}^{n\times n}$ is a function

$$\langle \cdot, \cdot \rangle \colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}, \quad (\boldsymbol{x}, \boldsymbol{y}) \mapsto \langle \boldsymbol{x}, \boldsymbol{y} \rangle$$

that is bilinear (linear in each of the two arguments), symmetric ($\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \langle \boldsymbol{y}, \boldsymbol{x} \rangle$), and satisfies $\langle \boldsymbol{x}, \boldsymbol{x} \rangle \geq 0$, with $\langle \boldsymbol{x}, \boldsymbol{x} \rangle = 0$ if and only if $\boldsymbol{x} = \boldsymbol{0}$. The standard inner product $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x}^\intercal \boldsymbol{y}$ is an example, and the notation $\langle \cdot, \cdot \rangle$ usually refers to this product. More generally, every matrix $\boldsymbol{A} \in \mathcal{S}^n_{++}$ defines an inner product by

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\boldsymbol{A}} := \langle \boldsymbol{x}, \boldsymbol{A}\boldsymbol{y} \rangle = \boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{y}.$$

The associated norm is $\|\boldsymbol{x}\|_{\boldsymbol{A}} = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle_{\boldsymbol{A}}}$. The unit sphere with respect to this norm,

$$E = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{x} = 1\},$$

is an **ellipsoid**, where the 2-norms of the largest and smallest axes are the largest and smallest eigenvalues of $\boldsymbol{A}^{-1}$.

Matrix decompositions

Matrices can be represented as products of simpler matrices. Important examples are:

1. QR **decomposition**. A matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ can be written as

   $$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R},$$

   where $\boldsymbol{Q} \in \mathbb{R}^{m\times m}$ is an orthogonal, and $\boldsymbol{R} \in \mathbb{R}^{m\times n}$ an upper triangular matrix. Gram-Schmidt orthogonalisation produces such a decomposition.

2. LU **decomposition**. A square matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ can be written as

   $$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$$

   where $\boldsymbol{L} \in \mathbb{R}^{n\times n}$ is a lower triangular, and $\boldsymbol{U} \in \mathbb{R}^{n\times n}$ an upper triangular matrix. Gaussian eliminations produces such a decomposition.

3. **Symmetric eigenvalue decomposition**. A symmetric matrix $\boldsymbol{A} \in \mathcal{S}^n$ can be written as

   $$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\intercal,$$

   where $\boldsymbol{Q}$ is an orthogonal matrix, with the eigenvectors as rows, and $\boldsymbol{\Lambda}$ a *diagonal* matrix with the eigenvalues on the diagonal.

4. **Cholesky decomposition**. A positive definite symmetric matrix $\boldsymbol{A} \in \mathcal{S}^n_{++}$ can be factored as

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^\intercal,$$

with $\boldsymbol{L} \in \mathbb{R}^{n\times n}$ lower-triangular and with strictly positive diagonal entries.

One of the most powerful matrix decompositions is the **singular value decomposition** (SVD). It states that any $m \times n$ matrix $\boldsymbol{A}$ can be written as

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal,$$

where $\boldsymbol{U}$ is a $m \times m$ orthogonal matrix, $\boldsymbol{V}$ an $n \times n$ orthogonal matrix, and $\boldsymbol{\Sigma}$ is a diagonal matrix with the **singular values** $\sigma_1 \geq \cdots \geq \sigma_{\min\{m,n\}}$ on the diagonal. The singular values are the square roots of the eigenvalues of $\boldsymbol{A}^\intercal\boldsymbol{A}$. The singular values are related to the matrix 2-norm and Frobenius norm of $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ as follows:

$$\|\boldsymbol{A}\|_2 = \sigma_1(\boldsymbol{A}), \quad \|\boldsymbol{A}\|_F = \sqrt{\sum_{i=1}^{n} \sigma_i^2(\boldsymbol{A})}.$$

If $\boldsymbol{A}$ is symmetric, the singular values are the absolute values of the eigenvalues of $\boldsymbol{A}$.

Matrix decompositions can help reduce a problem into one involving simpler (orthogonal, triangular) matrices. For example, to solve $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, one can first compute $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$, solve the simpler system of equations $\boldsymbol{Q}\boldsymbol{y} = \boldsymbol{b}$ by computing $\boldsymbol{y} = \boldsymbol{Q}^\intercal\boldsymbol{b}$, and then solve the triangular system $\boldsymbol{R}\boldsymbol{x} = \boldsymbol{y}$ by back-substitution.

Example A.10. Consider the matrix $\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}$. The QR decomposition is

$$\boldsymbol{Q} = \frac{1}{\sqrt{5}}\begin{pmatrix} -2 & 1 \\ 1 & 2 \end{pmatrix}, \quad \boldsymbol{R} = \frac{1}{\sqrt{5}}\begin{pmatrix} -5 & 4 \\ 0 & 3 \end{pmatrix},$$

the LU decomposition is

$$\boldsymbol{L} = \begin{pmatrix} 1 & 0 \\ -1/2 & 1 \end{pmatrix}, \quad \boldsymbol{U} = \begin{pmatrix} 2 & -1 \\ 0 & 3/2 \end{pmatrix},$$

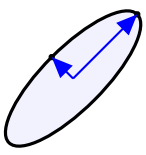
the symmetric eigenvalue decomposition and the SVD are given by

$$\boldsymbol{\Lambda} = \boldsymbol{\Sigma} = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}, \quad \boldsymbol{Q} = \boldsymbol{U} = \boldsymbol{V} = \begin{pmatrix} \cos(\pi/4) & \sin(\pi/4) \\ -\sin(\pi/4) & \cos(\pi/4) \end{pmatrix},$$

and the Cholesky decomposition is given by

$$\boldsymbol{L} = \frac{1}{\sqrt{2}}\begin{pmatrix} 2 & 0 \\ -1 & \sqrt{3} \end{pmatrix}.$$

The eigenvalue decomposition shows how to visualize the ellipse $\{\boldsymbol{A}\boldsymbol{x} \mid \|\boldsymbol{x}\|_2 = 1\}$:



Applying the transformation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\intercal\boldsymbol{x}$ corresponds to rotating the vector $\boldsymbol{x}$ clockwise by an angle of $\pi/4$, then stretching by a factor of 3 in the x-direction, and then rotating back.

3 Calculus

We write $C([a,b]) = C^0([a,b])$ for the set of continuous functions on an interval $[a,b]$, and for $k \geq 1$ we write $C^k([a,b])$ for the set of functions continuous on $[a,b]$, and whose first k derivatives $f', \ldots, f^{(k)}$ exist and are continuous on (a,b). In the above definition we allow $a = -\infty$ or $b = \infty$. If $a, b \in \mathbb{R}$, then any function $f \in C([a,b])$ is bounded. The **infimum** (largest lower bound) and **supremum** (smallest upper bound) of a function f on an interval $[a,b]$ are defined as

$$\inf_{x\in[a,b]} f(x) = \max\{y \in \mathbb{R} \mid \forall x \in [a,b], f(x) \geq y\},$$
$$\sup_{x\in[a,b]} f(x) = \min\{y \in \mathbb{R} \mid \forall x \in [a,b], f(x) \leq y\}.$$

Again, we allow the "values" $-\infty$ and ∞. If the infimum is attained (i.e., there exists x^* such that $f(x^*) = \inf_{x\in[a,b]} f(x)$), then we write $\min_{x\in[a,b]} f(x)$, and similarly $\max$ if the supremum is attained. Any $f \in C([a,b])$ for $a, b \in \mathbb{R}$ attains its infimum and supremum on $[a,b]$.

Three important results are the **Intermediate Value Theorem**, the **Mean Value Theorem**, and the **Taylor expansion**.

Theorem A.11 (Intermediate Value Theorem). *If $f \in C([a,b])$ and if y satisfies*

$$\inf_{x\in[a,b]} f(x) \leq y \leq \sup_{x\in[a,b]} f(x),$$

then there exists $\xi \in [a,b]$ such that $f(\xi) = y$. In particular, the infimum and supremum are attained.

Theorem A.12 (Mean Value Theorem). *Let $f \in C^1([a,b])$ and let $x, x_0 \in (a,b)$ with $x \neq x_0$. Then there exists a number $\xi \in (x_0, x)$ (or (x, x_0) if $x < x_0$) such that*

$$f(x) = f(x_0) + f'(\xi)(x - x_0).$$

This can also be written as

$$f'(\xi) = \frac{f(x) - f(x_0)}{x - x_0}.$$

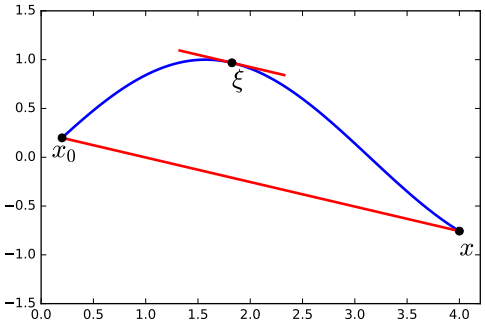


Figure A.1: MVT: there exists a point at which the slope (derivative) is the same as that of the secant connecting $(x_0, f(x_0))$ and $(x, f(x))$.

The Mean Value Theorem is a special case of the Taylor expansion.

Theorem A.13 (Taylor expansion). *Let $f \in C^{(n)}([a,b])$ and let $x, x_0 \in (a,b)$ with $x \neq x_0$. Then there exists $\xi \in (x, x_0)$ (or (x, x_0) if $x < x_0$) such that*

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1}$$

The first $(n + 1)$ terms of the above sum can be seen as an approximation to the function f that becomes more accurate as n increases. The last term is known as the *truncation error* in numerical approximation.

As an example, consider the Taylor expansion of the sine function at $x_0 = 0$,

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

The Taylor approximation to different orders is illustrated in the following figure.

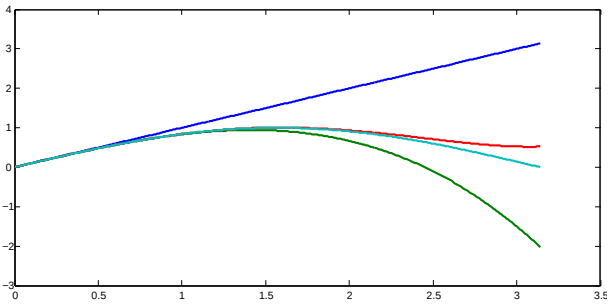


Figure A.2: Taylor expansion of $\sin(x)$.

A special case is of the Mean Value Theorem is **Rolle's Theorem**.

Theorem A.14 (Rolle's Theorem)**.** *Let $f \in C^1([a,b])$ with $f(a) = f(b)$. Then there exists a number $\xi \in (a,b)$ such that $f'(\xi) = 0$.*

The intuition is that if you walk across mountains and arrive at a point with the same elevation as where you started, then there must be places on the way where the slope is 0, that is, a local maximum or minimum. Of importance is also the following variant of the the Mean Value Theorem.

