MATHEMATICS of MACHINE LEARNING

by

MARTIN LOTZ
Mathematics Institute
The University of Warwick

March 2020

Contents

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**, 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 recent success of DeepMind's AlphaGo [1], a computer program based on reinforcement learning, at achieving super-human performance at the board game Go (围棋). Even though the rules of the game are simple, the task of beating the best human players seemed impossible only a decade 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 basic examples.

In its most basic form, the goal of 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 usually modelled as a metric space (such as $\mathbb{R}^d$), and the inputs could represent images, texts, emails, gene 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] https://deepmind.com/research/case-studies/alphago-the-story-so-far

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. This course will deal mainly with supervised learning.

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.

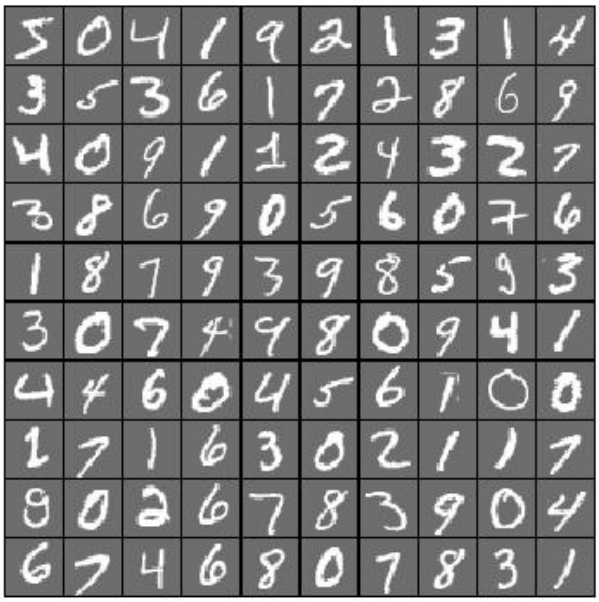


Figure 1.1: The MNIST (Modified National Institute of Standards and Technology, http://yann.lecun.com/exdb/mnist/) dataset is a large collection of images of hand-written digits, and is a frequently used benchmark in machine learning.

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

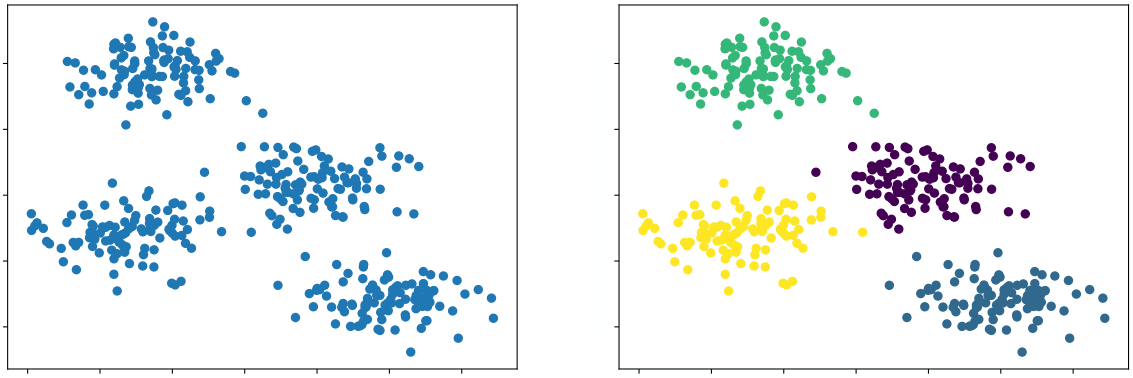


Figure 1.2: A collection of random points on the plane. The image on the right shows the four clusters as determined by the k-means algorithm.

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 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 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 $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^\top$ 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.

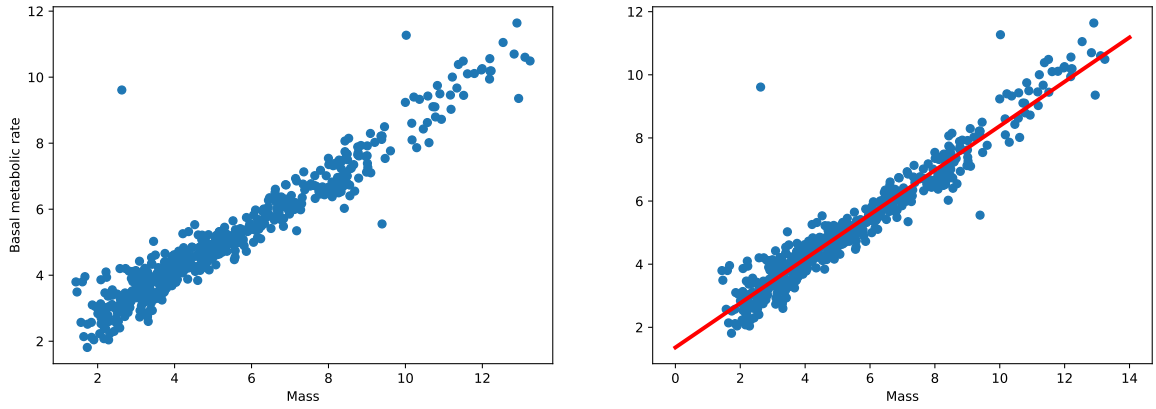


Figure 1.3: The relationship of mass to the logarithm of the basal metabolic rate in mammals. The data consists of 573 samples taken from the PanTHERA database, and the example featured in the episode Size Matters of the BBC series Wonders of Life. The right images shows the regression line determined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

	Goal	Soup
Article 1	5	0
Article 2	1	7

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) [2], 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

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

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) Neural networks are functions of the form

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

where each f_k is the componen-wise composition of an **activation function** σ with a linear map $\mathbb{R}^{d_{k-1}} \to \mathbb{R}^{d_k}, \boldsymbol{x} \mapsto \boldsymbol{W}^k \boldsymbol{x} + \boldsymbol{b}^k$. An activation function could be the **sigmoid** $\sigma(x) = 1/(1 + \mathrm{e}^{-x})$, which takes values between $(0, 1)$, and which can be interpreted as "selecting" certain coordinates of a vector depending on whether they are positive or negative (see Figure 1.4). 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 1.5. 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.

2

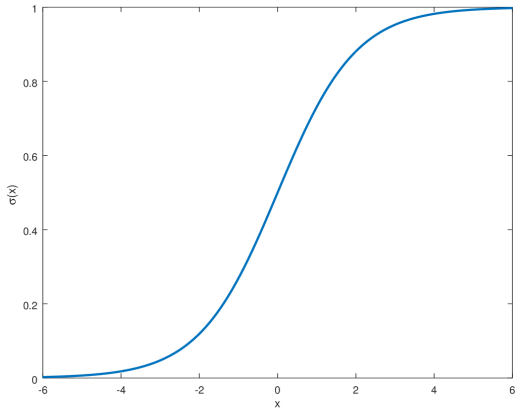


Figure 1.4: The sigmoid function

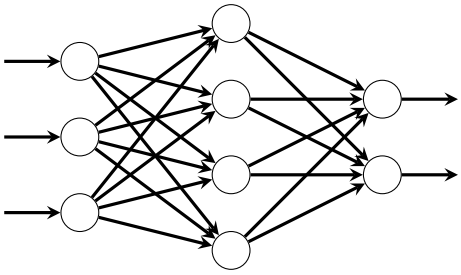


Figure 1.5: A neural network. Each layer correspond to applying a linear map to the outputs of the previous layer, followed by an activation function. Each arrow represents a *weight*. For example, the transition from the first layer to the second is a map $\mathbb{R}^3 \to \mathbb{R}^4$, and the weight 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.

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*

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

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., for 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 of candidate function $\mathcal{H}$ 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. The biggest challenge 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 data:

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

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 1.6 shows an example in which data from a function with noise is approximated by polynomials of different degrees.

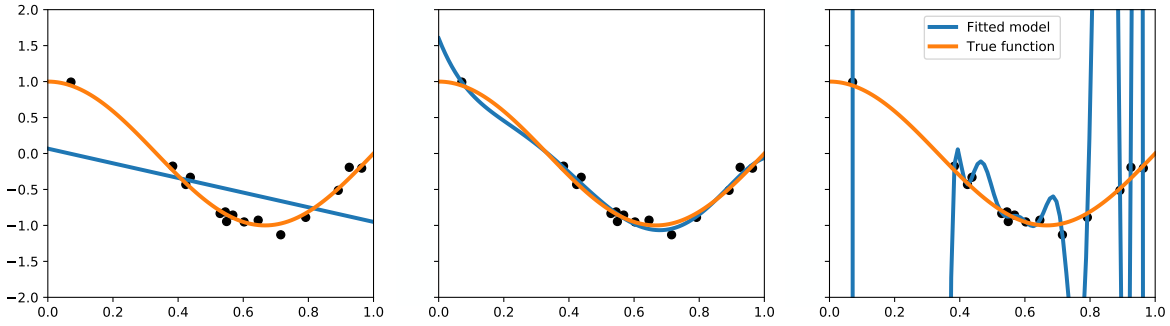


Figure 1.6: The data consists of 15 samples from the graph of a cosine function with added noise. The three displays show an approximation with a linear function, with a polynomial of degree 5, and with a polynomial of degree 15. The linear function has a large error on both the training set and in relation 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.

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 [31], see [28, 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 [16], followed by Rosenblatt's Perceptron [22]. The term "Machine Learning" was first coined by Arthur Samuel in 1959 [24]; 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 [21]. The field of Statistical Learning Theory arose in the 1960s through seminal work by Vapnik and Chervonenkis, see [29] for an overview. For an account of mathematical learning theory, see [5].

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.

2

Overview of Probability

In this lecture we review relevant notions from probability theory, with an emphasis on conditional expectation.

Probability spaces

A **probability space** is a triple $(\Omega, \mathcal{F}, \mathbb{P})$, consisting of a set Ω, a σ-algebra $\mathcal{F}$ of 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 countable collection $\{A_i\}$ 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**. 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 **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 $\{0, 1, \ldots, k\}$. 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, lower-case letters x, y, z for the values that these can take, and $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ if these are vectors in some $\mathbb{R}^d$.

Example 2.1. 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 measure 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) $\mathbb{P}(X \leq t)$ for $t \in \mathbb{R}$. The complement, $\mathbb{P}(X > t)$ (or $\mathbb{P}(X \geq t)$), 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 2.2. 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

  $$\mathbb{P}(X = k) = \binom{n}{k} p^k (1 - p)^{n-k} \tag{2.1}$$

  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.

Expectation

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 2.3. 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 2.4. 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 (2.1), 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

$$\mathbb{E}[F(X)] = \int_{\mathcal{X}} F(x) \rho(x) \, \mathrm{d}x \tag{2.2}$$

in the case of an absolutely continuous random variable, and similarly in the discrete case. [1]

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

$$\mathbb{E}[\mathbf{1}\{X \in A\}] = \mathbb{P}(X \in A), \tag{2.3}$$

as can be seen by applying (2.2) to the indicator function. The identity (2.3) 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 2.5. 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).

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.

[1] 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.

Example 2.6. 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.[2]

Theorem 2.7 (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 2.8. 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
$$f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)].$$

[2] That this is indeed a random variable in the formal sense follows from measure theory, we will not be concerned with those details.

- **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.

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

$$\mathbb{P}(A \mid B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}.$$

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

$$\mathbb{P}(A \cap B) = \mathbb{P}(A \mid B)\mathbb{P}(B),$$

from which we get the sometimes useful identity

$$\mathbb{P}(A) = \mathbb{P}(A \mid B)\mathbb{P}(B) + \mathbb{P}(A \mid B^c)\mathbb{P}(B^c), \tag{2.4}$$

where B^c denotes the complement of B in Ω.

Since by exchanging the role of A and B we get $\mathbb{P}(A \cap B) = \mathbb{P}(B \mid A)\mathbb{P}(A)$, we arrive at the famous **Bayes rule** for conditional probability:

$$\mathbb{P}(A \mid B) = \frac{\mathbb{P}(B \mid A)\mathbb{P}(A)}{\mathbb{P}(B)},$$

defined whenever both A and B have non-zero probability. These concepts clearly extend to random variables, where we can define, for example,

$$\mathbb{P}(X \in A \mid Y \in B) = \frac{\mathbb{P}(X \in A, Y \in B)}{\mathbb{P}(X \in B)}.$$

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

$$\mathbb{P}(X' \in A) = \mathbb{P}(X \in A \mid Y \in B).$$

In particular, $\mathbb{P}(X \in A \mid Y \in B) + \mathbb{P}(X \notin A \mid Y \in B) = 1$.

Example 2.9. 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 $\mathbb{P}(Y = 1 \mid X = 1)$. Translated into probabilistic language, we know that

$$\begin{aligned} \mathbb{P}(X = 1 \mid Y = 1) = 0.99, \quad \mathbb{P}(X = 0 \mid Y = 1) = 0.01 \\ \mathbb{P}(X = 0 \mid Y = 0) = 0.99, \quad \mathbb{P}(X = 1 \mid Y = 0) = 0.01. \end{aligned}$$

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 (2.4),

$$\begin{aligned} \mathbb{P}(X = 1) &= \mathbb{P}(X = 1 \mid Y = 0)\mathbb{P}(Y = 0) + \mathbb{P}(X = 1 \mid Y = 1)\mathbb{P}(Y = 1) \\ &= 0.01 \cdot 0.99 + 0.99 \cdot 0.01 = 0.0198. \end{aligned}$$

Hence, using Bayes' rule, we conclude that

$$\mathbb{P}(Y = 1 \mid X = 1) = \frac{\mathbb{P}(X = 1 \mid Y = 1)\mathbb{P}(Y = 1)}{\mathbb{P}(X = 1)} = \frac{0.99 \cdot 0.01}{0.0198} = 0.5.$$

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

$$\mathbb{E}[X \mid Y = y] = \sum_k k\mathbb{P}(X = k \mid Y = y). \tag{2.5}$$

This is simply the expectation of the random variable X' with distribution $\mathbb{P}(X' \in A) = \mathbb{P}(X \in A \mid Y = y)$. Intuitively, we assume that $Y = y$ is given/has been observed, and consider the expectation of X under this additional knowledge.