Theorem A.15 (Integral Mean Value Theorem). *Let $f, g \in C([a,b])$ and assume that $f(x)$ does not change sign on $[a,b]$. Then there exists a $\xi \in (a,b)$ such that*

$$\int_a^b f(x)g(x)\,dx = g(\xi)\int_a^b f(x)\,dx.$$

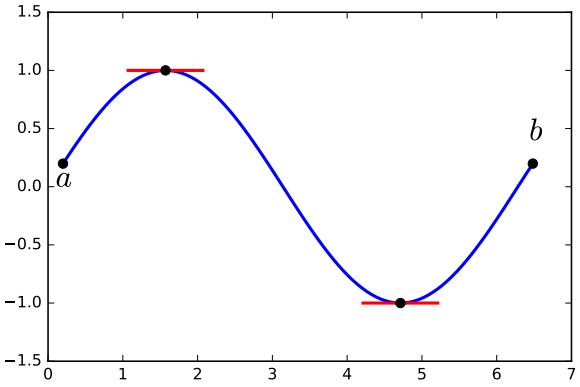


Figure A.3: Rolle's Theorem: there exist points with "flat" slope (derivative zero).

Topology

The **open ball** of radius ε around $\boldsymbol{p} \in \mathbb{R}^n$ is defined as

$$B^n(\boldsymbol{p}, \varepsilon) = \{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{p}\|_2 < \varepsilon\}.$$

We write $B^n := B^n(\mathbf{0}, 1)$ for the (open) **unit ball**. A subset $U \subseteq \mathbb{R}^n$ is called **open** if for every $p \in U$ there exists an $\varepsilon > 0$ such that $B(\boldsymbol{p}, \varepsilon) \subset U$. A set C is **closed** if $\overline{C} = \mathbb{R}^n \backslash C$ is open. The **closure** $\operatorname{cl} S$ of a set $S \subseteq \mathbb{R}^n$ is the intersection of all closed sets containing S, while the **interior** $\operatorname{int} S$ is the union of all open sets contained in S. The **boundary** of S is defined as $\operatorname{bd} S = \operatorname{cl} S \backslash \operatorname{int} S$. For example, the boundary of the open unit ball is the **unit sphere**

$$\operatorname{bd} B^n = S^{n-1} = \{\boldsymbol{x} \in \mathbb{R}^n \mid \|\boldsymbol{x}\|_2 = 1\}.$$

The superscript $n-1$ refers to the fact that this set is a manifold of dimension $n-1$. A **neighbourhood** N of a point $\boldsymbol{x} \in \mathbb{R}^n$ is a set such that there exists an open set U with $\boldsymbol{x} \in U \subseteq N$. An **open neighbourhood** is a neighbourhood that is open. Note that if $U_1 \subseteq \mathbb{R}^n$ and $U_2 \subseteq \mathbb{R}^m$ are open sets, then the product $U_1 \times U_2 \subseteq \mathbb{R}^{n+m}$ is also open.

Any subset $S \subseteq \mathbb{R}^n$ inherits a topological structure from $\mathbb{R}^n$, where the open sets in S are the sets of the form $U \cap S$, with $U \subseteq \mathbb{R}^n$ open. If a set S is contained in a lower-dimensional linear subspace $V \subset \mathbb{R}^n$, say, with $\dim V = k < n$, then S is always closed. However, it can be open *relative to its linear span*,

$$\operatorname{span}(S) = \left\{ \sum_{i=1}^{k} \lambda_i \boldsymbol{x}_i \mid \lambda_1, \ldots, \lambda_k \in \mathbb{R}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_k \in S \right\}.$$

We call a set S **relatively open** or **relatively closed** if it is open or closed in the induced topology on $\operatorname{span}(S)$. Another way of defining this notion goes as follows. Let $\{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k\}$ be an orthonormal basis of $\operatorname{span}(S)$, with $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k)$, and consider the map

$$\varphi_{\boldsymbol{B}} \colon \mathbb{R}^k \to \operatorname{span}(S), \quad \varphi_{\boldsymbol{B}}(\boldsymbol{x}) = \sum_{i=1}^{k} x_i \boldsymbol{b}_i.$$

Then a set S is relatively open or relatively closed if the preimage

$$\varphi_{\boldsymbol{B}}^{-1}(S) = \{\boldsymbol{x} \mid \varphi_{\boldsymbol{B}}(\boldsymbol{x}) \in S\}$$

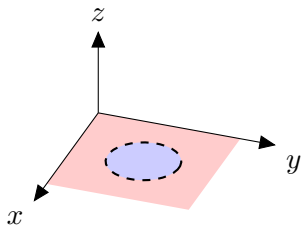


Figure A.4: The disk without boundary on the xy-plane is relatively open, and is the relative interior of the disk with boundary.

is open or closed in $\mathbb{R}^k$. Based on these notions, one defines the **relative closure** $\operatorname{relcl} S$ and **relative interior** $\operatorname{relint} S$ just as before.

A subset $S \subseteq \mathbb{R}^n$ is **bounded** if there exists number $M > 0$ such that $\|\boldsymbol{x}\|_2 < M$ for all $\boldsymbol{x} \in S$. Invoking the equivalence of norms (A.3) one sees that this definition does not depend on the norm chosen. A set $K \subseteq \mathbb{R}^n$ is called **compact** if it is closed and bounded. Equivalently, every cover of K with open sets contains a finite subcover. A function $f\colon \mathbb{R}^n \to \mathbb{R}^m$ is **continuous** if for every open set $U \subset \mathbb{R}^m$,

$$f^{-1}(U) := \{\boldsymbol{x} \in \mathbb{R}^n \mid f(\boldsymbol{x}) \in U\}$$

is an open subset of $\mathbb{R}^n$. A function defined on a subset $S \subseteq \mathbb{R}^n$ is said to be continuous if it is continuous on the induced topology. The set of continuous functions $f\colon S \to \mathbb{R}^m$ is denoted by $C(S, \mathbb{R}^m) = C^0(S, \mathbb{R}^m)$. If $f \in C(K, \mathbb{R})$, where K is compact, then f is bounded, and attains its infimum and supremum there: there exist $\boldsymbol{x}_*, \boldsymbol{x}^* \in K$ such that

$$\inf_{\boldsymbol{x}\in K} f(\boldsymbol{x}) = f(\boldsymbol{x}_*), \quad \sup_{\boldsymbol{x}\in K} f(\boldsymbol{x}) = f(\boldsymbol{x}^*).$$

A weaker notion is that of a **Lipschitz continuous** function. A function $f\colon S \to \mathbb{R}^m$ is called Lipschitz continuous with Lipschitz constant $L > 0$, if for all $\boldsymbol{x}, \boldsymbol{y} \in S$,

$$\|f(\boldsymbol{x}) - f(\boldsymbol{y})\|_2 \leq L\|\boldsymbol{x} - \boldsymbol{y}\|_2.$$

Notions about continuity of functions can be conveniently stated in terms of sequences. A **sequence** of points $\{\boldsymbol{x}_k\}_{k\in\mathbb{N}} \subset \mathbb{R}^n$ (for short, $\{\boldsymbol{x}_k\}$) **converges** to $\boldsymbol{x} \in \mathbb{R}^n$ as $k \to \infty$ with respect to a norm $\|\cdot\|$, written $\boldsymbol{x}_k \to \boldsymbol{x}$, if the sequence of numbers $\|\boldsymbol{x}_k - \boldsymbol{x}\|$ converges to 0,

$$\lim_{k\to\infty} \|\boldsymbol{x}_k - \boldsymbol{x}\| = 0.$$

Formally, this means that for every $\varepsilon > 0$ there exists an index k_0, such that for all $k > k_0$, $\|\boldsymbol{x}_k - \boldsymbol{x}\| < \varepsilon$. From the equivalence of norms (A.3) it follows that if a sequence converges with respect to one norm, it also converges with respect to the other norms. If we fix a norm and the associated topology on $\mathbb{R}^n$, then a function $f\colon \mathbb{R}^n \to \mathbb{R}$ is then continuous at $\boldsymbol{x}$, if for every sequence $\{\boldsymbol{x}_k\}_{k\in\mathbb{N}}$ with $\lim_{k\to\infty} \boldsymbol{x}_k \to \boldsymbol{x}$ with respect to the given norm, $\lim f(\boldsymbol{x}_k) \to f(\boldsymbol{x})$. A stronger notion of continuity is uniform continuity. A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is called **uniformly continuous** if for every $\varepsilon > 0$ there exists $\delta > 0$ such that *for all* $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, $\|\boldsymbol{x} - \boldsymbol{y}\| < \delta$ implies $|f(\boldsymbol{x}) - f(\boldsymbol{y})| < \varepsilon$.

A **subsequence** of a sequence $\{\boldsymbol{x}_k\}$ is an infinite subset of S. A **limit point** for a sequence $S = \{\boldsymbol{x}_k\}$ is a point $\boldsymbol{x}$ that is the limit of an subsequence of S. Formally, for every $\varepsilon > 0$ there exists a k_0 such that $\|\boldsymbol{x}_k - \boldsymbol{x}\| < \varepsilon$ *for some* $k > k_0$. A sequence $\{\boldsymbol{x}_k\}$ is called a **Cauchy sequence** if for every $\varepsilon > 0$ there exists an index $k_0 > 0$, such that for all $k, \ell > k_0$, $\|\boldsymbol{x}_k - \boldsymbol{x}_\ell\|_2 < \varepsilon$. The vector space $\mathbb{R}^n$ with the 2-norm (or any other norm) is a **Banach space**, which means that every Cauchy sequence contains a convergent subsequence.

All the topological notions discussed earlier have an interpretation in terms of sequences and limits:

1. A set C is closed if and only if for every sequence $\{x_k\} \subset C$, all limit points are in C;
2. The closure of a set S is the set of all limit points of sequences in S;
3. A set K is compact, if and only if every sequence of points $\{x_k\}$ in K has a limit point in K.

Given a function $f\colon \Omega \to \mathbb{R}^m$, where $\Omega \subseteq \mathbb{R}^n$, and $\boldsymbol{x} \in \Omega$, then f is **continuous at** $\boldsymbol{x}^*$ if

$$\lim_{\boldsymbol{x}\to\boldsymbol{x}^*} f(\boldsymbol{x}) = f(\boldsymbol{x}^*).$$

Formally, for every $\varepsilon > 0$ there exists a $\delta > 0$ such that whenever $\|\boldsymbol{x} - \boldsymbol{x}^*\| < \delta$, $\|f(\boldsymbol{x}) - f(\boldsymbol{x}^*)\| < \varepsilon$. This means that for every sequence of points $\{\boldsymbol{x}_k\}$ with $\lim_{k\to\infty} \boldsymbol{x}_k = \boldsymbol{x}^*$, the sequence $f(\boldsymbol{x}_k) \to f(\boldsymbol{x}^*)$ as $k \to \infty$ with respect to some norm on $\mathbb{R}^m$.

Differentiable functions

A function $f\colon \mathbb{R}^n \to \mathbb{R}^m$ is called (Fréchet) **differentiable** at $\boldsymbol{x}_0 \in \mathbb{R}^n$ if there exists a linear map $\boldsymbol{J}f(\boldsymbol{x}_0)\colon \mathbb{R}^n \to \mathbb{R}^m$, such that

$$\lim_{\boldsymbol{h}\to\boldsymbol{0}} \frac{\|f(\boldsymbol{x}_0 + \boldsymbol{h}) - f(\boldsymbol{x}_0) - \boldsymbol{J}f(\boldsymbol{x}_0)\boldsymbol{h}\|_2}{\|\boldsymbol{h}\|_2} = 0.$$

A function is differentiable on an open subset $U \subseteq \mathbb{R}^n$ if it is differentiable at every $\boldsymbol{x} \in U$. If $f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))^\intercal$ is differentiable, then all the partial derivatives exist, and $\boldsymbol{J}f(\boldsymbol{x}_0)$ is represented by the **Jacobian matrix**

$$\boldsymbol{J}f(\boldsymbol{x}_0) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix},$$

where the partial derivatives are evaluated at $\boldsymbol{x}_0$. If all the partial derivatives exist and are continuous in a neighbourhood of $\boldsymbol{x}_0$ (called **continuously differentiable**), then f is differentiable at $\boldsymbol{x}_0$.

If $m = 1$, then $J(\boldsymbol{x}_0)$ is the transpose of the **gradient** $\nabla f(\boldsymbol{x}_0)$ of f at $\boldsymbol{x}_0$,

$$\nabla f(\boldsymbol{x}_0) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)^\top.$$

The gradient points in the direction in which f increases the most.

A convenient way to visualise a function $f\colon \mathbb{R}^2 \to \mathbb{R}$ is through **level sets** $\{\boldsymbol{x} \in \mathbb{R}^2 \mid f(\boldsymbol{x}) = c\}$. For each $c \in \mathbb{R}$, such a level set defines a curve in $\mathbb{R}^2$, the curve on which the function value does not change. The gradient is always orthogonal to the level set, pointing in the direction in which f increases the most (see Figure A.5).

If the gradient, considered as a map $\mathbb{R}^n \to \mathbb{R}^n$, is itself differentiable at $\boldsymbol{x}_0$, then the Jacobian matrix of the gradient is called the **Hessian matrix**,

$$\nabla^2 f(\boldsymbol{x}_0) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{pmatrix}$$

Since

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i},$$

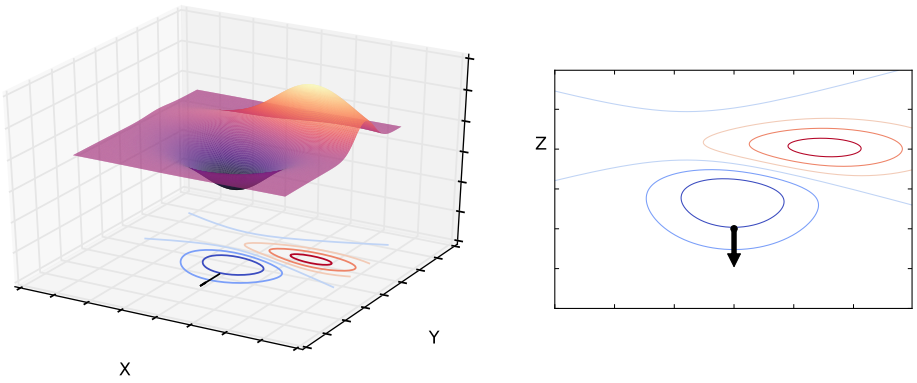


Figure A.5: A surface, level sets, and the gradient

the Hessian is a symmetric matrix.

The **directional derivative** $D_{\boldsymbol{x}_0} f(\boldsymbol{x}_0)$ of a function $f\colon \mathbb{R}^n \to \mathbb{R}^m$ in direction $\boldsymbol{v} \in \mathbb{R}^n$ at $\boldsymbol{x}_0$ is defined as

$$D_{\boldsymbol{v}} f(\boldsymbol{x}_0) = \lim_{h \to 0} \frac{f(\boldsymbol{x}_0 + h\boldsymbol{v}) - f(\boldsymbol{x}_0)}{h}.$$

In the special case where $\boldsymbol{v} = \boldsymbol{e}_i$, we obtain the partial derivative with respect to x_i,

$$\frac{\partial f}{\partial x_i}(\boldsymbol{x}_0) = D_{\boldsymbol{e}_i} f(\boldsymbol{x}_0).$$

If $f\colon \mathbb{R}^n \to \mathbb{R}$ is differentiable with continuous derivative near $\boldsymbol{x}_0$, then

$$D_{\boldsymbol{v}} f(\boldsymbol{x}_0) = \nabla f(\boldsymbol{x}_0)^\intercal \boldsymbol{v} = \langle \nabla f(\boldsymbol{x}_0), \boldsymbol{v} \rangle.$$

If $f\colon \mathbb{R}^n \to \mathbb{R}^m$ and $g\colon \mathbb{R}^m \to \mathbb{R}^p$ are differentiable in a neighbourhood of $\boldsymbol{x}_0 \in \mathbb{R}^n$ and $f(\boldsymbol{x}_0) \in \mathbb{R}^m$, respectively, then the composition $h = g \circ f\colon \mathbb{R}^n \to \mathbb{R}^p$ is continuously differentiable in a neighbourhood of $\boldsymbol{x}_0$, and the Jacobian matrix is defined by the **chain rule**:

$$Jh(\boldsymbol{x}_0) = Jg(f(\boldsymbol{x}_0))Jf(\boldsymbol{x}_0).$$

If $n = 1$, then $f\colon \mathbb{R} \to \mathbb{R}^n$ is called a **curve**, and we write

$$Jf = \frac{\mathrm{d}f}{\mathrm{d}t} = \dot{f} = (\dot{f}_1, \ldots, \dot{f}_n)^\intercal \in \mathbb{R}^n$$

for the derivative of the curve. If $\dot{f}(t_0) = \boldsymbol{v} \in \mathbb{R}^n$ and $g\colon \mathbb{R}^n \to \mathbb{R}$, then by the chain rule, the derivative of $g \circ f\colon \mathbb{R} \to \mathbb{R}$ is the directional derivative of g in the direction $\boldsymbol{v}$,

$$\frac{\mathrm{d}g \circ f(t_0)}{\mathrm{d}t} = \langle \nabla g(f(t_0)), \boldsymbol{v} \rangle.$$

Before going on to deal with higher derivative, we state the generalisation of the Mean Value Theorem to higher dimensions.

Theorem A.16 (Multivariate Mean Value Theorem)**.** *Let $f \in C^1(U)$ for an open set U with $\boldsymbol{x}_0, \boldsymbol{x} \in U$, $\boldsymbol{x} \neq \boldsymbol{x}_0$. Then there exists $t \in (0,1)$ such that*

$$f(\boldsymbol{x}) - f(\boldsymbol{x}_0) = \langle \nabla f(t\boldsymbol{x} + (1-t)\boldsymbol{x}_0), \boldsymbol{x} - \boldsymbol{x}_0 \rangle.$$

Note that $t\boldsymbol{x} + (1-t)\boldsymbol{x}_0$ parametrises the line segment connecting $\boldsymbol{x}$ and $\boldsymbol{x}_0$.

For a tuple of natural number $\alpha = (\alpha_1, \ldots, \alpha_n)$, set $|\alpha| = \sum_{i=1}^n \alpha_i$, and define the higher order partial derivative

$$D^\alpha f(\boldsymbol{x}) = \frac{\partial^{|\alpha|} f(\boldsymbol{x})}{\partial^{\alpha_1} x_1 \cdots \partial^{\alpha_n} x_n}.$$

For a set $S \subseteq \mathbb{R}^n$, denote by $C^k(S, \mathbb{R}^m)$ the set of functions f such that all partial derivatives $D^\alpha f$ with $|\alpha| \leq k$ exists and are continuous on $\mathrm{int} S$. If $m = 1$, we write $C^k(S) := C^k(S, \mathbb{R})$.

Define, for a vector $\boldsymbol{x}$ and multi-index α,

$$\boldsymbol{x}^\alpha := x_1^{\alpha_1} \cdots x_n^{\alpha_n}.$$

We then have the **Taylor expansion** around a point $\boldsymbol{x}_0$,

$$f(\boldsymbol{x}) = \sum_{|\alpha| \leq k} \frac{D^\alpha f(\boldsymbol{x}_0)}{\alpha!} (\boldsymbol{x} - \boldsymbol{x}_0)^\alpha + \sum_{|\alpha| = k} r_\alpha(\boldsymbol{x}) (\boldsymbol{x} - \boldsymbol{x}_0)^\alpha,$$

with $r_\alpha(\boldsymbol{x}) \to 0$ as $\boldsymbol{x} \to \boldsymbol{x}_0$.

If a differentiable function $f(\boldsymbol{x})$ has a local minimum or maximum at a point $\boldsymbol{x}$, then this point satisfies $\nabla f(\boldsymbol{x}) = 0$, that is, it is a critical point. The **Lagrange multiplier theorem** says something about local extrema under certain constraints.

Theorem A.17 (Lagrange multipliers). *Let $\boldsymbol{x}^*$ be maximum of $f(\boldsymbol{x})$ under the constraint $g(\boldsymbol{x}) = c$ (that is, a maximum among all points $\boldsymbol{x}$ such that $g(\boldsymbol{x}) = c$). Then there exist a $\lambda \in \mathbb{R}$ such that*

$$\nabla f(\boldsymbol{x}^*) = \lambda \nabla g(\boldsymbol{x}^*).$$

The **Lagrangian** of a function $f(\boldsymbol{x})$ with constraint $g(\boldsymbol{x}) = c$ is the function $\Lambda \colon \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ defined by

$$\Lambda(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) - \lambda(g(\boldsymbol{x}) - c).$$

The Lagrange multiplier theorem then says that if $\boldsymbol{x}^*$ is a maximum point of $f(\boldsymbol{x})$ under the constraint $g(\boldsymbol{x}) = c$, then there exists $\lambda \in \mathbb{R}$ such that the pair $(\boldsymbol{x}^*, \lambda)$ is a critical point of the Lagrangian $\Lambda(\boldsymbol{x}, \lambda)$.