Example 2.10. 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 $\mathbb{E}[X \mid Y = 1] = 2.5$ (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

$$\rho_{X|Y=y}(x) = \frac{\rho_{X,Y}(x, y)}{\rho_Y(y)}, \tag{2.6}$$

where $\rho_{X,Y}$ is the joint density of (X, Y) and ρ_Y the density of Y. The conditional expectation is then defined

$$\mathbb{E}[X \mid Y = y] = \int_{\mathcal{X}} x\rho_{X|Y=y}(x)\,\mathrm{d}x. \tag{2.7}$$

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 (2.5) and (2.7), we get a different number $\mathbb{E}[X \mid Y = y]$ 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 $\mathbb{E}[X \mid Y]$ on Y as follows:

$$\mathbb{E}[X \mid Y](y) = \mathbb{E}[X \mid Y = y].$$

If $X = f(Y)$ is completely determined by Y, then clearly

$$\mathbb{E}[X \mid Y](y) = \mathbb{E}[X \mid Y = y] = \mathbb{E}[f(Y) \mid Y = y] = \mathbb{E}[f(y) \mid Y = y] = f(y),$$

since the expected value of a constant is just that constant, and hence $\mathbb{E}[X \mid Y] = f(Y)$ as a random variable.

Using the definition of the conditional density (2.6), Fubini's Theorem and expression (2.7), we can write the expectation of X as

$$\begin{aligned}
\mathbb{E}[X] &= \int_{\mathcal{X}} x\rho_X(x)\,\mathrm{d}x \\
&= \int_{\mathcal{X}} x \int_{\mathcal{Y}} \rho_{(X,Y)}(x,y)\,\mathrm{d}y\,\mathrm{d}x \\
&= \int_{\mathcal{Y}} \left(\int_{\mathcal{X}} x\rho_{(X,Y)}(x,y)\,\mathrm{d}x \right) \mathrm{d}y \\
&= \int_{\mathcal{Y}} \left(\int_{\mathcal{X}} x\rho_{(X|Y=y)}(x)\,\mathrm{d}x \right) \rho_Y(y)\,\mathrm{d}y = \int_{\mathcal{Y}} \mathbb{E}[X \mid Y = y]\rho_Y(y)\,\mathrm{d}y.
\end{aligned}$$

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

$$\mathbb{E}[X] = \sum_y \mathbb{E}[X \mid Y = y]\mathbb{P}(Y = y).$$

The above identities can be written more compactly as

$$\mathbb{E}[\mathbb{E}[X \mid Y]] = \mathbb{E}[X].$$

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 $\mathbb{E}[\epsilon \mid X] = 0$ ("the expected value of ϵ, when knowing X, is zero"). We are interested in the value $\mathbb{E}[Y \mid X]$. For this, we get

$$\mathbb{E}[Y \mid X] = \mathbb{E}[f(X) \mid X] + \mathbb{E}[\epsilon \mid X] = f(X),$$

since $f(X)$ is completely determined by X.

Concentration of measure

A very important refinement of Chebyshev's inequality are **concentration inequalities**, which state that the probability of exceeding the expectation is *exponentially small*. A prototype of such an inequality is **Hoeffding's Bound**.

Theorem 2.11 (Hoeffding's Inequality)**.** *Let $X_1, \ldots, X_n$ be independent random variables taking values in $[0,1]$, and let $S_n = \frac{1}{n}\sum_{i=1}^n X_i$ be the average. Then for $t \geq 0$,*

$$\mathbb{P}\left(|S_n - \mathbb{E}[S_n]| \geq t\right) \leq 2\mathrm{e}^{-2nt^2}.$$

Note the implication of this: if we have a sequence of random variables $\{X_i\}$ bounded in $[0, 1]$ (for example, the result of repeated, identical experiments or observations) then as n increases, the probability that the *average* of the random variables deviates from its mean decreases exponentially with n. In particular, if the random variables all thave the same expectation μ, then (by linearity of expectation) we have $\mathbb{E}[\overline{X}_n] = \mu$, and the probability of straying from this value becomes very small very quickly!

Notes

Even though probability theory and statistics are central to machine learning, probabilistic concepts are often not used rigorously. For example, one frequently encounters expressions such as $\mathbb{P}(X \mid Y)$ which, taken literally, do not make sense. Depending on context, such an expression may refer to either the conditional expectation $\mathbb{E}[X \mid Y]$, the conditional probability $\mathbb{P}(X \in A \mid Y \in B)$, or the conditional density $\rho_{X|Y=y}(x)$. It turns out that for most practical purposes it does not really matter, but it is just something that a mathematics student used to rigorous definitions should be aware of.

A good general introduction to probability theory is §1.1 of [27]. Good references for concentration of measure and related topics are [3, 30].

Part I

Statistical Learning Theory

3

Binary Classification

In this lecture we begin the study of statistical learning theory in the case of binary classification. We will characterize the best possible classifier in the binary case, and relate notions of classification error to each other.

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 we will also consider classifiers taking values in $\{-1, 1\}$ where convenient. 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. Typically, 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 if $p \leq 1/2$ it is assigned to class 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 function 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 a set of observations

$$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n \subset \mathcal{X} \times \mathcal{Y}. \tag{3.1}$$

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 (3.1) are drawn from a probability distribution on $\mathcal{X} \times \mathcal{Y}$, and replace each data point $(\boldsymbol{x}_i, y_i)$ in (3.1) with a copy (X_i, Y_i) of a random variable (X, Y) on $\mathcal{X} \times \mathcal{Y}$. We are after a classifier h such that the expected value of the **empirical risk**

$$\hat{R}(h) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{h(X_i) \neq Y_i\} \tag{3.2}$$

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 the **risk**: it is the probability that the classifier gets something wrong.

Example 3.1. 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**

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

Note that if we write

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

then $\mathbb{E}[\epsilon|X] = 0$, because

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

The Bayes classifier

While in Example 3.1 we could choose (at least in principle) $h(x) = f(x)$ and get $R(h) = 0$, in the presence of noise this is not possible. However, we could define a classifier h^* by setting

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

We call this the **Bayes classifier**. The following result shows that this is the best possible classifier.

Theorem 3.2. *The Bayes classifier h^* satisfies*

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

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

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

$$R(h) = \mathbb{E}[\mathbf{1}\{h(X) \neq Y\}] = \mathbb{E}\left[\mathbb{E}[\mathbf{1}\{h(X) \neq Y\}|X]\right]. \tag{3.3}$$

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 $\mathbb{E}[Y\mathbf{1}\{h(x) = 0\}|X = x]$, and will therefore only be non-zero if $h(x) = 0$. We can therefore pull the indicator function out of the expectation.

Hence, using (3.3),

$$R(h) = \mathbb{E}[\underbrace{\mathbf{1}\{h(X) = 1\}(1 - f(X))}_{(1)} + \underbrace{\mathbf{1}\{h(X) = 0\}f(X)}_{(2)}]. \tag{3.4}$$

For (1), we decompose

$$\begin{aligned}
\mathbf{1}\{h(X) = 1\}(1 - f(X)) =& \mathbf{1}\{h(X) = 1, f(X) > 1/2\}(1 - f(X)) \\
&+ \mathbf{1}\{h(X) = 1, f(X) \leq 1/2\}(1 - f(X)) \\
\geq& \mathbf{1}\{h(X) = 1, f(X) > 1/2\}(1 - f(X)) \\
&+ \mathbf{1}\{h(X) = 1, f(X) \leq 1/2\}f(X),
\end{aligned} \tag{3.5}$$

where the inequality follows since $(1 - f(X)) \geq f(X)$ if $f(X) \leq 1/2$. By the same reasoning, for (2) we get

$$\begin{aligned}
\mathbf{1}\{h(X) = 0\}f(X) \geq& \mathbf{1}\{h(X) = 0, f(X) > 1/2\}(1 - f(X)) \\
&+ \mathbf{1}\{h(X) = 0, f(X) \leq 1/2\}f(X).
\end{aligned} \tag{3.6}$$

Combining the inequalities (3.5) and (3.6) within the bound (3.4) and collecting the terms that are multiplied with $f(X)$ and those that are multiplied with $(1 - f(X))$, we arrive at

$$\begin{aligned}
R(h) &\geq \mathbb{E}[\mathbf{1}\{f(X) \geq 1/2\}f(X) + \mathbf{1}\{f(X) > 1/2\}(1 - f(X))] \\
&= \mathbb{E}[\mathbf{1}\{h^*(X) = 0\}f(X) + \mathbf{1}\{h^*(X) = 1\}(1 - f(X))] = R(h^*),
\end{aligned}$$

where the last equality follows from (3.4) applied to $h = h^*$. The characterization (3.4) also shows that

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

which completes the proof. □

We have seen in Example 3.1 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, which means that we have a 50% chance of being correct!

The error

$$\mathcal{E}(h) = R(h) - R(h^*)$$

is called the **excess risk** or error of h with respect to the best possible classifier.

Approximation and Estimation

We conclude this lecture by relating notions of risk. In what follows, we assume that a class of classifiers $\mathcal{H}$ is given, from which we are allowed to choose. We denote by $\hat{h}_n$ the classifier obtained by minimizing the empirical risk $\hat{R}(h)$ over $\mathcal{H}$, that is

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

where the (X_i, Y_i) are i.i.d. copies of a random variable (X, Y) on $\mathcal{X} \times \mathcal{Y}$. Note that $\hat{h}_n$ is what we will typically obtain by computation from samples (x_i, y_i). The way it is defined, it depends on n, the class of functions $\mathcal{H}$, and the random variables (X_i, Y_i), and as such is itself a random variable.

We want $\hat{h}_n$ to *generalize* well, that is, we want

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

to be small. We know that the smallest possible value this risk can attain is given by $R(h^*)$, where h^* is the Bayes classifier. We can decompose the difference between the risk of $\hat{h}_n$ and that of the Bayes classifier as follows:

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

The first compares the performance of $\hat{h}_n$ against the best possible classifier *within the class* $\mathcal{H}$, while the second is a statement about the power of the class $\mathcal{H}$. We can reduce the estimation error by making the class $\mathcal{H}$ smaller, but then the approximation error increases. Ideally, we would like to find bounds on the estimation error that converge to 0 as the number of samples n increases.

Notes

4

Finite Hypothesis Sets

Given a fixed hypothesis set $\mathcal{H}$, we would like to study the classifier $\hat{h}$ computed from n random samples (X_i, Y_i) by minimizing the empirical risk over the class $\mathcal{H}$,

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

Hence, for any h, $\hat{R}(h)$ is a random variable that depends on the number of samples n and on the underlying probability distribution. The classifier $\hat{h}$ is also a random variable, and depends on n, the class $\mathcal{H}$, and the underlying distribution. This is the object that *we can compute* from observed data. If h^* denotes the Bayes classifier, then we would like to bound the **excess risk**

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

where we recall that $R(h) = \mathbb{P}(h(X) \neq Y)$. As opposed to $\hat{R}$, R is not a random variable: it depends solely on the probability distribution. Moreover, for any fixed h, $\mathbb{E}[\hat{R}(h)] = R(h)$. Denote by $\overline{h}$ a best possible classifier *in* $\mathcal{H}$, that is, the one that *generalizes* best:

$$R(\overline{h}) = \inf_{h \in \mathcal{H}} R(h).$$

The parameter $\overline{h}$ depends only on the class $\mathcal{H}$ and the probability distribution. A less ambitious goal is to bound the difference

$$R(\hat{h}) - R(\overline{h}). \tag{B}$$

We emphasize here that $\hat{R}(h)$ and $R(\hat{h})$ are both random variables, and $\hat{R}(\hat{h})$ is a random variable in two ways. Bounds on (A) and (B) are therefore *probabilistic*. More precisely, for any given *tolerance* $\delta \in (0, 1)$, we want to find constants $C(n, \delta)$ and $C'(n, \delta)$ such that

$$R(\hat{h}) - R(h^*) \leq C(n, \delta) \quad \text{and} \quad R(\hat{h}) - R(\overline{h}) \leq C'(n, \delta)$$

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 constants to decrease to 0 as $n \to \infty$. In this lecture we will derive bounds on (B) in the case where $\mathcal{H}$ is a finite set.

Risk bounds for finite sets of classifiers

In this section we prove the following bound.

Theorem 4.1. *Let $\mathcal{H} = \{h_1, \dots, h_K\}$ be a finite dictionary. Then for $\delta \in (0,1)$,*

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

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

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

As a first step towards bounding the supremum, we need to bound the difference $|R(h) - \hat{R}(h)|$ of an individual, fixed h. The key ingredient for such a bound is a concentration of measure inequality known as Hoeffding's bound.

Theorem 4.2 (Hoeffding's Inequality)**.** *Let $Z_1, \dots, Z_n$ 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 4.3. *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

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

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 4.1. The goal is to bound the supremum

$$\sup_{h \in \mathcal{H}} |\hat{R}(h) - R(h)|. \tag{4.2}$$

For this, we use the union bound. Indeed, for each h_i we can apply Lemma 4.3 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 (4.2) being bounded by t can be expressed equivalently as

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

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:

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

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

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

and using (4.1) the claim follows. □

One drawback of the bound in Theorem 4.1 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. We will see that using 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 situation in which the "variance" is not too big.