The **Implicit Function Theorem** is one of the most important results in analysis, and underlies much of differential geometry and physics. Let $F \colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ by a function that is continuously differentiable in a neighbourhood of a point $(\boldsymbol{x}_0, \boldsymbol{y}_0)$, with $\boldsymbol{x}_0 \in \mathbb{R}^n$ and $\boldsymbol{y}_0 \in \mathbb{R}^m$. The Jacobian $J_{\boldsymbol{x}}(\boldsymbol{x}_0, \boldsymbol{y}_0)$ with respect to the first set of n coordinates consists of the first n columns of the Jacobian matrix,

$$J_{\boldsymbol{x}}(\boldsymbol{x}_0, \boldsymbol{y}_0) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\boldsymbol{x}_0, \boldsymbol{y}_0) & \cdots & \frac{\partial f_1}{\partial x_n}(\boldsymbol{x}_0, \boldsymbol{y}_0) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(\boldsymbol{x}_0, \boldsymbol{y}_0) & \cdots & \frac{\partial f_m}{\partial x_n}(\boldsymbol{x}_0, \boldsymbol{y}_0) \end{pmatrix}$$

The interpretation is that we consider f as a function in only the first set of coordinates, with the remaining ones (denoted by $\boldsymbol{y}$) considered as parameters.

Theorem A.18 (Implicit Function Theorem). *Let $f \colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ be k times continuously differentiable in an open neighbourhood of $(\boldsymbol{x}_0, \boldsymbol{y}_0) \in \mathbb{R}^n \times \mathbb{R}^m$, and assume that $f(\boldsymbol{x}_0, \boldsymbol{y}_0) = \boldsymbol{0}$. Assume further that the Jacobian $J_{\boldsymbol{x}} f(\boldsymbol{x}_0, \boldsymbol{y}_0) \in \mathbb{R}^{n \times n}$ in the first n coordinates is* non-singular *at $(\boldsymbol{x}_0, \boldsymbol{y}_0)$. Then there exists an open neighbourhood $\boldsymbol{y}_0 \in U_{\boldsymbol{y}} \subseteq \mathbb{R}^m$, and a function $h \in C^k(U_{\boldsymbol{y}}, \mathbb{R}^n)$ such that*

- $h(\boldsymbol{y}_0) = \boldsymbol{x}_0$,

- $f(h(\boldsymbol{y}), \boldsymbol{y}) = 0$ *for* $\boldsymbol{y} \in U_{\boldsymbol{y}}$.

Moreover, the Jacobian of h *is given by*

$$Jh(\boldsymbol{y}) = -Jf_{\boldsymbol{y}}(h(\boldsymbol{y}), \boldsymbol{y})(J_{\boldsymbol{x}} f(h(\boldsymbol{y}), \boldsymbol{y}))^{-1}$$

for all $\boldsymbol{y} \in U_{\boldsymbol{y}}$.

Example A.19. Let $f(x, y) = x^2 + y^2 - 1$ and $(x_0, y_0) = (1, 0)$. The Jacobian of f in the first coordinate is

$$\frac{\partial f}{\partial x}(1, 0) = 2 \neq 0,$$

which is non-singular. Choosing the neighbourhood $U_y = (-1, 1)$, the open interval between -1 and 1, we get the function $h \colon U_y \to \mathbb{R}$ as

$$h(y) = \sqrt{1 - y^2}.$$

This function is defined and differentiable on $(-1, 1)$, and satisfies

$$f(h(y), y) = h(y)^2 + y^2 - 1 = (1 - y^2) + y^2 - 1 = 0, \quad y \in U_y.$$

The derivative of h can be computed using the chain rule:

$$\frac{\mathrm{d}f(h(y), y)}{\mathrm{d}y} = \frac{\partial f}{\partial x}(h(y), y)\frac{\mathrm{d}h}{\mathrm{d}y}(y) + \frac{\partial f}{\partial y}(h(y), y)\frac{\mathrm{d}y}{\mathrm{d}y}$$

from which we get

$$\frac{\mathrm{d}h}{\mathrm{d}y}(y) = -\frac{\partial f}{\partial y}(h(y), y)\left(\frac{\partial f}{\partial x}(h(y), y)\right)^{-1} = -y(1 - y^2)^{-3/2}.$$

In this example, the implicit function theorem just gives the usual way of parametrising part of the circle.

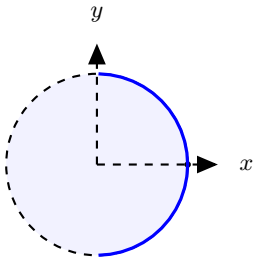


Figure A.6: The blue arc is parametrised by $h(y) = \sqrt{1 - y^2}$ for $y \in (-1, 1)$.

Differential Equations

A system of **ordinary differential equations** (**ODE**) relates a quantity to its rate of change. An **initial value problem** is the problem of finding a function $\boldsymbol{y} \colon \mathbb{R} \to \mathbb{R}^n$ that satisfies the conditions

$$\dot{\boldsymbol{y}} = f(\boldsymbol{y}, t), \quad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \tag{A.1}$$

where, as before, $\dot{\boldsymbol{y}}$ denotes the derivative with respect to t. The following existence and uniqueness theorem is of fundamental importance in mathematics.

Theorem A.20 (Existence and Uniqueness). *Consider the initial value problem* (A.1) *and assume that f is Lipschitz continuous in $\boldsymbol{y}$ and continuous in t. Then there exists $\epsilon > 0$ such that* (A.1) *has a unique solution $\boldsymbol{y}(t)$ in the interval $(t_0 - \epsilon, t_0 + \epsilon)$.*

Differential equations involving higher derivatives can be reduced to the form (A.1) by introducing new variables to represent the higher derivatives.

4 Probability

A **probability space** is a triple $(\Omega, \mathcal{F}, \mathbb{P})$, consisting of a set Ω, a σ-algebra $\mathcal{F}$ of **measurable** subsets of Ω, called **events**, and a **probability measure** $\mathbb{P}$. That $\mathcal{F}$ is a σ-algebra means that it contains $\emptyset$ and Ω and that it is closed under countable unions and complements. The probability measure $\mathbb{P}$ is a non-negative function $\mathbb{P}\colon \mathcal{F} \to [0, 1]$ such that $\mathbb{P}(\emptyset) = 0$, $\mathbb{P}(\Omega) = 1$, and

$$\mathbb{P}\left(\bigcup_i A_i\right) = \sum_i \mathbb{P}(A_i)$$

for a collection $\{A_i\} \subset \mathcal{F}$ with $A_i \cap A_j = \emptyset$ for $i \neq j$. We interpret $\mathbb{P}(A \cup B)$ as the probability of A *or* B happening, and $\mathbb{P}(A \cap B)$ as the probability of A *and* B happening. Note that $(A \cup B)^c = A^c \cap B^c$, where A^c is the complement of A in Ω.

If the A_i are not necessarily disjoint, then we have the important **union bound**

$$\mathbb{P}\left(\bigcup_i A_i\right) \leq \sum_i \mathbb{P}(A_i).$$

This bound is sometimes also referred to as the *zero-th moment method*.

Example A.21. Suppose we are rolling a die and A is the event that the result is an even number, while B is the event that the result is at least 4. Then

$$\mathbb{P}(A) = \frac{1}{2}, \quad \mathbb{P}(B) = \frac{1}{2}, \quad \mathbb{P}(A \cup B) = \frac{2}{3} < 1 = \mathbb{P}(A) + \mathbb{P}(B).$$

Clearly, the union bound gives a trivial bound here. In general, the union bound becomes less useful the more the sets involved overlap.

We say that an event A holds **almost surely** if $\mathbb{P}(A) = 1$ (note that this does not mean that the complement of A in Ω is empty).

Random variables

A measurable function between two measure spaces is a function for which the preimage of a measurable set is measurable. A **random variable** is a measurable map

$$X\colon \Omega \to \mathcal{X},$$

where $\mathcal{X}$ is typically $\mathbb{R}$, $\mathbb{R}^d$, $\mathbb{N}$, or a finite set $\{0, 1, \ldots, k\}$, all considered as measure spaces with the Borel σ-algebra (the smallest σ-algebra containing the open sets, where for discrete sets we consider the discrete topology). For a measurable set $A \subset \mathcal{X}$, we write

$$\mathbb{P}(X \in A) := \mathbb{P}(\{\omega \in \Omega\colon X(\omega) \in A\}).$$

We will usually use upper-case letters X, Y, Z for random variables, and lower-case letters x, y, z for the values that these can take.

Example A.22. A random variable specifies which events "we can see". For example, let $\Omega = \{1,2,3,4,5,6\}$ and define $X\colon \Omega \to \{0,1\}$ by setting $X(\omega) = \mathbf{1}\{\omega \in \{4,6\}\}$, where $\mathbf{1}$ denotes the indicator function. Then

$$\mathbb{P}(X=1) = \frac{1}{3}, \quad \mathbb{P}(X=0) = \frac{2}{3}.$$

If all the information we get about Ω is from X, then we can only determine whether the result of rolling a die gives an even number greater than 3 or not, but not the individual result.

The map $A \mapsto \mathbb{P}(X \in A)$ for subsets of $\mathcal{X}$ is called the **distribution** of the random variable. The distribution completely describes the random variable, and there will often be no need to refer to the domain Ω. If $F\colon \mathcal{X} \to \mathcal{Y}$ is another measurable map, then $F(X)$ is again a random variable. In particular, if $\mathcal{X}$ is a subset of $\mathbb{R}^d$, we can add and multiply random variables to obtain new random variables, and if X and Y are two distinct random variables, then (X,Y) is a random variable in the product space. In the latter case we also write $\mathbb{P}(X \in A, Y \in B)$ instead of $\mathbb{P}((X,Y) \in A \times B)$. Note that this also has an interpretation in terms of intersections of events: it is the probability that *both* $X \in A$ and $Y \in B$.

A **discrete** random variable takes countable many values, for example in a finite set $\{1,\dots,k\}$ or in $\mathbb{N}$. In such a case it makes sense to talk about the probability of individual outcomes, such as $\mathbb{P}(X=k)$ for some $k \in \mathcal{X}$. An **absolutely continuous** random variable takes values in $\mathbb{R}$ or $\mathbb{R}^d$ for $d > 1$, and is defined as having a **density** $\rho(x) = \rho_X(x)$, such that

$$\mathbb{P}(X \in A) = \int_A \rho(x)\,\mathrm{d}x.$$

In the case where $\mathcal{X} = \mathbb{R}$, we consider the **cumulative distribution function** (cdf) $\mathbb{P}(X \le t)$ for $t \in \mathbb{R}$. The complement, $\mathbb{P}(X > t)$ (or $\mathbb{P}(X \ge t)$), is referred to as the **tail**. Many applications are concerned with finding good bounds on the tail of a probability, as the the tail often models the probability of rare events. If X is absolutely continuous, then the probability of X taking a particular single value vanishes, $\mathbb{P}(X=a) = 0$. For a random variable $Z = (X,Y)$ taking values in $\mathcal{X} \times \mathcal{Y}$, we can consider the **joint density** $\rho_Z(x,y)$, but also the individual densities of X and Y, for which we have

$$\rho_X(x) = \int_{\mathcal{Y}} \rho_Z(x,y)\,\mathrm{d}y.$$

The ensuing distributions for X and Y are called the **marginal distributions**.