Notes

5

Probably Approximately Correct

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

- The *empirical risk* $\hat{R}$ and the classifier $\hat{h}$ 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

$$R(\hat{h}) - \inf_{h \in \mathcal{H}} R(h) \leq \sqrt{\frac{2\log(K) + 2\log(2/\delta)}{n}}. \tag{5.1}$$

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 (5.1) 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:

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

Assuming $h^* \in \mathcal{H}$ and $Y = f(X)$, we have $R(h^*) = 0$, and h^* would be the *correct* classifier. The classifier $\hat{h}$ 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 $\text{size}(\mathcal{H})$ 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) $\text{size}(\mathcal{H}) = \log(K)$ would be adequate here. Similarly, we denote by $\text{size}(\mathcal{X})$ 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 $\text{size}(\mathcal{X})$ or $\text{size}(\mathcal{H})$ is not the same as the cardinality of these sets!

Definition 5.1. (PAC Learning [1]) A hypothesis class $\mathcal{H}$ is called **PAC-learnable** if there exists a classifier $\hat{h} \in \mathcal{H}$ 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 $\mathcal{X} \times \mathcal{Y}$,

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

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 $\hat{h}$ from the data runs in time polynomial in $1/\epsilon$, $1/\delta$, $\text{size}(\mathcal{X})$ and $\text{size}(\mathcal{H})$.

Remark 5.2. 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.

Generalization bounds and noise

We conclude by commenting briefly on an improvement of the generalization bound (5.1) when incorporating assumptions on the distribution. While the bound (5.1) incorporates the number of samples and the size of $\mathcal{H}$, it does not take into account properties of the distribution, for example, the uncertainty $\epsilon = Y - f(X)$, where $f(X) = \mathbb{E}[Y|X]$ is the regression function. 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 5.3. *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 $\delta \in (0, 1)$,*

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

[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*.

In the PAC learning context, we see that

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

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. The proof of this result relies on a concentration of measure result similar to Hoeffding's inequality, called Bernstein's inequality.

Theorem 5.4 (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 $\sigma^2 = \frac{1}{n}\sum_{i=1}^n \operatorname{Var}(Z_i)$. Then for $t > 0$,*

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

We outline the idea of the proof. 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

$$\begin{aligned}R(\hat{h}) - R(h^*) &\leq \frac{1}{n}\sum_{i=1}^n \underbrace{(Z_i(\hat{h}) - \mathbb{E}[Z_i(\hat{h})])}_{\overline{Z}_i(\hat{h})}\\ &\leq \max_{h\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^n \overline{Z}_i(h).\end{aligned} \tag{5.2}$$

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

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

We can now apply Bernstein's inequality to the probability that the sum (5.2) 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 $\sigma^2(\hat{h})$. 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

6

Learning Shapes

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. We then move on to the more general framework of Rademacher complexity.

Learning Rectangles

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).

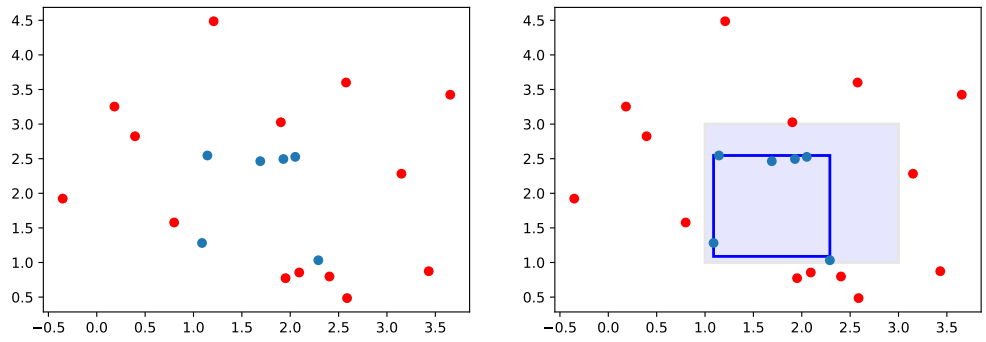


Figure 6.1: The blue points are in the true rectangle, while the red points are outside of it. The smaller blue rectangle represents the computed classifier $\hat{h}$, while the shaded are corresponds to the true rectangle that generated the original labelling.

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

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

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 $\epsilon/4$.

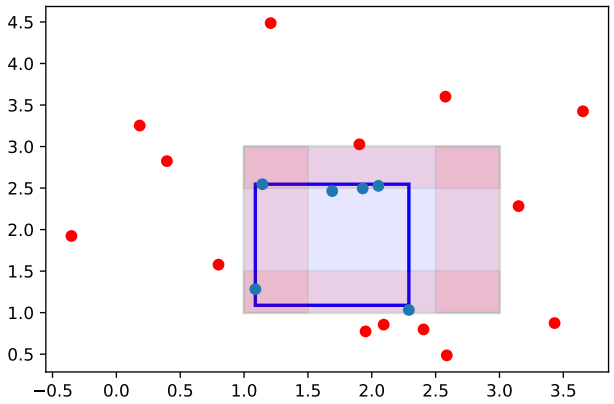


Figure 6.2: Four boundary regions with probability mass $\epsilon/4$ each.

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:

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

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 $1 - \delta$.

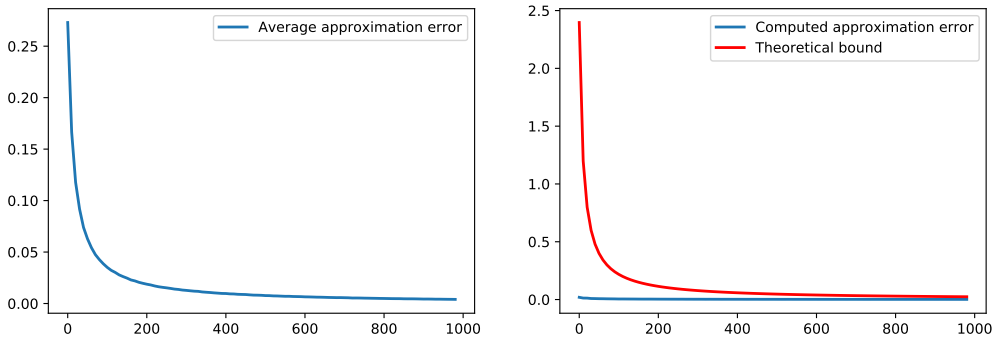


Figure 6.3: The left graphs shows the *average* risk as n increases. The right right graph shows the risk bound ϵ when given a confidence $1 - \delta = 0.99$ (blue curve), and the theoretical generalization bound as derived in the example.

Remark 6.1. 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

7

Rademacher Complexity

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

$$\sup_{h\in\mathcal{H}} |\hat{R}(h) - R(h)| = \sup_{h\in\mathcal{H}} \left| \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{h(X_i) \neq Y_i\} - \mathbb{E}[\mathbf{1}\{h(X_i) \neq Y_i\}] \right|. \tag{A}$$

For finite $\mathcal{H}$, the probability that this quantity exceeds a given t can be bounded using the union bound, but this approach does not work for infinite sets. We therefore derive a different method that is based on intrinsic complexity measures of $\mathcal{H}$. The first such measure that we will encounter is the Rademacher complexity.

Rademacher Complexity

In the following, we write $Z_i = (X_i, Y_i)$, $i \in \{1, \dots, n\}$, for the set of (random) pairs of samples and labels in $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, with points in $\mathcal{Z}$ denoted by $z = (x, y)$. To every $h \in \mathcal{H}$ we associate a function $g\colon \mathcal{Z} \to \{0, 1\}$ by setting

$$g(z) = \mathbf{1}\{h(x) \neq y\},$$

and we denote the class of these functions by $\mathcal{G}$.[1] Using this notation, we get

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

We will bound this expression in terms of a of a property of the set $\mathcal{G}$, the **Rademacher complexity**. For what follows, we say that a random variable has the **Rademacher distribution** if it takes the values 1 or -1 with probability $1/2$ each. When an expression potentially depends on different random quantities, for example random variables X and Y, we write $\mathbb{E}_X$ to denote the expectation with respect to only X.

Definition 7.1. (Rademacher Complexity) Let $\mathcal{G}$ be a class of real-valued functions on $\mathcal{Z}$. Let $\boldsymbol{\sigma} = (\sigma_1, \dots, \sigma_n)$ be a random vector, such that the σ_i are independent Rademacher random variables, and let $\boldsymbol{z} \in \mathcal{Z}$. The **empirical Rademacher complexity** of the family of functions $\mathcal{G}$ with respect to $\boldsymbol{z}$ is defined as

$$\hat{\mathcal{R}}_{\boldsymbol{z}}(\mathcal{G}) = \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{g\in\mathcal{G}} \frac{1}{n} \sum_{i=1}^{n} \sigma_i \cdot g(z_i) \right]. \tag{7.1}$$

[1] We could, and will eventually, use any other loss function.

The **Rademacher complexity** is the expectation:

$$\mathcal{R}_n(\mathcal{G}) = \mathbb{E}_{\boldsymbol{Z}}[\hat{\mathcal{R}}_{\boldsymbol{Z}}(\mathcal{G})]. \tag{7.2}$$

Remark 7.2. Note that the expected value in (7.1) is only over the random signs, and that the point $\boldsymbol{z}$ is fixed. It is only in (7.2) that we replace the point by a random vector $\boldsymbol{Z} = (Z_1, \ldots, Z_n)$ and take the expectation. As the distribution of $\boldsymbol{\sigma}$ is discrete, we could also rewrite the empirical Rademacher complexity as average:

$$\hat{\mathcal{R}}_{\boldsymbol{z}}(\mathcal{G}) = \frac{1}{2^n} \sum_{\boldsymbol{\sigma} \in \{-1,1\}^n} \left(\sup_{g \in \mathcal{G}} \frac{1}{n} \sum_{i=1}^{n} \sigma_i \cdot g(z_i) \right).$$

Note also that the definition works for *any* set of real-valued functions g, not only those that arise from the loss function applied to a classifier. In the literature there are various variations on the notion of Rademacher complexity. It can be defined simply for sets: Given a set $S \subset \mathbb{R}^n$, the Rademacher complexity of S is defined as

$$\mathcal{R}(S) = \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{\boldsymbol{x} \in S} \frac{1}{n} \sum_{i=1}^{n} \sigma_i x_i \right]. \tag{7.3}$$

Our notion is a special case: when $S = \{(g(z_1), \ldots, g(z_n)) \colon g \in \mathcal{G}\}$, then we have $\mathcal{R}_{\boldsymbol{z}}(\mathcal{G}) = \mathcal{R}(S)$.

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 $1 - \delta$,*

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

Before going into the proof, we list some examples.

Example 7.4. Assume $\mathcal{H} = \{h\}$ consists of only one function. Then for any point $(z_1, \ldots, z_n) \in \mathcal{Z}^n$, 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 i, $g(z_i)$ and $-g(z_i)$ appear an equal number of times when averaging over all possible sign vectors.

Example 7.5. Let $\mathcal{H}$ be the set of *all* binary classifiers. It follows that for any $(z_1, \ldots, z_n) \in \mathcal{Z}^n$, the set of vectors $(g(z_1), \ldots, g(z_n))$ 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 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 $\hat{\mathcal{R}}_{\boldsymbol{z}}(\mathcal{G}) = 1/2$.

Example 7.6. We will see that for a finite set $\mathcal{H} = \{h_1, \ldots, h_K\}$ and $\boldsymbol{z} \in \mathcal{Z}^n$, we get 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*.

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 $\{Z_i\}$ be a set of independent random variables defined on a space $\mathcal{Z}$, let $\{c_i\} \subset \mathbb{R}$ be constants with $c_i > 0$, and let $f\colon \mathcal{Z} \to \mathbb{R}$ be a function such that for all $i \in \{1,\dots,n\}$ and $z_i \neq z_i'$,*

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

Then for all $t > 0$, the following inequality holds:

$$\begin{aligned} \mathbb{P}\left(f(Z_1,\dots,Z_n) > \mathbb{E}[f(Z_1,\dots,Z_n)] + t\right) &\leq \mathrm{e}^{-\frac{2t^2}{\sum_{i=1}^n c_i^2}} \\ \mathbb{P}\left(f(Z_1,\dots,Z_n) < \mathbb{E}[f(Z_1,\dots,Z_n)] - t\right) &\leq \mathrm{e}^{-\frac{2t^2}{\sum_{i=1}^n c_i^2}} \end{aligned} \tag{7.5}$$

Using the union bound we can combine the two inequalities (7.5) to one inequality for the absolute value $|f(Z_1,\dots,Z_n) - \mathbb{E}[f(Z_1,\dots,Z_n)]|$, with an additional factor of 2 in front of the exponential bound. Note that McDiarmid's inequality contains Hoeffding's inequality as a special case when f is the average.

Proof of Theorem 7.3. Define the function

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

Then for $i \in \{1,\dots,n\}$ and $z_i \neq z_i'$, and using the fact that the difference of suprema is not bigger than the supremum of a difference, we get

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

from which we conclude that

$$|\Phi(z_1,\dots,z_i,\dots,z_n) - \Phi(z_1,\dots,z_i',\dots,z_n)| \leq \frac{1}{n}.$$

The function Φ thus satisfies the conditions of McDiarmid's inequality with $c_i = 1/n$, and from this inequality we get

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

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

$$\Phi(Z_1,\dots,Z_n) \leq \mathbb{E}[\Phi(Z_1,\dots,Z_n)] + \sqrt{\frac{\log(1/\delta)}{2n}}.$$

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

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

where for the inequality we used the fact that the sup of an expectation is not more than the expectation of the sup. We next use an idea known as *symmetrization*. The key observation is that each summand $g(Z_i') - g(Z_i)$ is just as likely to be positive as it is to be negative. In other words, if we replace $g(Z_i') - g(Z_i)$ with its negative, $g(Z_i) - g(Z_i')$, then the above expectation does not change. 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:

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

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

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

[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 $\boldsymbol{\sigma}$. □

The empirical Rademacher complexity $\hat{\mathcal{R}}_{\boldsymbol{z}}$ is a function of $(z_1, \ldots, z_n)$, 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

$$|\hat{\mathcal{R}}_{\boldsymbol{z}'}(\mathcal{G}) - \hat{\mathcal{R}}_{\boldsymbol{z}}(\mathcal{G})| \leq \frac{1}{n}.$$

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

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

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, note that the inequalities (7.4) and (7.6) are *one-sided* inequalities: they bound a difference but not the absolute value of this difference. These can easily be adapted to give a bound on the supremum of the absolute difference $|\hat{R}(h) - R(h)|$. As a consequence of Example 7.5 we see that when considering the set of *all* binary classifiers, we do not get a generalization bound that converges to 0 as $n \to \infty$ using the Rademacher complexity bound.

Notes

8

VC Theory

As usual we operate on a pair of input-output spaces $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ with $\mathcal{Y} = \{-1, 1\}$. Let $\mathcal{H}$ be a set of classifiers with associated set

$$\mathcal{G} = \{g \colon \mathcal{Z} \to \{0, 1\} \colon g(z) = \mathbf{1}\{h(x) \neq y\}, h \in \mathcal{H}\}.$$

We saw that we could bound the maximum difference between the generalization risk $R(h)$ and the empirical risk $\hat{R}(h)$ of a classifier using the Rademacher complexity $\mathcal{R}_n(h)$:

$$\sup_{h \in \mathcal{H}} (R(h) - \hat{R}(h)) \leq 2\mathcal{R}_n(\mathcal{G}) + \sqrt{\frac{\log(1/\delta)}{2n}}. \tag{8.1}$$

Instead of considering the set $\mathcal{G}$, we can also consider the Rademacher complexity of $\mathcal{H}$ itself. For what follows, assume that the classifiers in $\mathcal{H}$ take the values $\{-1, 1\}$.

Lemma 8.1. *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 depends on writing $\mathbf{1}\{h(x) \neq y\} = (1 - yh(x))/2$, and is left as an exercise. 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. The theory is named after Vladimir Vapnik and Alexey Chervonenkis, who developed it in the 1960s.

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 8.2. 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 8.3. 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 ($\mathbb{P}(\sigma_i = +1) = \mathbb{P}(\sigma_i = -1) = 1/2$) and $\boldsymbol{x} = (x_1, \ldots, x_n)$.

Lemma 8.4. *(Massart's Lemma) If $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_K\} \subset \mathbb{R}^n$ consists of K elements, 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 8.5. *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

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

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) we immediately arrive at the following bound.

Corollary 8.6. *For all $h \in \mathcal{H}$,*

$$R(h) \leq \hat{R}(h) + \sqrt{\frac{2 \log(\Pi_{\mathcal{H}}(n))}{n}} + \sqrt{\frac{\log(1/\delta)}{2n}} \tag{8.2}$$

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

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

that is, if all dichotomies are possible.

Example 8.7. 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 8.8. If $\mathcal{H} = \{h_1, h_2\}$ consists of only two classifiers, one of which is constant 1 and the other is constant 0, 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 8.9. 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

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

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.

Notes

9

The VC Inequality

The VC dimension of a hypothesis set $\mathcal{H}$ is the maximal cardinality of a subset of the input space $\mathcal{X}$ that is *shattered* by $\mathcal{H}$:

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

where the *growth function* $\Pi_{\mathcal{H}}(n)$ counts the number of possible dichotomies,

$$\Pi_{\mathcal{H}}(n) = \max_{\boldsymbol{x} \in \mathcal{X}^n} \left|\{(h(x_1), \ldots, h(x_n)) \colon h \in \mathcal{H}\}\right|.$$

The VC dimension is a combinatorial quantity that depends only on $\mathcal{H}$. It acts as a surrogate for cardinality when dealing with infinite sets.

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

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

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

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

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 9.2. ***(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 $1 - \delta$,*

$$\sup_{h\in\mathcal{H}} R(h) - \hat{R}(h) \le \sqrt{\frac{2d\log\left(\frac{\mathrm{e}n}{d}\right)}{n}} + \sqrt{\frac{\log(1/\delta)}{2n}}.$$

Proof. In Corollary 8.6, Lecture 8, we saw that

$$\sup_{h\in\mathcal{H}} R(h) - \hat{R}(h) \le \sqrt{\frac{2\log\left(\Pi_{\mathcal{H}}(n)\right)}{n}} + \sqrt{\frac{\log(1/\delta)}{2n}}.$$

The claim now follows directly from Lemma 9.1. □

We remark that the bounds here were all stated as *one-sided* bounds: that is, they are stated as bounds on the difference $R(h) - \hat{R}(h)$ and not the absolute value of the difference. We can get two-sided bounds by making adjustments in the derivation of bounds using the Rademacher complexity (using the second case of McDiarmid's inequality) and arrive at the following bound, which holds with probability at least $1 - \delta$:

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

Note that the only difference is the factor of $2/\delta$, which is a consequence of combining two one-sided inequalities to one two-sided inequality using the union bound.

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:

[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.

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 $\triangle$). For $n = 4$ there are 16 possible labellings, and if we arrange the points in diamond form $\diamond$, 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, $i \in \{1, \ldots, 4\}$, 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.

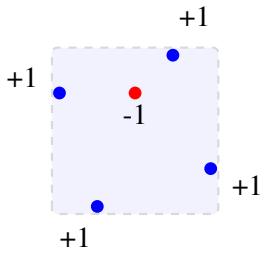


Figure 9.1: A dichotomy that is not captured by rectangles.

Notes

10

General Loss Functions

So far we looked at the problem of binary classification. We considered a set $\mathcal{H}$ of classifiers $h\colon \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y}$ was a set with two elements ($\{-1, 1\}$ or $\{0, 1\}$, for example). Given a distribution on $\mathcal{X} \times \mathcal{Y}$, we considered the *generalization risk*,

$$R(h) = \mathbb{E}[\mathbf{1}\{h(X) \neq Y\}] = \mathbb{P}(h(X) \neq Y), \tag{10.1}$$

and the *empirical risk*,

$$\hat{R}(h) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{h(X_i) \neq Y_i\}, \tag{10.2}$$

which is based on a sequence of random observations (X_i, Y_i), $i \in \{1, \dots, n\}$. For each set of realizations $\{(x_i, y_i)\}$ one can (in principle) construct a classifier $\hat{h} \in \mathcal{H}$ that minimizes the empirical risk (10.2). We are ultimately interested in the generalization risk $R(\hat{h})$ and not the empirical risk $\hat{R}(\hat{h})$ (this would just tell us something about how well our classifier works on the training data). Specifically, we are interested in how close the risk $R(\hat{h})$ is to the *optimal* generalization risk $R(\hat{h}) = \inf_{h \in \mathcal{H}} R(h)$. To analyse this, we split the difference into two parts:

$$\begin{aligned} R(\hat{h}) - \inf_{h \in \mathcal{H}} R(h) &\leq R(\hat{h}) - \hat{R}(\hat{h}) + \hat{R}(\overline{h}) - \inf_{h \in \mathcal{H}} R(h) \\ &\leq 2 \sup_{h \in \mathcal{H}} \left| \hat{R}(h) - R(h) \right|. \end{aligned}$$

The term within the sup to be bounded is just the difference between the average of i.i.d. random variables and their expectation. To bound this difference we used:

- Hoeffding's or Bernstein's inequality and the union bound for finite $\mathcal{H}$, to obtain a bound in terms of $\log(|\mathcal{H}|)$;
- McDiarmid's inequality applied directly to the supremum and symmetrization to obtain a bound in terms of the Rademacher complexity.

The Rademacher complexity could then be bounded, via Massart's inequality, in terms of the growth function of $\mathcal{H}$, which in turn could be bounded again, via the Sauer-Shelah Lemma, in terms of the VC dimension of $\mathcal{H}$. It is natural to ask how much of this depends on the fact that we used binary classification and the unit loss $\mathbf{1}\{h(X) \neq Y\}$, and to what extent these bounds generalize to other types of classifiers and loss functions.

General Loss Functions

Consider a set of function $\mathcal{H} = \{h\colon \mathcal{X} \to \mathcal{Y}\}$ with $\mathcal{Y} = [-1, 1]$ (or any other subset of $\mathbb{R}$), and a loss function $L\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$. Besides the unit-loss that we studied so far, examples include:

- $L(x, y) = (x - y)^2$ (quadratic loss);
- $L(x, y) = |x - y|$ (ℓ_1-loss);
- $L(x, y) = |x - y|^p$ (ℓ_p-loss);
- $L(x, y) = \log(1 + e^{-xy})$ (log-loss).

Sometimes the loss function is scaled to ensure that it is in a certain range. For example, for the quadratic loss, $(h(x) - y)^2/2 \in [0, 1]$ if $h(x) \in [-1, 1]$ and $y \in [-1, 1]$. The *generalization risk* is defined as

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

while the *empirical risk* is

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

Just as in the binary case with unit loss, we denote by $\overline{h}$ a classifier with optimal generalization risk in $\mathcal{H}$, and by $\hat{h}$ a minimizer of $\hat{R}(h)$. Also as in the binary case, we can bound

$$R(\hat{h}) - R(\overline{h}) \leq 2 \sup_{h \in \mathcal{H}} \left| \hat{R}(h) - R(h) \right|.$$

Consider the set of functions

$$L \circ \mathcal{H} := \{g\colon \mathcal{Z} \to [0, 1]\colon g(z) = L(h(x), y)\}.$$

This plays the role of the set $\mathcal{G}$ defined in Lecture 6. For $\boldsymbol{z} = (z_1, \ldots, z_n)$, with $z_i = (x_i, y_i)$, we can define the *empirical Rademacher complexity* as

$$\hat{\mathcal{R}}_{\boldsymbol{z}}(L \circ \mathcal{H}) = \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{g \in L \circ \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \sigma_i g(z_i) \right],$$

and the *Rademacher complexity* as

$$\mathcal{R}_n(L \circ \mathcal{H}) = \mathbb{E}_{\boldsymbol{Z}} \left[\hat{\mathcal{R}}_{\boldsymbol{Z}}(L \circ \mathcal{H} \right].$$

Assuming that $L(x, y) \in [0, 1]$ (which we can by scaling the loss function accordingly) one can use the McDiarmid's bounded difference inequality and the same symmetrisation argument as in the binary case to derive the bound

$$\sup_{h \in \mathcal{H}} \left| \hat{R}(h) - R(h) \right| \leq 2\mathcal{R}_n(L \circ \mathcal{H}) + \sqrt{\frac{\log(2/\delta)}{2n}},$$

which holds with probability at least $1 - \delta$. For finite sets of classifiers $\mathcal{H} = \{h_1, \ldots, h_K\}$ we get the same cardinality bounds as in the binary case.

Theorem 10.1. *Let $\mathcal{H} = \{h_1, \dots, h_K\}$ be a set of classifiers $h\colon \mathcal{X} \to [-1, 1]$, and let L be a loss function taking values in $[0, 1]$. Then*

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

Proof. Fix $\boldsymbol{z} = (z_1, \dots, z_n)$. Then by Massart's Lemma (Lemma 8.4 in Lecture 8), the empirical Rademacher complexity is bounded by

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

where

$$r = \sup\{\|\boldsymbol{x}\|\colon \boldsymbol{x} \in S\}, \quad S = \{g(z_1), \dots, g(z_n))\colon g \in L \circ \mathcal{H}\}.$$

Since by assumption $g(z) = L(h(x), y) \in [0, 1]$, $r \leq \sqrt{n}$ and the result follows by taking the expectation over $\boldsymbol{Z}$. □

For infinite $\mathcal{H}$ we cannot repeat the arguments used in the case of binary classifiers with unit loss. The bounds based on growth function and VC dimension depend on the fact that the number of possible dichotomies is finite and this is no longer the case when considering functions h with infinite range. One way of dealing with such a situation is to *approximate* an infinite set by a finite set in such a way, that every element of the infinite set is *close* to a point in the finite subset.

Notes

11

Covering Numbers

In this lecture we consider hypothesis set $\mathcal{H} = \{h\colon \mathcal{X} \to \mathcal{Y} = [-1, 1]\}$ and a loss function $L\colon \mathcal{Y} \times \mathcal{Y} \to [0, 1]$. Define the set of functions

$$L \circ \mathcal{H} := \{g\colon \mathcal{Z} \to [0, 1]\colon g(z) = L(h(x), y)\}.$$

This set plays the role of the set $\mathcal{G}$ defined in Lecture 6. For $\boldsymbol{z} = (z_1, \ldots, z_n)$, with $z_i = (x_i, y_i)$, we can define the *empirical Rademacher complexity* as

$$\hat{\mathcal{R}}_{\boldsymbol{z}}(L \circ \mathcal{H}) = \mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{g \in L\circ\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \sigma_i g(z_i)\right],$$

where the expectation is over all sign vector $\sigma = (\sigma_1, \ldots, \sigma_n)$ with independent σ_i that satisfy $\mathbb{P}\{\sigma_i = +1\} = \mathbb{P}\{\sigma_i = -1\} = 1/2$. The *Rademacher complexity* is defined as as

$$\mathcal{R}_n(L \circ \mathcal{H}) = \mathbb{E}_{\boldsymbol{Z}}\left[\hat{\mathcal{R}}_{\boldsymbol{Z}}(L \circ \mathcal{H})\right],$$

where the expectation is over all n-tuples $\boldsymbol{Z} = (Z_1, \ldots, Z_n)$, where $Z_i = (X_i, Y_i)$. As in the case of binary classification with unit loss, one can use McDiarmid's bounded difference inequality and a symmetrisation argument to derive the bound

$$\sup_{h\in\mathcal{H}} \left|\hat{R}(h) - R(h)\right| \leq 2\mathcal{R}_n(L \circ \mathcal{H}) + \sqrt{\frac{\log(2/\delta)}{2n}},$$

which holds with probability at least $1 - \delta$. For finite $\mathcal{H} = \{h_1, \ldots, h_K\}$, the arguments used in the case of binary classification carry over seamlessly.