Example A.23. Three of the most common distributions are:

- **Bernoulli distribution** $\mathrm{Ber}(p)$, taking values in $\{0,1\}$ and defined by

  $$\mathbb{P}(X=1) = p, \quad \mathbb{P}(X=0) = 1-p$$

  for some $p \in [0,1]$. We can replace the range $\mathcal{X} = \{0,1\}$ by any other two-element set, for example $\{-1,1\}$, but then the relation to other distributions may not hold any more.

- **Binomial distribution** $\mathrm{Bin}(n,p)$, taking values in $\{0,\dots,n\}$ and defined by

  $$\mathbb{P}(X=k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{A.1}$$

  for $k \in \{0,1,\dots,n\}$ and some $p \in [0,1]$. We can also write a binomial random variable as a sum of Bernoulli random variables, $X = X_1 + \cdots + X_n$, since $X = k$ if and only if k of the summands have the value 1.

- **Normal distribution** $\mathcal{N}(\mu, \sigma^2)$, also referred to as Gaussian, with mean μ and variance σ^2, defined on $\mathbb{R}$ and with density
$$\gamma(x) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$
This is the most important distribution in probability and statistics, as most other distributions can be approximated by it.

Moments

The **expectation** (or mean, or expected value) of a discrete random variable is defined as
$$\mathbb{E}[X] = \sum_{k \in \mathcal{X}} k \cdot \mathbb{P}(X = k).$$
For an absolutely continuous random variable with density $\rho(x)$, it is defined as
$$\mathbb{E}[X] = \int_{\mathcal{X}} \rho(x) \, \mathrm{d}x.$$
Note that the expectation does not always need to exist since the sum or integral need not converge. When we require it to exist, we often write this as $\mathbb{E}[X] < \infty$.

Example A.24. The expectation of a Bernoulli random variable with parameter p is $\mathbb{E}[X] = p$. The expectation of a Binomial random variable with parameters n and p is $\mathbb{E}[X] = np$. For example, if one were to flip a biased coin that lands on heads with probability p, then this would correspond to the number of heads one would "expect" after n coin flips. The expectation of the normal distribution $\mathcal{N}(\mu, \sigma^2)$ is μ. This is the location on which the "bell curve" is centred.

One of the most useful properties is **linearity of expectation**. If $X_1, \ldots, X_n$ are random variables taking values in a subset of $\mathbb{R}^d$ and $a_1, \ldots, a_n \in \mathbb{R}$, then
$$\mathbb{E}[a_1 X_1 + \cdots + a_n X_n] = a_1 \, \mathbb{E}[X_1] + \cdots + a_n \, \mathbb{E}[X_n].$$

Example A.25. The expected value of a Bernoulli random variable with parameter p is
$$\mathbb{E}[X] = 1 \cdot \mathbb{P}(X = 1) + 0 \cdot \mathbb{P}(X = 0) = p.$$
The linearity of expectation then immediately gives the expectation of the Binomial distribution with parameters n and p. Since such a random variable can be written as $X = X_1 + \cdots + X_n$, with X_i Bernoulli, we get
$$\mathbb{E}[X] = \mathbb{E}[X_1 + \cdots + X_n] = \mathbb{E}[X_1] + \cdots + \mathbb{E}[X_n] = np.$$
This would be (slightly) harder to deduce from the direct definition (A.1), when one would have to use the binomial theorem.

If $F \colon \mathcal{X} \to \mathcal{Y}$ is a measurable function, then the expectation of the random variable $F(X)$ can be expressed as
$$\mathbb{E}[F(X)] = \int_{\mathcal{X}} F(x)\rho(x) \, \mathrm{d}x \tag{A.2}$$
in the case of an absolutely continuous random variable, and similarly in the discrete case. [2]

[2] We will not always list the formulas for both the discrete and continuous, when the form of one of these cases can be easily guessed from the form of the other case. In any case, the sum in the discrete setting is also just an integral with respect to the discrete measure.

An important special case is the indicator function

$$F(X) = \mathbf{1}\{X \in A\} = \begin{cases} 1 & X \in A \\ 0 & X \notin A. \end{cases}$$

Then

$$\mathbb{E}[\mathbf{1}\{X \in A\}] = \mathbb{P}(X \in A), \tag{A.3}$$

as can be seen by applying (A.2) to the indicator function. The identity (A.3) is useful, as it allows to properties of the expectation, such as linearity, in the study of probabilities of events. The expectation also has the following monotonicity property: if $0 \leq X \leq Y$, where X, Y are real-valued random variables, then $\mathbb{E}[X] \leq \mathbb{E}[Y]$.

Another important identity for random variables is the following. Assume X is absolutely continuous, takes values in $\mathbb{R}$, and $X \geq 0$. Then

$$\mathbb{E}[X] = \int_0^\infty \mathbb{P}(X > t)\, \mathrm{d}t.$$

Using this identity, one can deduce bounds on the expectation from bounds on the tail of a probability distribution.

The **variance** of a random variable is the expectation of the square deviation from the mean:

$$\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2].$$

The variance measures the "spread" of a distribution: random variables with a small variance are more likely to stay close to their expectation.

Example A.26. The variance of the normal distribution is σ^2. The variance of the Bernoulli distribution is $p(1-p)$ (verify this!), while the variance of the Binomial distribution is $np(1-p)$.

The variance scales as $\mathrm{Var}(aX + b) = a^2\mathrm{Var}(X)$. In particular, it is translation invariant. The variance is in general not additive (but it is, if the random variables are independent).

Besides the expectation and the variance, the **higher moments** $\mathbb{E}[X^k]$ of a random variable are of interest. These are encoded in the **exponential moment generating function** (mgf)

$$\mathbb{E}\left[\mathrm{e}^{\lambda X}\right] = \sum_{k=0}^{\infty} \frac{\mathbb{E}[X^k]}{k!}.$$

The moments need not all be finite. Related to the moment generating function is the **characteristic function**

$$\varphi_X(t) = \mathbb{E}\left[\mathrm{e}^{itX}\right].$$

The characteristic function is multiplicative:

$$\varphi_{X+Y}(t) = \varphi_X(t)\varphi_Y(t). \tag{A.4}$$

It allows one to express the density $\rho(x)$ of an absolutely continuous random variable X as

$$\rho(x) = \frac{1}{2\pi} \int_{t=-\infty}^{\infty} \mathrm{e}^{-itx} \varphi_X(t)\, \mathrm{d}t.$$

The characteristic function plays an important role in a common proof of the central limit theorem.

Example A.27. Let X be a Bernoulli random variable with parameter p. Then the characteristic function is $\varphi_X(t) = 1 - p + p\mathrm{e}^{it}$. For a binomial random variable, $X \sim \mathrm{Bin}(n, p)$, the characteristic function is $(1 - p + p\mathrm{e}^{it})^n$. This follows from the multiplicative property (A.4).

Convergence

There are various notions of convergence associated with a sequence of random variables $\{X_n\}_{n\geq 1}$ with distributions $\{\mathbb{P}_n\}$. For simplicity we assume real-valued distributions here.

1. (**Convergence in distribution**) A sequence $\{X_n\}_{n\geq 1}$ converges to X in distribution, $X_n \xrightarrow{d} X$, if for every $a \in \mathbb{R}$ at which $\mathbb{P}(X \leq a)$ is continuous,
$$\lim_{n\to\infty} \mathbb{P}_n(X_n \leq a) = P(X \leq a).$$
2. ()

The **portmanteau lemma** provides a series of equivalent characterizations of convergence in distribution. Perhaps the most important one is that a sequence of random variables $\{X_n\}$ converges in distribution to X if and only if for every

Independence

A set of random variables $\{X_i\}$ taking values in the same range $\mathcal{X}$ is called **independent** if for any subset $\{X_{i_1}, \ldots, X_{i_k}\}$ and any subsets $A_j \subset \mathcal{X}$, $1 \leq j \leq k$, we have
$$\mathbb{P}(X_{i_1} \in A_1, \ldots, X_{i_k} \in A_k) = \mathbb{P}(X_{i_1} \in A_1) \cdots \mathbb{P}(X_{i_k} \in A_k).$$
In words, the probability of any of the events happening simultaneously is the product of the probabilities of the individual events. A set of random variables $\{X_i\}$ is said to be **pairwise independent** if every subset of two variables is independent. Note that pairwise independence does not imply independence.

Example A.28. Assume you toss a fair coin two times. Let X be the indicator variable for heads on the first toss, Y the indicator variable for heads on the second toss, and Z the random variable that is 1 if $X = Y$ and 0 if $X \neq Y$. Taken individually, each of these random variables is a Bernoulli random variable with $p = 1/2$. They are also pairwise independent, as is easily verified, but not independent, since
$$\mathbb{P}(X = 1, Y = 1, Z = 1) = \frac{1}{4} \neq \frac{1}{8} = \mathbb{P}(X = 1)\mathbb{P}(Y = 1)\mathbb{P}(Z = 1).$$
Intuitively, the information that $X = 1$ and $Y = 1$ already implies $Z = 1$, so adding this constraint does not alter the probability on the left-hand side.

We say that a set of random variables $\{X_i\}$ is **i.i.d.** if they are **independent and identically distributed**. This means that each X_i can be seen as a *copy* of X_1 that is independent of it, and in particular all the X_i have the same expectation and variance.

One of the most important results in probability (and, arguably, in nature) is the (strong) **law of large numbers**. Given random variables $\{X_i\}$, define the sequence of averages as
$$\overline{X}_n = \frac{1}{n}(X_1 + \cdots + X_n).$$
Since each random variable is, by definition, a function on a sample space Ω, we can consider the pointwise limit
$$\lim_{n\to\infty} \overline{X}_n,$$
which is the random variable that for each $\omega \in \Omega$ takes the limit $\lim_{n\to\infty} \overline{X}_n(\omega)$ as value.[3]

[3]That this is indeed a random variable in the formal sense follows from measure theory, we will not be concerned with those details.

Theorem A.29 (Law of Large Numbers)**.** *Let $\{X_i\}$ be a sequence of i.i.d. random variables with $\mathbb{E}[X_1] = \mu < \infty$. Then the sequence of averages $\overline{X}_n$ converges almost surely to μ:*

$$\mathbb{P}(\lim_{n\to\infty} \overline{X}_n = \mu) = 1.$$

Example A.30. Let each X_i be a Bernoulli random variable with parameter p. One could think this as flipping a coin that will show heads with probability p. Then $\overline{X}_n$ is the *average* number of heads when flipping the coin n times. The law of large numbers asserts that as n increases, this average approaches p almost surely. Intuitively, when flipping the coin a billion times, the number of heads we get divided by a billion will be indistinguishable from p: if we do not know p we can estimate it in this way.

Some useful inequalities

In applications it is often not possible to get precise expressions for a probability we are interested in, most often because we don't know the exact distribution we are dealing with and only have access to parameters such as the expectation or the variance. There are several useful inequalities that help us bound the tail or deviation probabilities. For the following, we assume $\mathcal{X} \subset \mathbb{R}$.