Theorem 11.1. *Let $\mathcal{H} = \{h_1, \ldots, h_K\}$ be a set of functions $h\colon \mathcal{X} \to [-1, 1]$, and let L be a loss function taking values in $[0, 1]$. Then*

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

Proof. Fix $\boldsymbol{z} = (z_1, \ldots, z_n)$. Then by Massart's Lemma (Lemma 8.4 in Lecture 8), the empirical Rademacher complexity is bounded by

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

where

$$r = \sup\{\|\boldsymbol{x}\| \colon \boldsymbol{x} \in S\}, \quad S = \{(g(z_1), \dots, g(z_n)) \colon g \in L \circ \mathcal{H}\}.$$

Since by assumption $g(z) = L(h(x), y) \in [0, 1]$, $r \le \sqrt{n}$ and the result follows by taking the expectation over $\boldsymbol{Z}$. □

For infinite $\mathcal{H}$, we cannot repeat the arguments that lead to the VC inequality in the binary classification case, since these arguments were based on the fact that even for infinite $\mathcal{H}$, the number of possible dichotomies is finite. This limitation can be circumvented by *approximating* $L \circ \mathcal{H}$ by a finite subset that is sufficiently dense. This leads to the concept of **covering numbers**.

Covering Numbers

Recall that a *metric* on a set S is a function $\mathrm{d} \colon S \times S \to \mathbb{R}_{\ge 0}$ such that $\mathrm{d}(x, y) = 0$ if and only if $x = y$, $\mathrm{d}(x, y) = \mathrm{d}(y, x)$, and $\mathrm{d}(x, y) + \mathrm{d}(y, z) \le \mathrm{d}(x, z)$. A *pseudo-metric* is defined like a metric, but replacing the first condition with the looser requirement that $\mathrm{d}(x, x) = 0$ (that is, $\mathrm{d}(x, y) = 0$ may also be possible if $x \ne y$).

Definition 11.2. Given a pseudo-metric space (S, d), an ϵ-net is a subset $T \subset S$ such that for every $x \in S$ there exists $y \in T$ with $\mathrm{d}(x, y) \le \epsilon$. The *covering number* corresponding to S and ϵ is the smallest cardinality of an ϵ-net:

$$N(S, \mathrm{d}, \epsilon) = \inf\{|T| \colon T \text{ is an } \epsilon\text{-net}\}.$$

Example 11.3. Consider the Euclidean unit ball $B^d = B_2^d = \{x \in \mathbb{R}^d \colon \|x\|_2 \le 1\}$. We construct an ϵ-net T as follows. Start with an arbitrary point $T = \{x_1\}$ and add points to T as follows: if $T = \{x_1, \dots, x_k\}$, then choose a point x_{k+1} such that $\|x_{k+1} - x_j\| > \epsilon$ for all j if possible, and add it to T, and if this is not possible, then stop. This process terminates since B_2^d is bounded. The resulting set $T = \{x_1, \dots, x_N\}$ is an ϵ-net by construction, and the distance between any two points in this set it larger than ϵ. Hence, the balls $B^d(x_j, \epsilon/2)$ of radius $\epsilon/2$ around the points in T are disjoint. Since the union of these balls is contained in the larger ball $(1 + \epsilon/2)B^d$ (the scaling of the unit ball by a factor of $(1 + \epsilon/2)$), we get the volume inequality

$$N \operatorname{vol}(B^d(x_1, \epsilon/2)) = \operatorname{vol}\left(\bigcup_i B^d(x_i, \epsilon/2)\right) \le \operatorname{vol}\left((1 + \epsilon/2)B^d\right). \tag{11.1}$$

The volume of a ball of radius r is $r^d \cdot \operatorname{vol}(B^d)$, hence

$$\operatorname{vol}(B^d(x_1, \epsilon/2)) = (\epsilon/2)^d \operatorname{vol}(B^d) \text{ and } \operatorname{vol}((1 + \epsilon/2)B^d) = (1 + \epsilon/2)^d \operatorname{vol}(B^d),$$

and we get from (11.1) that

$$N \le \frac{(1 + \epsilon/2)^d}{(\epsilon/2)^d} = \left(\frac{2}{\epsilon} + 1\right)^d \le \left(\frac{3}{\epsilon}\right)^d$$

if $\epsilon < 1$. Of course, if $\epsilon \ge 1$ then we have an ϵ-net of size 1.

We next consider the set $L \circ \mathcal{H}$ with the empirical ℓ_1 distance

$$\mathrm{d}_1^{\boldsymbol{z}}(g_1, g_2) = \frac{1}{n} \sum_{i=1}^n |g_1(z_i) - g_2(z_i)|.$$

We get the following bound for the empirical Rademacher complexity at $\boldsymbol{z}$.

Theorem 11.4. *Let $\mathcal{H}$ be a hypothesis set of functions taking values in $[-1,1]$, and let L be a loss function taking values in $[0,1]$. Then for any $\boldsymbol{z} \in \mathcal{Z}^n$ we have*

$$\hat{\mathcal{R}}_{\boldsymbol{z}}(L \circ \mathcal{H}) \leq \inf_{\epsilon > 0} \left\{ \epsilon + \sqrt{\frac{2 \log(N(L \circ \mathcal{H}, \mathrm{d}_1^{\boldsymbol{z}}, \epsilon))}{n}} \right\}.$$

The covering number takes over the role of VC dimension. Note that the covering number increases as ϵ decreases, and we get a trade-off between small ϵ and small covering number.

Proof of Theorem 11.4. Fix $\boldsymbol{z} \in \mathcal{Z}^n$ and $\epsilon > 0$, and let T be a smallest ϵ-net for $(L \circ \mathcal{H}, \mathrm{d}_1^{\boldsymbol{z}})$. It follows that for any $g \in L \circ \mathcal{H}$ we can find a $g' \in T$ such that $\mathrm{d}_1^{\boldsymbol{z}}(g, g') \leq \epsilon$. Hence,

$$\begin{aligned}
\hat{\mathcal{R}}_{\boldsymbol{z}}(L \circ \mathcal{H}) &= \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{g \in L \circ \mathcal{H}} \frac{1}{n} \sum_{i=1}^n \sigma_i g(z_i) \right] \\
&\leq \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{g \in L \circ \mathcal{H}} \frac{1}{n} \sum_{i=1}^n \sigma_i g(z_i) - \sigma_i g'(z_i) \right] + \mathbb{E}_{\boldsymbol{\sigma}} \left[\frac{1}{n} \sum_{i=1}^n \sigma_i g'(z_i) \right] \\
&\leq \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{g \in L \circ \mathcal{H}} \frac{1}{n} \sum_{i=1}^n |g(z_i) - g'(z_i)| \right] + \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{g \in L \circ \mathcal{H}} \frac{1}{n} \sum_{i=1}^n \sigma_i g'(z_i) \right] \\
&\leq \sup_{g \in L \circ \mathcal{H}} \mathrm{d}_1^{\boldsymbol{z}}(g, g') + \mathbb{E}_{\boldsymbol{\sigma}} \left[\max_{g' \in T} \frac{1}{n} \sum_{i=1}^n \sigma_i g'(z_i) \right] \\
&\leq \epsilon + \sqrt{\frac{2 \log(N(L \circ \mathcal{H}, \mathrm{d}_1^{\boldsymbol{z}}, \epsilon))}{n}},
\end{aligned}$$

where we used the bound for finite sets, Theorem 11.1. Since $\epsilon > 0$ was arbitrary, this bounds holds for the infimum among $\epsilon > 0$. □

The bound derived is for the set $L \circ \mathcal{H}$. Under a boundedness condition on the loss function, we can bound the Rademacher complexity of this set in terms of that of $\mathcal{H}$.

Theorem 11.5. *If $L \colon \mathcal{Y} \times \mathcal{Y} \to [0,1]$, then*

$$\hat{\mathcal{R}}_{\boldsymbol{z}}(L \circ \mathcal{H}) \leq 2 \hat{\mathcal{R}}_{\boldsymbol{x}}(\mathcal{H}),$$

where $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{z} = (z_1, \ldots, z_n)$ with $z_i = (x_i, y_i)$.

In light of this result, we conclude this section with a bound on $\hat{\mathcal{R}}_{\boldsymbol{z}}(\mathcal{H})$ for a specific class of functions. In what follows we denote by $B_p^d \subset \mathbb{R}^d$ the unit ball with respect to the p-norm. In particular,

$$\begin{aligned}
B_1^d &= \{x \in \mathbb{R}^d \colon \|x\|_1 = \sum_{i=1}^d |x_i| \leq 1\}, \\
B_\infty^d &= \{x \in \mathbb{R}^d \colon \|x\|_\infty = \max_i |x_i| \leq 1\}.
\end{aligned}$$

Notice that the set B_∞^d is a hypercube. We now consider $\mathcal{X} = B_1^d$ and the class of functions

$$\mathcal{H} = \{h \colon \mathcal{X} \to \mathcal{Y} \colon h(x) = \langle a, x \rangle,\ a \in B_\infty^d\}.$$

By the Hölder inequality, the functions h satisfy

$$|h(x)| = |\langle a, x \rangle| \leq \|a\|_\infty \|x\|_1 \leq 1.$$

Two functions $h, g \in \mathcal{H}$ are thus represented by two vectors $a, b \in \mathbb{R}^d$, and for their distance we have

$$\mathrm{d}_1^{\boldsymbol{x}}(h, g) = \frac{1}{n} \sum_{i=1}^{n} |h(x_i) - g(x_i)| = \frac{1}{n} \sum_{i=1}^{n} |\langle a - b, x_i \rangle|.$$

From the Hölder inequality we get $|\langle a - b, x \rangle| \leq \|a - b\|_\infty \|x\|_1$, and since by assumption each $x_i \in B_1^d$, we have $\|x_i\|_1 \leq 1$ and

$$\mathrm{d}_1^{\boldsymbol{x}}(h, g) \leq \|a - b\|_\infty.$$

An ϵ-net therefore corresponds to a set $T = \{b_1, \ldots, b_N\} \subset B_\infty^d$ such that for every $a \in B_\infty^d$ there exists a j such that $\|a - b_j\|_\infty \leq \epsilon$. It follows from Exercise 4.1 that the covering number is bounded by

$$N(\mathcal{H}, \mathrm{d}_1^{\boldsymbol{x}}, \epsilon) \leq \left(\frac{3}{\epsilon}\right)^d.$$

We thus get the bound

$$\hat{\mathcal{R}}_{\boldsymbol{x}}(\mathcal{H}) \leq \inf_{\epsilon > 0} \left\{ \epsilon + \sqrt{\frac{2d \log(3/\epsilon)}{n}} \right\}.$$

If n is sufficiently large, then setting $\epsilon = 3\sqrt{d \log(n)/n} < 1$, we get the bound

$$\hat{\mathcal{R}}_{\boldsymbol{x}}(\mathcal{H}) \leq 4\sqrt{\frac{d \log(n)}{n}}.$$

Note that the resulting bound does not depend on $\boldsymbol{x}$. It is possible to remove the logarithmic factor $\log(n)$ using a more sophisticated technique called *chaining*.

Notes

12

Model Selection

Given an input space $\mathcal{X}$, an output space $\mathcal{Y}$, a class $\mathcal{H}$ of functions $h\colon \mathcal{X} \to \mathcal{Y}$, a loss function $L\colon \mathcal{Y}\times\mathcal{Y} \to \mathbb{R}_{\geq 0}$, and data $S = \{(x_i, y_i)\}_{i=1}^n$, we would like to solve the **Empirical Risk Minimization** (ERM) problem

$$\begin{aligned} &\text{minimize} \hat{R}(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(x_i), y_i) \\ &\text{subject to} \quad h \in \mathcal{H}. \end{aligned} \tag{A}$$

This is an example of a **constrained optimization problem**. The function to be minimized is the **objective function** and a solution $\hat{h}_S$ of (A) is called a **minimizer**. Several problems can arise when trying to solve this minimization problem.

1. The problem (A) may be hard to solve. This can be the case if the class $\mathcal{H}$ is large, the number of samples n is large, or when the objective function is not differentiable or not even continuous.
2. Do we even want to solve (A)? If the class $\mathcal{H}$ is large we may find a minimizer $\hat{h}_S$ that fits the data well but does not *generalize*.

Since a certain generalization error is unavoidable, we can often replace (A) with a surrogate that is computationally easier to handle and provides a solution that is close enough to the one we are looking for. The choice of such an approximation is also informed by the choice of $\mathcal{H}$. We therefore first study the problem of finding a suitable class $\mathcal{H}$, also known as **model section**.

Model Selection

Consider now the set of inputs again as a set of pairs of random variables $S = \{(X_i, Y_i)\}_{i=1}^n \subset \mathcal{X}\times\mathcal{Y}$, so that $\hat{h}_S$ is a random variable. Ideally we would like the generalization risk $R(\hat{h}_S)$ to be close to the Bayes risk R^*, which is the *best possible* generalization risk. Recall the decomposition

$$R(\hat{h}_S) - R^* = \underbrace{R(\hat{h}_S) - \inf_{h\in\mathcal{H}} R(h)}_{\text{Estimation error}} + \underbrace{\inf_{h\in\mathcal{H}} R(h) - R^*}_{\text{Approximation error}} \tag{12.1}$$

of the excess risk. Denote by $\overline{h}$ the minimizer of $R(h)$ in $\mathcal{H}$. In previous lectures we saw that

$$R(\hat{h}_S) - R(\overline{h}) \leq 2\sup_{h\in\mathcal{H}} \left| R(h) - \hat{R}(h) \right|, \tag{12.2}$$

and therefore any bound that holds for the right-hand side with high probability (such as those based on VC dimension or covering numbers) also holds for the left-hand side. If $\mathcal{H}$ is large, then the bound may not be good enough, and in addition the minimizer $\hat{h}_S$ may be hard to compute. If, on the other hand, $\mathcal{H}$ is too small, then the approximation error can be large. One way to address this issue is to consider a nested family of sets $\mathcal{H}_k \subset \mathcal{H}_{k+1}$ of increasing complexity and to choose a k with optimal overall performance. We illustrate this using an example.

Example 12.1. Let $T \subset \mathbb{R}^2$ be a disk and let (X, Y) be a pair of random variables on $\mathbb{R}^2 \times \{0, 1\}$ such that

$$f_T(x) = \mathbb{E}[Y|X = x] = \begin{cases} 1 & x \in T \\ 0 & x \notin T \end{cases}$$

Consider the unit loss function, so that $R(h) = \mathbb{P}\{h(X) \neq Y\}$ for some function $h \colon \mathbb{R}^2 \to \{0, 1\}$. The function f_T is the *Bayes classifier*, as it satisfies $R(f_T) = R^* = 0$. For any hypothesis set $\mathcal{H}$ we can combine (12.1) and the bound (12.2) to get

$$R(\hat{h}_S) \leq \underbrace{2 \sup_{h \in \mathcal{H}} \left| R(h) - \hat{R}(h) \right|}_{\text{Bound on stimation error}} + \underbrace{\inf_{h \in \mathcal{H}} R(h)}_{\text{Approximation error}} .$$

Let $\mathcal{H}_k$ denote the set of indicator functions of regions bounded bys convex polygons with at most k sides (see Figure 25.1). To bound the estimation error, we use the fact that the VC dimension of the class of convex k-gons is $2k + 1$ (see Problem Set 4), and get the bound

$$\sup_{h \in \mathcal{H}_k} |\hat{R}(h) - R(h)| \leq \sqrt{\frac{(4k + 2) \log(n)}{n}} + \sqrt{\frac{\log(2/\delta)}{2n}} \tag{12.3}$$

with probability $1 - \delta$ (we simplified the logarithmic term in the first part of the bound). For the approximation error, we look at the well-known problem of approximating the circle with a regular polygon. The area enclosed by a regular k-gon inscribed in a circle of radius r is $r^2(k/2) \sin(2\pi/k)$, so the area of the complement in the disk is

$$\pi r^2 - r^2(k/2) \sin(2\pi/k) = O(k^{-2}), \tag{12.4}$$

where the equality follows from the Taylor expansion of the sine function. If the underlying probability distribution on $\mathbb{R}^2$ is the uniform distribution on a larger set or can be approximated as such, then (12.4) gives an upper bound for the approximation error (it can be less), and combined with (12.3) illustrates the estimation-approximation trade-off. The larger the number of sides of the polygons, the smaller the approximation error becomes, but the estimation error can become large due to *overfitting*. Thus even if the unknown shape we want to learn is a circle, if the number of samples is small we may be better off restricting to simpler models! This also has the additional advantage of saving computational cost.

There are general strategies for optimizing for k. One theoretical method is known as **structural risk minimization** (SRM). In this model, the parameter k enters into the optimization problem to be solved. An alternative, practical approach is **cross-validation**. In this approach, the training set S is subdivided into a smaller training set and a *validation set*. In a nutshell, the ERM problem is solved for different parameters k on the training set, and the one that performs best on the validation set is chosen.

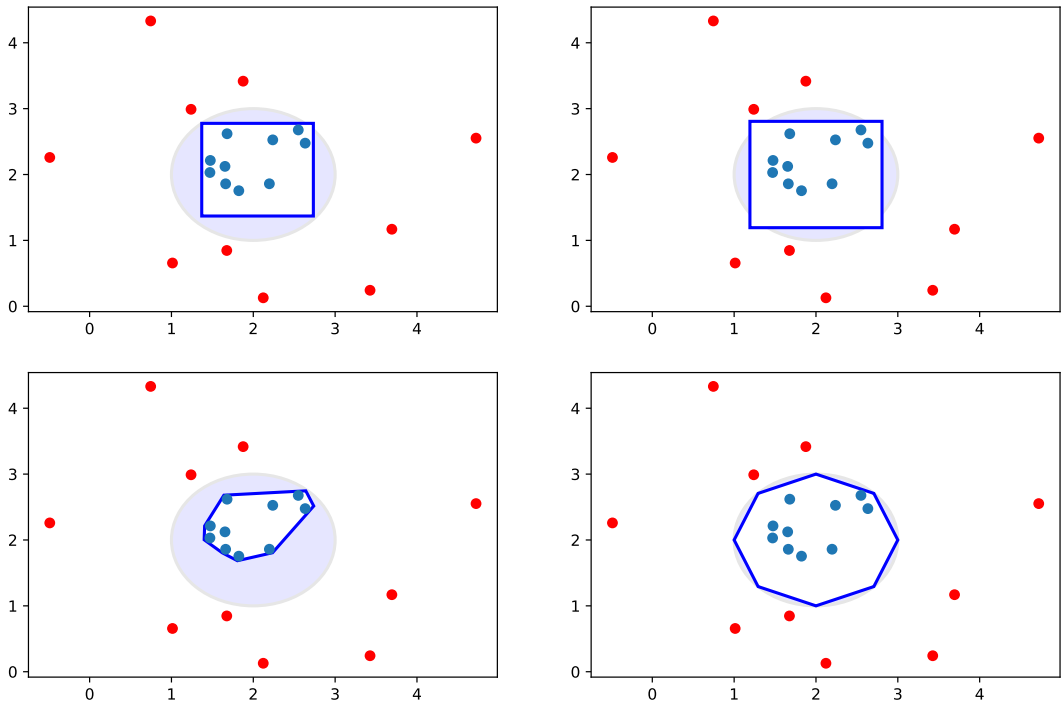


Figure 12.1: Learning a circle with polygons. The left panel shows the *estimation error* when trying to learn the shape using polygons with at most 4 and with at most 8 sides from the data. This is the error typically incurred by empirical risk minimization. The right panel illustrates the *approximation error*. This error measures how good we can approximate the ground truth with our function class.

Part II

Optimization

13

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

$$\begin{aligned} &\text{minimize} f(\boldsymbol{x}) \\ &\text{subject to} \quad \boldsymbol{x} \in \Omega \end{aligned} \tag{13.1}$$

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**. If $\Omega = \mathbb{R}^d$, then the problem is an **unconstrained optimization problem**. Among all $\boldsymbol{x} \in \Omega$, we seek one with smallest f-value. Typically, the constraint set Ω will consist of such $\boldsymbol{x} \in \mathbb{R}^d$ that satisfy certain equations and inequalities,

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

A vector $\boldsymbol{x}^*$ satisfying the constraints is called an **optimum**, a **solution**, or a **minimizer** of the problem (13.1), if $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all other $\boldsymbol{x}$ 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

$$\text{minimize } f(\boldsymbol{x}), \tag{13.2}$$

where $\boldsymbol{x} \in \mathbb{R}^d$.

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

Definition 13.1. A point $\boldsymbol{x}^* \in \mathbb{R}^d$ is a

- *global minimizer* of (13.2) if for all $\boldsymbol{x} \in \mathbb{R}^d$, $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$;
- a *local minimizer*, if there is an open neighbourhood U of $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in U$;
- a *strict local minimizer*, if there is an open neighbourhood U of $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) < f(\boldsymbol{x})$ for all $\boldsymbol{x} \in U$;
- an *isolated minimizer* if there is an open neighbourhood U of $\boldsymbol{x}^*$ such that $\boldsymbol{x}^*$ is the only local minimizer in U.

Without any further assumptions on f, finding a minimizer is a hopeless task: we simply can not 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 $\nabla f(\boldsymbol{x})$ for the gradient of f at $\boldsymbol{x}$, i.e., the vector

$$\nabla f(\boldsymbol{x}) = \left(\frac{\partial f}{\partial x_1}(\boldsymbol{x}), \ldots, \frac{\partial f}{\partial x_d}(\boldsymbol{x}) \right)^\top$$

Theorem 13.2. *Let $\boldsymbol{x}^*$ be a local minimizer of f and assume that $f \in C^1(U)$ for a neighbourhood of U of $\boldsymbol{x}^*$. Then $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$.*

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 $\boldsymbol{x}$, $\nabla^2 f(\boldsymbol{x})$, is the $d \times d$ symmetric matrix given by the second derivatives,

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

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

Recall also that a matrix $\boldsymbol{A}$ is **positive semidefinite**, written $\boldsymbol{A} \succeq \mathbf{0}$, if for every $\boldsymbol{x} \in \mathbb{R}^d$, $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} \geq 0$, and positive definite, written $\boldsymbol{A} \succ \mathbf{0}$, if $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} > 0$. 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 13.3. *Let $f \in C^2(U)$ for some open set U and $\boldsymbol{x}^* \in U$. If $\boldsymbol{x}^*$ is a local minimizer, then $\nabla f(\boldsymbol{x}^*) = 0$ and $\nabla^2 f(\boldsymbol{x}^*)$ is positive semidefinite. Conversely, if $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$ and $\nabla^2 f(\boldsymbol{x}^*)$ is positive definite, then $\boldsymbol{x}^*$ 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 13.4. 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 $\mathbb{E}[\epsilon] = 0$. As function class we take

$$\mathcal{H} = \{h \colon h(\boldsymbol{x}) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p,\ \boldsymbol{\beta} = (\beta_0, \dots, \beta_p) \in \mathbb{R}^{p+1}\}. \tag{13.3}$$

The goal is to determine the *model parameters* $\beta_0, \dots, \beta_p$ from data.

To determine these, we can collect $n \geq p$ sample realizations (from observations or experiments),

$$\{(y_i, x_{i1}, \dots, x_{ip}), \quad 1 \leq i \leq n\}.$$

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,

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

we can write the empirical risk concisely as

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

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

$$\text{minimize} \, \frac{1}{2n} \|\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}\|_2^2.$$

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

$$\begin{aligned} f(\boldsymbol{\beta}) = \frac{1}{2n} \|\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}\|_2^2 &= \frac{1}{2n} (\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y})^\top (\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}) \\ &= \frac{1}{2n} \boldsymbol{\beta}^\top \boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{\beta} - 2\boldsymbol{y}^\top \boldsymbol{X} \boldsymbol{\beta} + \boldsymbol{y}^\top \boldsymbol{y}, \end{aligned} \tag{13.4}$$

where $\boldsymbol{X}^\top$ is the matrix transpose. Quadratic functions of the form (13.4) are convex, so this is a convex optimization problem. 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,

$$\boldsymbol{\beta}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}. \tag{13.5}$$

In practice one would not compute $\boldsymbol{\beta}^*$ by evaluating (13.5). There are more efficient methods available, such as gradient descent, the conjugate gradient method, and several variations of these. It is important to note that even in this simple example, solving the optimization problem can be problematic 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] The model we

use is $Y = \beta_0 + \beta_1 X$, with Y the basal metabolic rate and X the mass. Using data for 573 mammals from the PanTHERIA database[2], we can assemble the vector $\boldsymbol{y}$ and the matrix $\boldsymbol{X} \in \mathbb{R}^{n \times (p+1)}$ in order to compute the $\boldsymbol{\beta} = (\beta_0, \beta_1)^\top$. Here, $p = 1$ and $n = 573$. We illustrate how to solve this problem in Python. As usual, we first have to import some relevant libraries: **numpy** for numerical computation, **pandas** for loading and transforming datasets, **cvxpy** for convex optimization, and **matplotlib** for plotting.

In [1]:
```
# Import some important Python modules
import numpy as np
import pandas as pd
from cvxpy import *
import matplotlib.pyplot as plt
```

We next have to load the data. The data is saved in a table with 573 rows and 2 columns, where the first column list the mass and the second the basal metabolic rate.

In [2]:
```
# Load data into numpy array
bmr = pd.read_csv('../../data/bmr.csv',header=None).as_matrix()
# We can find out the dimension of the data
bmr.shape
```

Out [2]: (573, 2)

To see the first three and the last three rows of the dataset, we can use the "print" command.

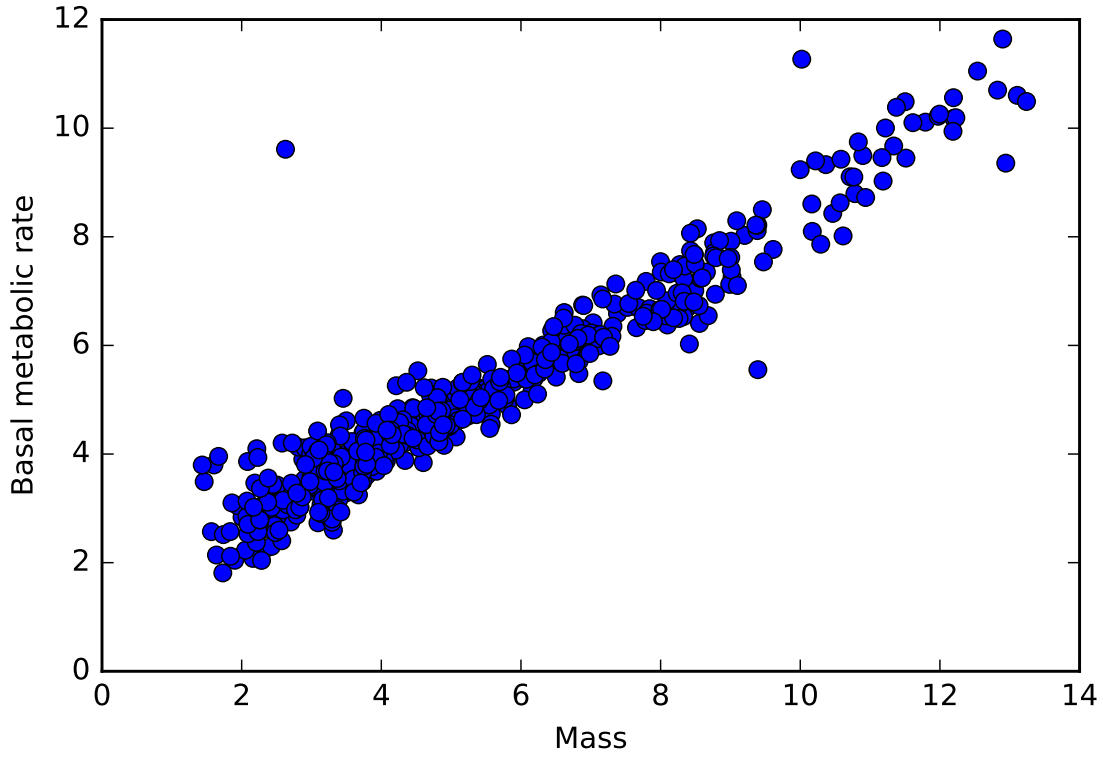
[1] This example is from the episode “Size Matters” of the BBC series Wonders of Life.
[2] http://esapubs.org/archive/ecol/E090/184/#data

In [3]:
```
print(bmr[0:3,:])
```

```
[[ 13.108   10.604 ]
 [  9.3918   8.2158]
 [ 10.366    9.3285]]
```

To visualise the whole dataset, we can make a scatterplot by interpreting each row as a coordinate on the plane, and marking it with a dot.