- **Jensen's Inequality** Let $f\colon \mathbb{R} \to \mathbb{R}$ be a convex function, that is, $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$ for $\lambda \in [0,1]$. Then
$$f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)].$$
- **Markov's Inequality** ("first moment method") For $X \geq 0$ and $\lambda > 0$,
$$\mathbb{P}(X \geq \lambda) \leq \frac{\mathbb{E}[X]}{\lambda}.$$
- **Chebyshev's Inequality** ("second moment method") For $\lambda > 0$,
$$\mathbb{P}(|X - \mathbb{E}[X]| \geq \lambda) \leq \frac{\mathrm{Var}(X)}{\lambda^2}.$$
- **Exponential Moment Inequality** For any $s, \lambda \geq 0$,
$$\mathbb{P}(X \geq \lambda) \leq \mathrm{e}^{-s\lambda}\,\mathbb{E}[\mathrm{e}^{sX}].$$

Note that both the Chebyshev and the exponential moment inequality follow from the Markov inequality applied to certain transformations of X.

The normal distribution

The normal, or Gaussian distribution deserves special attention. A random variable X on $\mathbb{R}$ is **normally distributed** or **Gaussian** with mean μ and variance σ^2, written $X \sim \mathcal{N}(\mu, \sigma^2)$, if it has density

$$\gamma_{\mu,\sigma^2}(x) = \frac{1}{\sqrt{2\pi}\sigma}\mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

If $X \sim \mathrm{N}(0,1)$, we denote by

$$\Phi(z) := \mathbb{P}(X \leq z) = \int_{-\infty}^{z} \gamma_{0,1}(x)\,\mathrm{d}x$$

the cumulative distribution function. There is not closed-form expression for this integral, and traditionally the values of this function would be computed numerically and recorded in tables. If $X \sim \mathrm{N}(\mu, \sigma^2)$, then $(X - \mu)/\sigma \sim \mathrm{N}(0, 1)$, so it suffices to have a table for Φ in the standard normal case (with $\mu = 0$ and $\sigma^2 = 1$). See Figure A.7 for an illustration of the Gaussian distribution. We note that while there is no closed-form expression for the distribution function, the moments are well known and easy to compute. The moment generating function of a Gaussian $X \sim \mathcal{N}(\mu, \sigma^2)$ is

$$\mathbb{E}[\mathrm{e}^{\lambda X}] = \mathrm{e}^{\mu\lambda + \frac{\sigma^2\lambda^2}{2}}.$$

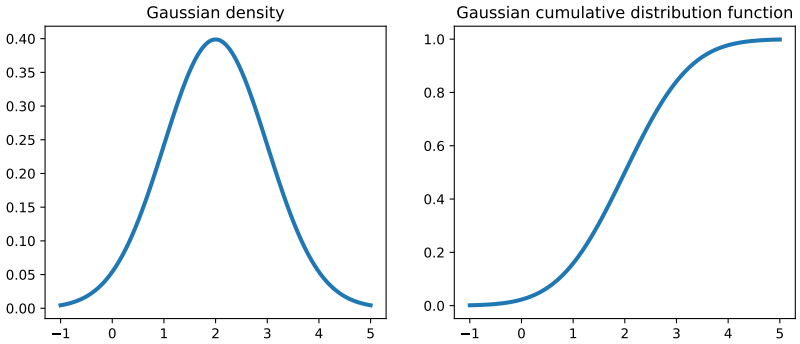


Figure A.7: The density [left] and distribution function [right] of a Gaussian random variable.

The importance of the Gaussian distribution stems from the **central limit theorem**, which states that the average of independent random samples tends to the normal distribution.

Theorem A.31. *(Central Limit Theorem) Let $X_1, \ldots, X_n$ be i.i.d. random variables with expected value μ and finite variance $\sigma^2 > 0$, and set $\overline{X}_n = (X_1 + \cdots + X_n)/n$. Then*

$$\lim_{n\to\infty} \mathbb{P}(\sqrt{n}(\overline{X}_n - \mu) \leq z) = \Phi(z/\sigma).$$

A multivariate Gaussian is a random vector $X = (X_1, \ldots, X_n)$ with density

$$\gamma(\boldsymbol{x}) = \frac{1}{(2\pi)^{n/2}\det(\boldsymbol{\Sigma})^{1/2}} \mathrm{e}^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\intercal \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}.$$

The vector μ is the mean and the matrix $\boldsymbol{\Sigma}$ is the **covariance matrix** of X. The multivariate Gaussian has important invariance property: if $X \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ and $\boldsymbol{Q} \in O(n)$ is orthogonal, then $\boldsymbol{Q}X \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$.

Conditional probability and expectation

Given events $A, B \subset \Omega$ with $\mathbb{P}(B) \neq 0$, the **conditional probability** of A conditioned on B is defined as

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}.$$

One interprets this as the probability of A if we assume B. That is, if we observed B, then we replace the whole set Ω by B and consider B to be the new space of events, considering only the part of events A that lie in B. We can rearrange the expression for conditional probability to

$$\mathbb{P}(A \cap B) = \mathbb{P}(A|B)\mathbb{P}(B),$$

from which we get the sometimes useful identity

$$\mathbb{P}(A) = \mathbb{P}(A|B)\mathbb{P}(B) + \mathbb{P}(A|B^c)\mathbb{P}(B^c), \tag{A.5}$$

where B^c denotes the complement of B in Ω.

Since by exchanging the role of A and B we get $\mathbb{P}(A \cap B) = \mathbb{P}(B|A)\mathbb{P}(A)$, we arrive at the famous **Bayes rule** for conditional probability:

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B)},$$

defined whenever both A and B have non-zero probability. These concepts clearly extend to random variables, where we can define, for example,

$$\mathbb{P}(X \in A|Y \in B) = \frac{\mathbb{P}(X \in A, Y \in B)}{\mathbb{P}(X \in B)}.$$

Note that if X and Y are independent, then the conditional probability is just the normal probability of X: knowing that $Y \in B$ does not give us any additional information about X! If we fix an event such as $\{Y \in B\}$, then we can define the **conditioning** of the random variable X to this event as the random variable X' with distribution

$$\mathbb{P}(X' \in A) = \mathbb{P}(X \in A|Y \in B).$$

In particular, $\mathbb{P}(X \in A|Y \in B) + \mathbb{P}(X \notin A|Y \in B) = 1$.

Example A.32. Consider the case of testing for doping at a sports event. Let X be the indicator variable for the presence of a certain drug, and Y the indicator variable for whether the person tested has taken the drug. Assume that the test is 99% accurate when the drug is present and 99% accurate when the drug is not present. We would like to know the probability that a person who tested positive actually took the drug, namely $\mathbb{P}(Y = 1|X = 1)$. Translated into probabilistic language, we know that

$$\begin{aligned} &\mathbb{P}(X = 1|Y = 1) = 0.99, \quad \mathbb{P}(X = 0|Y = 1) = 0.01 \\ &\mathbb{P}(X = 0|Y = 0) = 0.99, \quad \mathbb{P}(X = 1|Y = 0) = 0.01. \end{aligned}$$

Assuming that only 1% of the participants have taken the drug, which translates to $\mathbb{P}(Y = 1) = 0.01$, we find that the overall probability of a positive test result is, using (A.5),

$$\begin{aligned} \mathbb{P}(X = 1) &= \mathbb{P}(X = 1|Y = 0)\mathbb{P}(Y = 0) + \mathbb{P}(X = 1|Y = 1)\mathbb{P}(Y = 1) \\ &= 0.01 \cdot 0.99 + 0.99 \cdot 0.01 = 0.0198. \end{aligned}$$

Hence, using Bayes' rule, we conclude that

$$\mathbb{P}(Y = 1|X = 1) = \frac{\mathbb{P}(X = 1|Y = 1)\mathbb{P}(Y = 1)}{\mathbb{P}(X = 1)} = \frac{0.99 \cdot 0.01}{0.0198} = 0.5.$$

That is, we get the surprising result that even though our test is very unlikely to give false positives and false negatives, the probability that a person tested positive has actually taken the drug is only 50%. The reason is that the event itself is highly unlikely.

We now come to the notion of **conditional expectation**. Let X, Y be random variables. If X is discrete, then the conditional expectation of X conditioned on an event $Y = y$ is defined as

$$\mathbb{E}[X|Y = y] = \sum_k k\mathbb{P}(X = k|Y = y). \tag{A.6}$$

This is simply the expectation of the random variable X' with distribution $\mathbb{P}(X' \in A) = \mathbb{P}(X \in A | Y = y)$. Intuitively, we assume that $Y = y$ is given/has been observed, and consider the expectation of X under this additional knowledge.

Example A.33. Assume we are rolling dice, let X be the random variable giving the result, and let Y be the indicator variable for the event that the result is at most 4. Then $\mathbb{E}[X] = 3.5$ and $\mathbb{E}[X|Y = 1] = 2.5$ (verify this!). This is the expected value if we have the additional information that the result is at most 4.

In the absolutely continuous case we can define a conditional density

$$\rho_{X|Y=y}(x) = \frac{\rho_{X,Y}(x, y)}{\rho_Y(y)}, \tag{A.7}$$

where $\rho_{X,Y}$ is the joint density of (X, Y) and ρ_Y the density of Y. By a common abuse of notation, we will often write

$$\rho(x|y) := \rho_{X|Y=y}(x).$$

The conditional expectation is then defined

$$\mathbb{E}[X|Y = y] = \int_{\mathcal{X}} x\rho(x|y)\, \mathrm{d}x. \tag{A.8}$$

We replace the *density* ρ_X of X with an updated density $\rho_{X|Y=y}$ that takes into account that a value $Y = y$ has been observed when computing the expectation of X.

When looking at (A.6) and (A.8), we get a different number $\mathbb{E}[X|Y = y]$ for each $y \in \mathcal{Y}$, where we assume $\mathcal{Y}$ to be the space where Y takes values. Hence, we can *define* a random variable $\mathbb{E}[X|Y]$ on Y as follows:

$$\mathbb{E}[X|Y](y) = \mathbb{E}[X|Y = y].$$

If $X = f(Y)$ is completely determined by Y, then clearly

$$\mathbb{E}[X|Y](y) = \mathbb{E}[X|Y = y] = \mathbb{E}[f(Y)|Y = y] = \mathbb{E}[f(y)|Y = y] = f(y),$$

since the expected value of a constant is just that constant, and hence $\mathbb{E}[X|Y] = f(Y)$ as a random variable.