In [4]:
```
# Display scatterplot of data (plot all the rows as points)
bmr1 = plt.plot(bmr[:,0],bmr[:,1],'o')
plt.xlabel("Mass")
plt.ylabel("Basal metabolic rate")
plt.show()
```



The plot above suggests that the relation of the basal metabolic rate to the mass is linear, i.e., of the form

$$Y = \beta_0 + \beta_1 X,$$

where X is the mass and Y the BMR. We can find β_0 and β_1 by solving an optimization problem as described above. We first have to assemble the matrix $\boldsymbol{X}$ and the vector $\boldsymbol{y}$.

In [5]:
```
n = bmr.shape[0]
p = 1
X = np.concatenate((np.ones((n,1)),bmr[:,0:p]),axis=1)
y = bmr[:,-1]
```

In [6]:

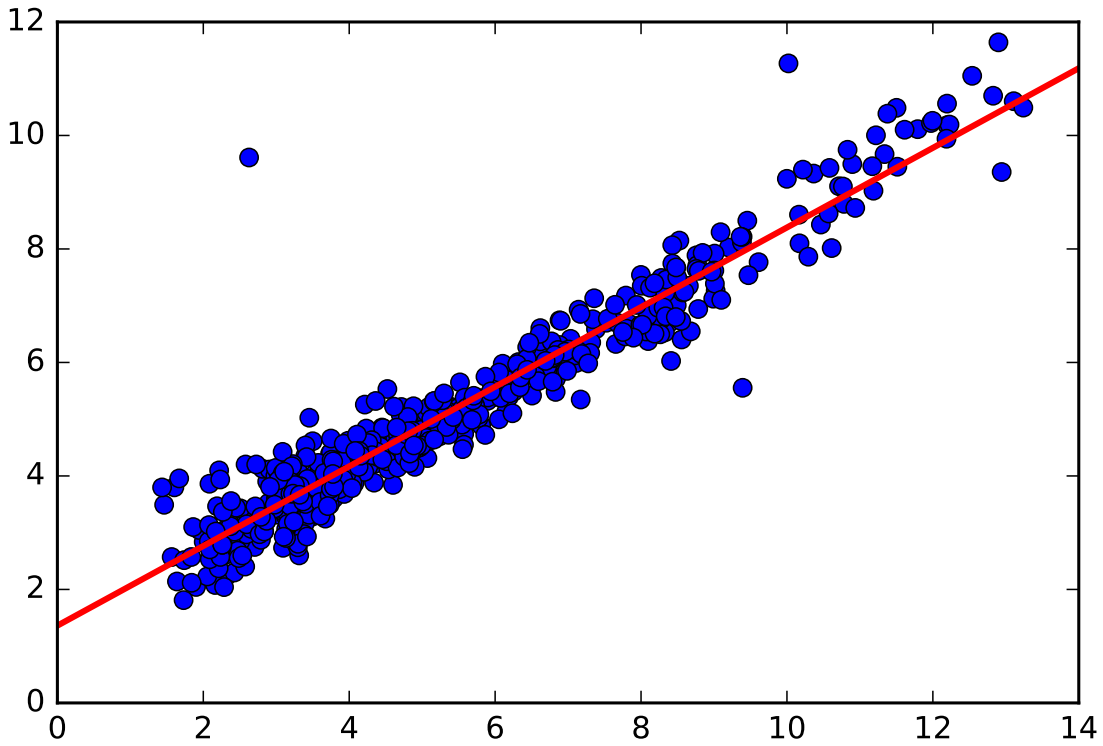
```
# Create a (p+1) vector of variables
Beta = Variable(p+1)

# Create sum-of-squares objective function
objective = Minimize(sum_entries(square(X*Beta - y)))

# Create problem and solve it
prob = Problem(objective)
prob.solve()

print("status: ", prob.status)
print("optimal value: ", prob.value)
print("optimal variables: ", Beta[0].value, Beta[1].value)
```

```
status:  optimal
optimal value:  152.736200529558
optimal variables:  1.3620698558275837 0.7016170245505547
```

Now that we solved the problem and have the values $\beta_0 = 1.362$ and $\beta_1 = 0.702$, we can plot the line and see how it fits the data.

In [6]:

```
plt.plot(bmr[:,0],bmr[:,1],'o')

xx = np.linspace(0,14,100)
bmr = plt.plot(xx, Beta[0].value+Beta[1].value*xx, color='red',\
 linewidth=2)
plt.show()
```



Even though for illustration purposes we used the CVXPY package, this particular problem can be solved directly using the least squares solver in numpy.

In [7]:
```
import numpy.linalg as la
beta = la.lstsq(X,y)
print(beta[0])
```

```
[ 1.36206997  0.70161692]
```

Example 13.5. (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.

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 $\Omega \subset [m] \times [n] = \{(i,j) \mid 1 \leq i \leq m, 1 \leq j \leq n\}$ 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

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

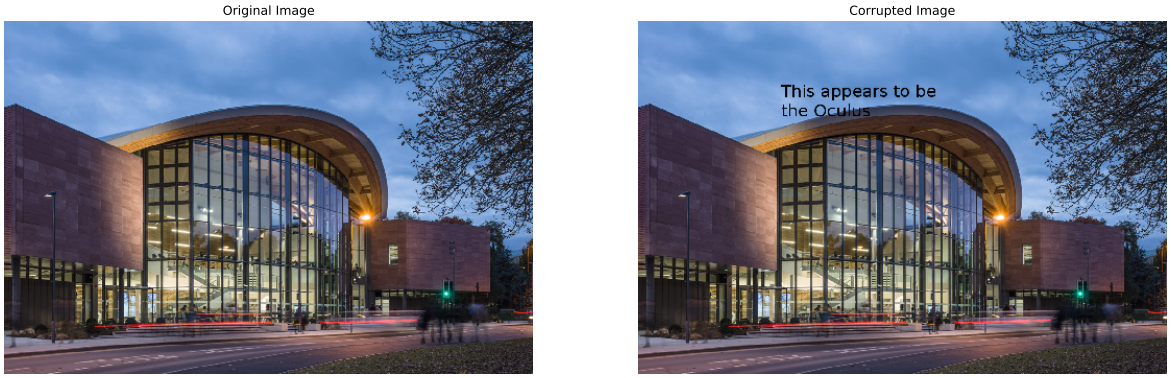
The TV-norm is an example of a convex function and the constraints are linear conditions which define a convex set. This is again an example of a **convex optimization problem** and can be solved efficiently by a range of algorithms. For the time being we will not go into the algorithms but solve it using CVXPY.

The example below is based on an example from the CVXPY Tutorial[3], and it is recommended to look at this tutorial for other interesting examples!

Warning: the example below uses some more advanced Python programming, it is not necessary to understand.

In our first piece of code below, we load the image and a version of the image with text written on it, and display the images. The **Python Image Library (PIL)** is used for this purpose.

In [9]:
```
from PIL import Image

# Load the images and convert to numpy arrays for processing.
U = np.array(Image.open("../images/oculus.png"))
Ucorr = np.array(Image.open("../images/oculus-corr.png"))

# Display the images
fig, ax = plt.subplots(1, 2,figsize=(10, 5))

ax[0].imshow(U);
ax[0].set_title("Original Image")
ax[0].axis('off')

ax[1].imshow(Ucorr);
ax[1].set_title("Corrupted Image")
ax[1].axis('off');
```



After having the images at our disposal, we determine which entries of the corrupted image are known. We store these in a *mask* M, with entries $m_{ijk} = 1$ if the colour k of the (i, j)-th pixel is known, and 0 otherwise.

In [10]:
```
# Each image is now an m x n x 3 array, with each pixel
# represented by three numbers between 0 and 255,
# corresponding to red, green and blue
rows, cols, colours = U.shape

# Create a mask: this is a matrix with a 1 if the corresponding
# pixel is known, and zero else
M = np.zeros((rows, cols, colours))
for i in range(rows):
    for j in range(cols):
        for k in range(colours):
            if U[i, j, k] == Ucorr[i, j, k]:
                M[i, j, k] = 1
```

[3] http://www.cvxpy.org/en/latest/tutorial/index.html

We are now ready to solve the optimization problem using CVXPY. As the problem is rather big (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. For the example at hand, we choose the SCS solver, which can be specified when calling the **solve** function.

In [11]:
```
# Determine the variables and constraints
variables = []
constraints = []
for k in range(colours):
    X = Variable(rows, cols)
    # Add variables
    variables.append(X)
    # Add constraints by multiplying the relevant variable matrix
    # elementwise with the mask
    constraints.append(mul_elemwise(M[:, :, k], X) ==
      \ (M[:, :, k], Ucorr[:, :, k]))

# Create a problem instance with
objective = Minimize(tv(variables[0],variables[1],variables[2]))

# Create a problem instance and solve it using the SCS solver
prob = Problem(objective, constraints)
prob.solve(verbose=True, solver=SCS)
```

Out [11]: 8263910.812250629

Now that we solved the optimization problem, we have a solution stored in 'variables'. We have to transform this back into an image and display the result.

In [12]:
```

# Load variable values into a single array.
Urec = np.zeros((rows, cols, colours), dtype=np.uint8)
for i in range(colours):
    Urec[:, :, i] = variables[i].value

fig, ax = plt.subplots(1, 2,figsize=(10, 5))

# Display the inpainted image.
ax[0].imshow(Urec);
ax[0].set_title("Inpainted Image")
ax[0].axis('off')

ax[1].imshow(np.abs(Ucorr[:,:,0:3] - Urec));
ax[1].set_title("Difference Image")
ax[1].axis('off');
```

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

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

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 applying the total variation inpainting procedure to set a parrot free.

Caged parrot

Free parrot

Notes

14

Convexity

Convexity is a central theme in optimization. The reason is that convex optimization problems have many favourable properties, such as lower computational complexity and the property that local minima are also global minima. Despite being a seemingly special property, convex optimization problems arise surprisingly often.

Convex functions

Definition 14.1. A set $C \subseteq \mathbb{R}^d$ is **convex** if for all $\boldsymbol{x}, \boldsymbol{y} \in C$ and $\lambda \in [0,1]$, the line $\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y} \in C$. A **convex body** is a convex set that is closed and bounded.

Definition 14.2. Let $S \subseteq \mathbb{R}^d$. A function $f\colon S \to \mathbb{R}$ is called *convex* if S is convex and for all $\boldsymbol{x}, \boldsymbol{y} \in S$ and $\lambda \in [0,1]$,

$$f(\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda)f(\boldsymbol{y}).$$

The function f is called *strictly convex* if

$$f(\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y}) < \lambda f(\boldsymbol{x}) + (1-\lambda)f(\boldsymbol{y}).$$

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

Figure 14.1 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 14.3. *Let $f\colon \mathbb{R}^d \to \mathbb{R}$ be a convex function. Then any local minimizer of f is a global minimizer.*

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

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

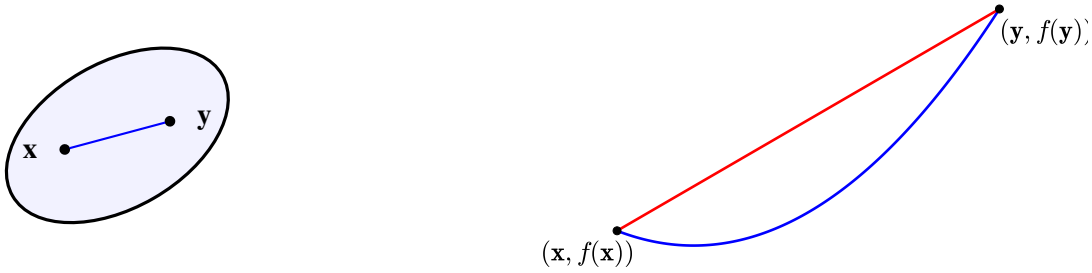


Figure 14.1: A convex set and a convex function

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

Remark 14.4. 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*, it need not be differentiable. The function $f(x) = |x|$ is a typical example: it is convex, but not differentiable at $x = 0$.

Example 14.5. Affine functions $f(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{a} \rangle + b$ and the exponential function e^x are examples of convex functions.