Using the definition of the conditional density (A.7), Fubini's Theorem and expression (A.8), we can write the expectation of X as

$$\begin{aligned}
\mathbb{E}[X] &= \int_{\mathcal{X}} x\rho_X(x)\, \mathrm{d}x \\
&= \int_{\mathcal{X}} x \int_{\mathcal{Y}} \rho_{(X,Y)}(x, y)\, \mathrm{d}y\, \mathrm{d}x \\
&= \int_{\mathcal{Y}} \left(\int_{\mathcal{X}} x\rho_{(X,Y)}(x, y)\, \mathrm{d}x \right) \mathrm{d}y \\
&= \int_{\mathcal{Y}} \left(\int_{\mathcal{X}} x\rho_{(X|Y=y)}(x)\, \mathrm{d}x \right) \rho_Y(y)\, \mathrm{d}y = \int_{\mathcal{Y}} \mathbb{E}[X|Y = y]\rho_Y(y)\, \mathrm{d}y.
\end{aligned}$$

One can interpret this as saying that we get the expected value of X by integrating the expected values conditioned on $Y = y$ with respect to the density of Y. In the discrete case, the identity has the form

$$\mathbb{E}[X] = \sum_y \mathbb{E}[X|Y = y]\mathbb{P}(Y = y).$$

The above identities can be written more compactly as

$$\mathbb{E}[\mathbb{E}[X|Y]] = \mathbb{E}[X].$$

In the context of machine learning, we assume that we have a (hidden) map $f\colon \mathcal{X} \to \mathcal{Y}$ from an input space to an output space, and that, for a given input $x \in \mathcal{X}$, the *observed* output is $y = f(x) + \epsilon$, where ϵ is random noise with $\mathbb{E}[\epsilon] = 0$. If we consider the input as a random variable X, then the output is random variable

$$Y = f(X) + \epsilon,$$

with $\mathbb{E}[\epsilon|X] = 0$ ("the expected value of ϵ, when knowing X, is zero"). We are interested in the value $\mathbb{E}[Y|X]$. For this, we get

$$\mathbb{E}[Y|X] = \mathbb{E}[f(X)|X] + \mathbb{E}[\epsilon|X] = f(X),$$

since $f(X)$ is completely determined by X.

5 Statistics

Statistics uses probability theory to analyse empirical data. Statistical inference aims to use data analysis to determine properties of some underlying probability distribution.

Estimation and the bias-variance trade-off

Suppose we observe samples $\{x_1, \ldots, x_n\} \subset \mathcal{X}$, where $\mathcal{X}$ is a space of outcomes of an experiment. In statistics, the assumption is often made that these samples are realizations of a random variable X, distributed according to some probability distribution. We may be interested in determining the whole distribution that gave rise to the data, or, as is more often the case, just some properties of it. The setting for a statistical estimation problem thus consists of:

- A space of outcomes $\mathcal{X}$;
- A parametrized family of probability distributions $\{\mathbb{P}_\theta\colon \theta \in \Theta\}$;
- A map $g\colon \Theta \to \mathcal{Y}$, mapping a parameter to a quantity of interest to us.

For example, if $\mathcal{X} = \mathbb{R}$, $\theta = (\mu, \sigma^2)$ and $\mathbb{P}_\theta = \mathcal{N}(\mu, \sigma^2)$, then $g(\theta) = g(\mu, \sigma^2) = \mu$ if we are only interested in the mean. A **point estimator** is any map $T\colon \mathcal{X}^n \to \mathcal{Y}$ that maps any collection of n data points to a "guess" of $g(\theta)$.

If $(X_1, \ldots, X_n)$ are i.i.d. random variables on $\mathcal{X}$ distributed according to $\mathbb{P}_\theta$ and $\mathcal{Y}$ is a measure space, then $T = T(X_1, \ldots, X_n)$ is itself a random variable on $\mathcal{Y}$. We denote by $\mathbb{E}_\theta$ and Var_θ the expectation and variance with respect to the distribution on $\mathcal{Y}$ induced by the distribution $\mathbb{P}_\theta$ on $\mathcal{X}$. The **mean square error (MSE)** of an estimator T is defined as

$$\mathbb{E}_\theta[(T - g(\theta))^2].$$

The **bias** of T is defined as

$$b_T(\theta) = \mathbb{E}_\theta[T] - g(\theta).$$

The **standard error** of T is defined as

$$\sigma_T(\theta) = \sqrt{\mathrm{Var}_\theta(T)}.$$

The bias-variance trade-off can then be described as decomposing the mean square error as a sum of bias and variance:

$$\mathbb{E}_\theta[(T - g(\theta))^2] = b_T(\theta)^2 + \mathrm{Var}_\theta(T).$$

Ideally, we would like the bias and the variance to be both small, but typically this is not possible. An estimator is called **unbiased** if $b_T(\theta) = 0$ for all θ. Variants of the bias-variance trade-off are of fundamental importance in machine learning.

Example A.34. Consider n coin flips, with outcomes $X = 1$ (head) or $X = 0$ (tail), where the probability of head is p. Assuming we are after p, the estimator is a map $T\colon \{0,1\}^n \to [0,1]$. The bias-variance trade-off is perhaps best illustrated by considering extreme cases. If we fixe some $p_0 \in [0,1]$ and take T to be the constant map, $T(x_1, \ldots, x_n) = p_0$, then clearly $\mathrm{Var}_p(T) = 0$, but the bias is $b_T(p) = p_0 - p$. A more principled approach is to choose

$$T(X_1, \ldots, X_n) = \frac{1}{n}\sum_{i=1}^{n} X_i.$$

Then $\mathbb{E}_p[T] = p$ and the method is therefore unbiased. However, the variance is

$$\mathrm{Var}_p(T) = \frac{p(1-p)}{n} > 0.$$

While the variance is positive, we see in this example that it convergence to 0 if we increase the number of samples n.

Example A.35. If $X_1, \ldots, X_n$ are i.i.d. $\mathcal{N}(\mu, \sigma^2)$ random variables, then

$$T(X_1, \ldots, X_n) = \overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$$

is an unbiased estimator of the mean μ, while

$$T(X_1, \ldots, X_n) = \frac{1}{n-1}\sum_{i=1}^{n}(X - \overline{X}_n)^2$$

is an unbiased estimator of the variance σ^2. Note the unusual choice of normalization: intuitively, one would think of dividing by n instead of $n-1$, but the factor $1/(n-1)$ is essential to make this estimator unbiased.

If we want to evaluate the behaviour of an estimator as the sample size n increases, we need to consider families of estimators $\{T_n\}$. A family of estimators $\{T_n\}$ is called **consistent**, if for all $\epsilon > 0$ and all $\theta \in \Theta$,

$$\lim_{n\to\infty} \mathbb{P}(|T_n - g(\theta)| > \epsilon) = 0.$$

The variance estimator in Example A.35 is consistent.

Maximum likelihood

Arguably the most important method for constructing an estimator is **maximum likelihood**. Assume the distribution induced by $\mathbb{P}_\theta$ on $\mathcal{X}^n$ has density $\rho_\theta(x_1, \ldots, x_n)$. For samples $(x_1, \ldots, x_n) \in \mathcal{X}^n$, the **likelihood function** is defined as

$$L\colon \Theta \to \mathbb{R}, \quad L(\theta) = \rho_\theta(x_1, \ldots, x_n).$$

The **maximum likelihood estimator** of a parameter θ is then defined as

$$T(x_1, \ldots, x_n) = \arg\max_{\theta} L(\theta).$$

In words, one selects the parameter θ for which the density $\rho_\theta(x_1, \ldots, x_n)$ is the largest, i.e., the density for which the observed data is "most likely". In practice, it is often more convenient to maximize the logarithm of the likelihood, also known as **log-likelihood**.

Example A.36. Consider the example with n coin flips from Example A.34. Here we use the probability function $\mathbb{P}(X_1 = x_1, \ldots, X_n = x_n)$ instead of the density, which is given by

$$L(p) = p^k(1-p)^{n-k}$$

if k entries are 1 and $n-k$ entries are 0. Maximizing the logarithm of this function gives

$$\arg\max_{p} \left(k \log(p) + (n-k)\log(1-p)\right) = \frac{k}{n} = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

This recovers the consistent estimator from Example A.34.

Example A.37. Consider the problem of estimating the mean μ of a normal distribution with unit variance from n samples $x_1, \ldots, x_n$. The likelihood function is then the density of a multivariate normal distribution, and the logarithm of this density is given

$$\log L(\mu) = -\sum_{i=1}^{n}(x_i - \mu)^2 + \text{constant}.$$

Maximizing the log-likelihood is therefore equivalent of solving a linear least squares problem, consisting of minimizing the average squared deviation from the data. The solution to this problem is precisely the sample mean $(1/n)\sum_{i=1}^{n} x_i$.

While in some cases the maximum likelihood estimator can be computed explicitly, in other cases iterative algorithms such as **expectation-maximization (EM)** are used.

Bayesian statistics

In Bayesian statistics, the parameters governing a distribution are themselves viewed as random variables, and the densities or probabilities that underlie observed data are interpreted as conditional densities or probabilities, conditioned on these parameters. For example, given a data set $\mathcal{D} = \{x_1, \ldots, x_n\}$, drawn independently from a distribution with density.

Conjugate priors

Maximum a posteriori estimation

In Bayesian statistics, probability is interpreted as a measure of uncertainty, rather than as a frequency. Moreover, the parameters that enter into a model are interpreted as random quantities in their own right. With this interpretation, densities parametrized by a set Θ are interpreted as conditional densities $\rho(x|\theta)$. Similarly, the **likelihood function** is interpreted as a conditional density

$$\rho(x_1, \ldots, x_n|\theta).$$

The assumed density $g(\theta)$ on Θ is called teh **prior density**. The method of **maximum a posteriori estimation (MAP)** aims to estimate θ my maximizing the **posterior density** $\rho(\theta|x_1, \ldots, x_n)$, which can be expressed in terms of the likelihood and the prior distribution using Bayes' Theorem. Getting rid of constants that do not influence the resulting maximization problem, we can define the MAP estimator as

$$T_{\mathrm{MAP}}(x_1, \ldots, x_n) = \arg\max_{\theta} \rho(x_1, \ldots, x_n|\theta)g(\theta).$$

We see that the difference to maximum likelihood is seen through the presence of the prior density. As with maximum-likelihood, a variant of the expectation-maximization algorithm can be used to compute the MAP estimator.

6 Finite precision arithmetic

In practical applications, one often cannot simply plug numbers into formulae and get all the exact results. Most numerical data also requires an infinite amount of storage (just try to store π on a computer!), but a piece of paper or a computer only has limited space. These are some of the reasons that lead us to work with **approximations**.

Measuring errors

To measure the quality of approximations, we use the concept of **relative error**. Given a quantity x and a computed approximation $\hat{x}$, the **absolute error** is given by

$$E_{\mathrm{abs}}(\hat{x}) = |x - \hat{x}|,$$

while the *relative error* is given as

$$E_{\mathrm{rel}}(\hat{x}) = \frac{|x - \hat{x}|}{|x|}.$$

The benefit of working with relative errors is that they are scale invariant. Absolute errors can be meaningless at times: for example, an error of one hour is irrelevant when estimating the age of Stan the Tyrannosaurus Rex at Manchester Museum, but it is crucial when determining the time of a lecture. That is because in the former one hour corresponds to a relative error is of the order 10^{-11}, while in the latter it is of the order 1.

Floating point and significant figures

The established way of representing real numbers on computers is using **floating-point arithmetic**. In the double precision version of the IEEE standard for floating-point arithmetic, a number is represented using 64 bits, where a bit is either 1 or 0. A number is written

$$x = \pm f \times 2^e,$$

where f is a fraction in $[0, 1]$, represented using 52 bits, and e is the exponent, using 11 bits, and one bit is for the sign. There are largest possible numbers, and there are gaps between representable numbers. The largest and smallest numbers representable in this form are of the order of $\pm 10^{308}$, enough for most practical purposes. A bigger concern are the gaps, which means that the results of many computations almost always have to be rounded to the closest floating-point number.

When going through calculations without using a computer, we usually use the terminology of **significant figures** (s.f.) and work with 4 significant figures in base 10. For example, in base 10, $\sqrt{3}$

equals 1.732 to 4 significant figures. To count the number of significant figures in a given number, start with the first non-zero digit from the left and, moving to the right, count all the digits thereafter, counting final zeros if they are to the right of the decimal point. For example, 1.2048, 12.040, 0.012048, 0.0012040 and 1204.0 all have 5 significant figures (s.f.). In rounding or truncation of a number to n s.f., the original is replaced by the closest number with n s.f. An approximation $\hat{x}$ of a number x is said to be **correct to** n **significant figures** if both $\hat{x}$ and x round to the same n s.f. number.

Remark A.38. Note that final zeros to left of the decimal point may or may not be significant: the number 1204000 has a least 4 significant figures, but without any more information there is no way of knowing whether or not any more figures are significant. When 1203970 is rounded to 5 significant figures to give 1204000, an explanation that this has 5 significant figures is required. This could be made clear by writing it in scientific notation: 1.2040×10^6. In some cases we also have to agree whether to round up or round down: for example, 1.25 could equal 1.2 or 1.3 to two significant figures. If we agree on rounding up, then to say that $a = 1.2048$ to 5 s.f. means that the exact value of a satisfies $1.20475 \leq a < 1.40485$.

Example A.39. Suppose we want to find the solution to the quadratic equation

$$ax^2 + bx + c = 0.$$

The two solutions to this problem are given by

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}. \tag{A.1}$$

In principle, to find x_1 and x_2 one only needs to evaluate the expressions for given a, b, c. Assume, however, that we are only allowed to compute to four significant figures, and consider the particular equation

$$x^2 + 39.7x + 0.13 = 0.$$

Using the formula A.1, we have, always rounding to four significant figures,

$$a = 1, b = 39.7, c = 0.13,$$

$$b^2 = 1576.09 = 1576 \text{ (to 4 s.f.) }, 4ac = 0.52 \text{ (to 4 s.f.)},$$

$$b^2 - 4ac = 1575.48 = 1575 \text{ (to 4 s.f.) }, \sqrt{b^2 - 4ac} = 39.69.$$

Hence, the computed solutions (to 4 significant figures) are given by

$$\overline{x}_1 = -0.005, \ \overline{x}_2 = -39.69$$

The exact solutions, however, are

$$x_1 = -0.0032748..., \ x_2 = -39.6907...$$

The solution x_1 is completely wrong, at least if we look at the relative error:

$$\frac{|\overline{x}_1 - x_1|}{|x_1|} = 0.5268.$$

While the accuracy can be increased by increasing the number of significant figures during the calculation, such effects happen all the time in scientific computing and the possibility of such effects has to be taken into account when designing numerical algorithms.

By analysing what causes the error it is sometimes possible to modify the method of calculation in order to improve the result. In the present example, the problems are being caused by the fact that $b \approx \sqrt{b^2 - 4ac}$, and therefore

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a} = \frac{-39.7 + 39.69}{2}$$

causes what is called “catastrophic cancellation”. A way out is provided by the observation that the two solutions are related by

$$x_1 \cdot x_2 = \frac{c}{a}. \tag{A.2}$$

When $b > 0$, the calculation of x_2 according to (A.1) shouldn't cause any problems, in our case we get -39.69 to four significant figures. We can then use (A.2) to derive $\overline{x}_1 = c/(a\overline{x}_2) = -0.00327$.

There are other potential sources of error besides those introduced by rounding operations.

1. Overflow
2. Errors in the model
3. Human or measurements errors
4. Truncation or approximation errors

The first is rarely an issue, as we can represent numbers of order 10^{308} on a computer. The second two are important factors that need to be addressed when working on real-world problems. The fourth has to do with the fact that many computations are done approximately rather than exactly. For computing the exponential, for example, we might use a method that gives the approximation

$$\mathrm{e}^x \approx 1 + x + \frac{x^2}{2}.$$

As it turns out, many optimization problems work with approximations of the functions of interest, and the solution found is only an approximation to the “true" solution of the problem. Quantifying the quality of such an approximation is an important aspect in the design and analysis of optimization algorithms.

Bibliography

[1] Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tohoku Mathematical Journal, Second Series*, 19(3):357–367, 1967.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Peter L Bartlett, Stéphane Boucheron, and Gábor Lugosi. Model selection and error estimation. *Machine Learning*, 48(1-3):85–113, 2002.

[4] Peter L Bartlett and Shahar Mendelson. Rademacher and Gaussian complexities: Risk bounds and structural results. *The Journal of Machine Learning Research*, 3:463–482, 2003.

[5] Alexander Barvinok. *A course in convexity*, volume 54 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2002.

[6] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[7] Richard Bellman. Dynamic programming. *science*, 153(3731):34–37, 1966.

[8] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[9] Jakob Bernoulli. *Ars conjectandi, opus posthumum. Accedit Tractatus de seriebus infinitis, et epistola gallicé scripta de ludo pilae reticularis*. Thurneysen Brothers, Basel, 1713.

[10] Dimitri P Bertsekas. *Convex optimization theory*. Athena Scientific Belmont, 2009.

[11] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

[12] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer, 1998.

[13] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[14] Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press, 2013.

[15] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, Cambridge, 2004.

[16] Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[17] Peter Bürgisser and Felipe Cucker. *Condition: The geometry of numerical algorithms*. Number 349 in Grundlehren der Mathematischen Wissenschaften. Springer Verlag, 2013.

[18] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4):493–507, 1952.

[19] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[20] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[21] Nello Cristianini, John Shawe-Taylor, et al. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[22] Felipe Cucker and Ding Xuan Zhou. *Learning theory: an approximation theory viewpoint*, volume 24. Cambridge University Press, 2007.

[23] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[24] George Bernard Dantzig. *Linear programming and extensions*, volume 48. Princeton university press, 1998.

[25] Luc Devroye, László Györfi, and Gábor Lugosi. *A probabilistic theory of pattern recognition*, volume 31. Springer Science & Business Media, 2013.

[26] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

[27] Alhussein Fawzi, Hamza Fawzi, and Omar Fawzi. Adversarial vulnerability for any classifier. In *Advances in Neural Information Processing Systems*, pages 1178–1187, 2018.

[28] Simon Foucart and Holger Rauhut. *A mathematical introduction to compressive sensing*, volume 336 of *Applied and Numerical Harmonic Analysis*. Birkhäuser, Basel, 2013.

[29] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[30] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[31] Eduard Gorbunov, Filip Hanzely, and Peter Richtárik. A unified theory of SGD: Variance reduction, sampling, quantization and coordinate descent. *arXiv preprint arXiv:1905.11261*, 2019.

[32] Robert Mansel Gower, Nicolas Loizou, Xun Qian, Alibek Sailanbayev, Egor Shulgin, and Peter Richtárik. Sgd: General analysis and improved rates. *arXiv preprint arXiv:1901.09401*, 2019.

[33] Andreas Griewank. Who invented the reverse mode of differentiation. *Documenta Mathematica, Extra Volume ISMP*, pages 389–400, 2012.

[34] Ingo Gühring, Mones Raslan, and Gitta Kutyniok. Expressivity of deep neural networks. *arXiv preprint arXiv:2007.04759*, 2020.

[35] T. Hastie, J. Friedman, and R. Tibshirani. *The elements of statistical learning*, volume 2. Springer, 2009.

[36] Magnus R Hestenes, Eduard Stiefel, et al. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409–436, 1952.

[37] Catherine F Higham and Desmond J Higham. Deep learning: An introduction for applied mathematicians. *SIAM Review*, 61(4):860–891, 2019.

[38] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[39] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[40] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[41] Ronald A Howard. Dynamic programming and markov processes. 1960.

[42] Michael I. Jordan. Artificial intelligence—the revolution hasn't happened yet. *Harvard Data Science Review*, 1(1), 7 2019. https://hdsr.mitpress.mit.edu/pub/wot7mkc1.

[43] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[44] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[45] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[47] Harold W Kuhn and Albert W Tucker. Nonlinear programming. In *Traces and emergence of nonlinear programming*, pages 247–258. Springer, 2014.

[48] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[49] Michel Ledoux. *The concentration of measure phenomenon*, volume 89 of *Mathematical Surveys and Monographs*. American Mathematical Society, Providence, RI, 2001.

[50] Michel Ledoux and Michel Talagrand. *Probability in Banach Spaces: isoperimetry and processes*. Springer Science & Business Media, 2013.

[51] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[52] Stephane Mallat. *A wavelet tour of signal processing: the sparse way*. Academic press, 2008.

[53] Jiří Matoušek. *Lectures on discrete geometry*, volume 212 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2002.

[54] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[55] Colin McDiarmid. Concentration. In *Probabilistic methods for algorithmic discrete mathematics*, pages 195–248. Springer, 1998.

[56] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2012.

[57] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.

[58] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[59] Yurii Nesterov. *Lectures on convex optimization*, volume 137. Springer, 2018.

[60] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006.

[61] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[62] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[63] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

[64] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[65] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[66] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[67] Yousef Saad. Iterative methods for linear systems of equations: A brief historical journey. *arXiv preprint arXiv:1908.01083*, 2019.

[68] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, pages 71–105, 1959.

[69] Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, 1972.

[70] Juergen Schmidhuber. Unsupervised minimax: Adversarial curiosity, generative adversarial networks, and predictability minimization. *arXiv preprint arXiv:1906.04493*, 2019.

[71] Bernhard Schölkopf, Alexander J Smola, Francis Bach, et al. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[72] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, 2014.

[73] Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972.

[74] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[75] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014.

[76] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[77] Lloyd N Trefethen. *Approximation theory and approximation practice*, volume 128. Siam, 2013.

[78] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[79] V. N. Vapnik and A. Ya. Chervonenkis. *Theory of Pattern Recognition [in Russian]*. Nauka, USSR, 1974.

[80] Vladimir Vapnik. *The nature of statistical learning theory*. Springer, 2013.

[81] VN Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971.

[82] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[83] Roman Vershynin. *High-dimensional probability: An introduction with applications in data science*, volume 47. Cambridge University Press, 2018.

[84] Martin J Wainwright and Michael Irwin Jordan. *Graphical models, exponential families, and variational inference*. Now Publishers Inc, 2008.

[85] Karl Weierstrass. Über die analytische Darstellbarkeit sogenannter willkürlicher Functionen einer reellen Veränderlichen. *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin*, 2:633–639, 1885.