Example 14.6. 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 $\boldsymbol{A}^\top = \boldsymbol{A}$, 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

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

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 14.7. *1. Let $f \in C^1(\mathbb{R}^d)$. Then f is convex if and only if for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$,*

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x}).$$

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

Example 14.8. Consider a quadratic function of the form

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

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

$$\begin{aligned}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} &= \begin{pmatrix} x_1 & \cdots & x_n \end{pmatrix} \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \\ &= \begin{pmatrix} x_1 & \cdots & x_n \end{pmatrix} \begin{pmatrix} a_{11}x_1 + \cdots + a_{1n}x_n \\ \vdots \\ a_{n1}x_1 + \cdots + a_{nn}x_n \end{pmatrix} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_ix_j.\end{aligned}$$

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

$$\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} = \sum_{i=1}^{n} a_{ii}x_i^2 + 2\sum_{1\le i<j\le n} a_{ij}x_ix_j.$$

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

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

In summary, we have

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

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

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

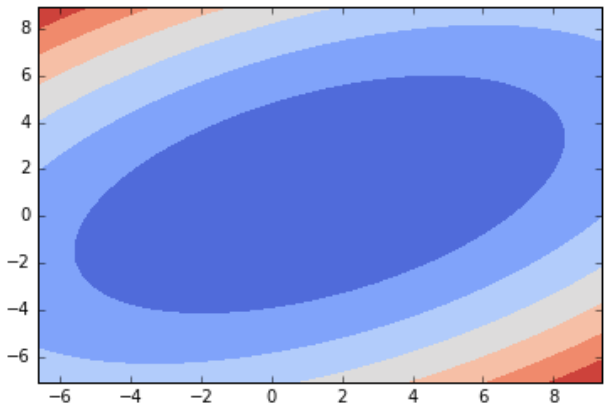
The matrix $\boldsymbol{A}^T\boldsymbol{A}$ is always symmetric and positive semidefinite (why?) so that the function f is convex.

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

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

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


8
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
8

15

Lagrangian Duality

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

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

where $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{f} = (f_1, \ldots, f_m)^\top$, $\boldsymbol{h} = (h_1, \ldots, h_p)^\top$, 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 $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}$ as $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, with $h_j(\boldsymbol{x}) = \boldsymbol{a}_j^\top \boldsymbol{x} - b_j$, $\boldsymbol{a}_j^\top$ being the j-th row of $\boldsymbol{A}$. The **feasible set** of (1) is the set

$$\mathcal{F} = \{\boldsymbol{x} \mid f_i(\boldsymbol{x}) \leq 0,\ \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$$

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 (1) 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 15.1. *Let $f \in C^1(\mathbb{R}^n)$ be a convex, differentiable function, and consider a convex optimization problem of the form* (1)*. Then $\boldsymbol{x}^*$ is an optimal point of the optimization problem*

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

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

$$\langle \nabla f(\boldsymbol{x}^*), \boldsymbol{y} - \boldsymbol{x}^* \rangle \geq 0, \tag{15.1}$$

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

Proof. Suppose $\boldsymbol{x}^*$ is such that (1) holds. Then, since f is a convex function, for all $\boldsymbol{y} \in \mathcal{F}$ we have,

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

which shows that $\boldsymbol{x}^*$ is a minimizer in $\mathcal{F}$. To show the opposite direction, assume that $\boldsymbol{x}^*$ is a minimizer but that (15.1) does not hold. This means that there exists a $\boldsymbol{y} \in \mathcal{F}$ such that $\langle \nabla f(\boldsymbol{x}^*), \boldsymbol{y} - \boldsymbol{x}^* \rangle < 0$. Since both $\boldsymbol{x}^*$ and $\boldsymbol{y}$ are in $\mathcal{F}$ and $\mathcal{F}$ is convex, any point $\boldsymbol{z}(\lambda) = (1-\lambda)\boldsymbol{x}^* + \lambda \boldsymbol{y}$ with $\lambda \in [0,1]$ is also in $\mathcal{F}$. At $\lambda = 0$ we have

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

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

Example 15.2. In the absence of constraints, $\mathcal{F} = \mathbb{R}^n$, and the statement says that

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

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

Geometrically, the first order optimality condition means that the set

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

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

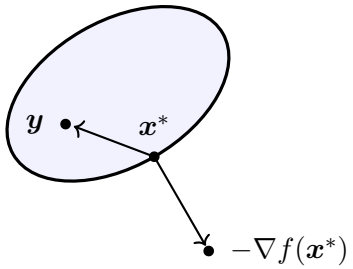


Figure 15.1: Optimality condition

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

$$\nabla f(x^*, y^*) = \lambda \nabla h(x^*, y^*). \tag{15.2}$$

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

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

then (15.2) 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 15.3. 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

$$\begin{aligned}
&\text{minimize} && f(\boldsymbol{x}) \\
&\text{subject to} && \boldsymbol{f}(\boldsymbol{x}) \leq \boldsymbol{0} \\
&&& \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0},
\end{aligned} \tag{15.3}$$

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 (15.3).

The *Lagrangian* of the system is defined as

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

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 15.4. The *Lagrange dual* of the problem (15.3) is the function

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

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 15.5. *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 $\boldsymbol{x}^*$ be a feasible point for (15.3), that is,

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

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

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

In particular,

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

Since this holds for *all* feasible $\boldsymbol{x}^*$, it holds in particular for the $\boldsymbol{x}^*$ that minimizes (15.3), for which $f(\boldsymbol{x}^*) = p^*$. □

A point $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ with $\boldsymbol{\lambda} \geq \mathbf{0}$ and $(\boldsymbol{\lambda}, \boldsymbol{\mu}) \in \operatorname{dom}(g)$ is called a **feasible point** of the dual problem. The **Lagrange dual problem** of the optimization problem (15.3) is the problem

$$\text{maximize } g(\boldsymbol{\lambda}, \boldsymbol{\mu}) \qquad \text{subject to} \quad \boldsymbol{\lambda} \geq \mathbf{0}. \tag{15.4}$$

If q^* is the optimal value of (15.4), then $q^* \leq p^*$. In the special case of linear programming we actually have $q^* = p^*$. We will see that under certain conditions, we have $q^* = p^*$ for more general problems, but this is not always the case.

Notes

16

KKT Conditions

For convex problems of the form

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & \boldsymbol{f}(\boldsymbol{x}) \leq \boldsymbol{0} \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \end{aligned} \tag{1}$$

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

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

We saw that $g(\boldsymbol{\lambda}, \boldsymbol{\mu})$ is a lower bound on the optimal value of (1). Note that here we wrote the conditions $h_j(\boldsymbol{x}) = 0$ as system of linear equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, since for the problem to be convex, we require that the h_j be linear functions. 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

Consider a linear programming problem of the form

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

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

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

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

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

From Lemma 15.5 we conclude that

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

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

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

In the example of linear programming, we have seen that the optimal value of the dual problem is equal to the optimal value of the primal problem. In general, we have

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

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 16.1. *(Slater conditions) Assume that the interior of the domain $\mathcal{D}$ of* (1) *is non-empty, that the problem* (1) *is convex, and that there exists a point $\boldsymbol{x} \in \mathcal{D}$ such that*

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

Then $d^* = p^*$, the primal optimal value coincides with the dual optimal value.

Example 16.2. The problem $\text{minimize}\, e^{-x}$ subject to $x^2/y \leq 0,\ y \geq 0$ is an example of a convex problem that does not satisfy strong duality.

Example 16.3. Consider the problem

$$\text{minimize}\, \boldsymbol{x}^\top \boldsymbol{x} \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}.$$

The Lagrangian is $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\mu}) = \boldsymbol{x}^\top \boldsymbol{x} + \boldsymbol{\mu}^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$. For any $\boldsymbol{\mu}$, we can find the infimum

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

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

$$\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\mu}) = 2\boldsymbol{x} + \boldsymbol{A}^\top \boldsymbol{\mu} = \boldsymbol{0},$$

which gives the solution

$$\boldsymbol{x} = -\frac{1}{2} \boldsymbol{A}^\top \boldsymbol{\mu}.$$

The dual function is therefore

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

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

$$-\frac{1}{4} \boldsymbol{\mu}^\top \boldsymbol{A}^\top \boldsymbol{A} \boldsymbol{\mu} - \boldsymbol{b}^\top \boldsymbol{\mu} \leq \inf\{\boldsymbol{x}^\top \boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}.$$

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

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{f}(\boldsymbol{x}) \leq \boldsymbol{0} \\ & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}, \end{array} \tag{16.1}$$

If p^* is the optimal solution of (16.1) 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 $\boldsymbol{x}$,

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

The difference $f(\boldsymbol{x}) - g(\boldsymbol{\lambda}, \boldsymbol{\mu})$ 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 $\boldsymbol{x}$ and $(\boldsymbol{\lambda}, \boldsymbol{\mu})$. For any such points we know that

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

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 $(\boldsymbol{x}^k, \boldsymbol{\lambda}^k, \boldsymbol{\mu}^k)$, then we can stop if the duality gap is less than, say, a predefined tolerance ε.

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

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

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

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

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

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

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

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

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 16.4. *(KKT conditions) Let $\boldsymbol{x}^*$ and $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ be primal and dual optimal solutions of* (16.1) *with zero duality gap. The the following conditions are satisfied:*

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

Moreover, if the problem is convex and the Slater Conditions (Theorem 16.1) are satisfied, then any points satisfying the KKT conditions have zero duality gap.

Notes

17

Support Vector Machines I

In this lecture we return to the task of classification. As seen earlier, examples include spam filters, letter recognition, or text classification. In this lecture we 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 $h(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$ 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

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

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

$$y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) - 1 \geq 0. \tag{17.1}$$

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 (17.1) is $\delta = 2/\|\boldsymbol{w}\|_2$. Given a hyperplane H described in (17.1) and a point $\boldsymbol{x}$ such that we have the equality $\boldsymbol{w}^\top \boldsymbol{x} + b = 1$ (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 $\langle \boldsymbol{w}, c\boldsymbol{w} \rangle + b =$

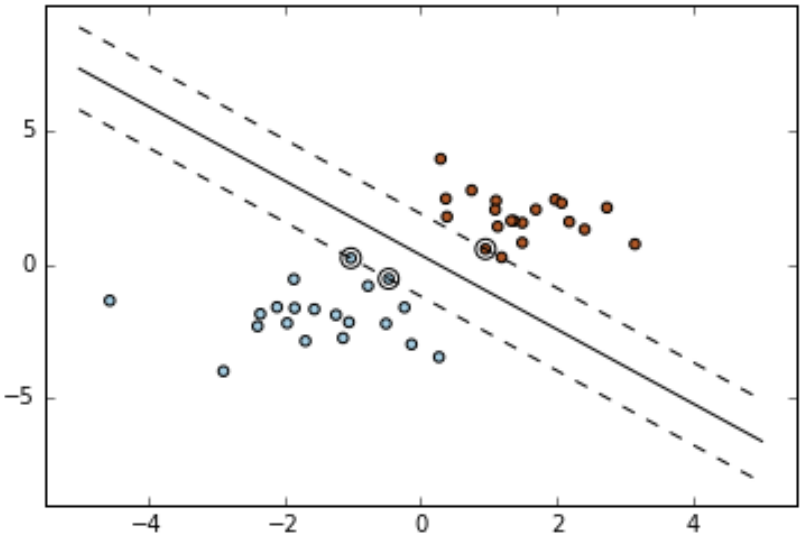


Figure 17.1: A hyperplane separating two sets of points with margin and support vectors.

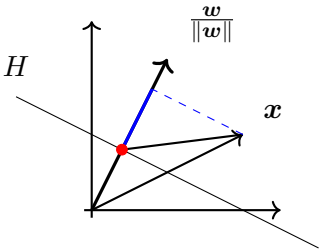


Figure 17.2: Computing the distance to the hyperplane

0. This implies that $c = -b/\|\boldsymbol{w}\|^2$, and consequently $\boldsymbol{p} = -(b/\|\boldsymbol{w}\|^2)\boldsymbol{w}$. The distance is then

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

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

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

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

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

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

$$\begin{aligned}\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda}) &= \boldsymbol{w} - \boldsymbol{X}^\top \boldsymbol{\lambda} = \boldsymbol{0} \\ \frac{\partial \mathcal{L}}{\partial b}(\boldsymbol{w}, b, \boldsymbol{\lambda}) &= \boldsymbol{y}^\top \boldsymbol{\lambda} = 0.\end{aligned} \tag{17.2}$$

If $\boldsymbol{y}^T \boldsymbol{\lambda} \neq 0$, then the conditions (17.2) cannot be satisfied and the problem is unbounded from below. If $\boldsymbol{y}^T \boldsymbol{\lambda} = 0$, then the first condition in (17.2) is necessary and sufficient for a minimizer. Replacing $\boldsymbol{w}$ by $\boldsymbol{X}^\top \boldsymbol{\lambda}$ and $\boldsymbol{\lambda}^\top \boldsymbol{y}$ by 0 in the Lagrangian function then gives the expression for the Lagrange dual $g(\boldsymbol{\lambda})$,

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

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

$$\text{minimize } \frac{1}{2}\boldsymbol{\lambda}^\top \boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{\lambda} - \boldsymbol{\lambda}^\top \boldsymbol{e} \quad \text{subject to} \quad \boldsymbol{\lambda} \geq \boldsymbol{0}, \tag{17.3}$$

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

Notes

18

Support Vector Machines II

The linear separation problem was introduced as the problem of finding a hyperplane

$$H = \{\boldsymbol{x} \in \mathbb{R}^d \colon \boldsymbol{w}^T \boldsymbol{x} + b = 0\}$$

that would separate data points $\boldsymbol{x}_i$ with label $y_i = 1$ from data points $\boldsymbol{x}_j$ with label $y_j = -1$. For convenience, we collect our data in matrices and vectors, where $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ is the matrix with rows $y_i \boldsymbol{x}_i^\top$ and $\boldsymbol{y}$ the vectors of labels y_i.

Assuming that the data is linearly separable, the problem of finding a separating hyperplane of maximal margin can 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

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

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 $\boldsymbol{w} = \boldsymbol{X}^\top \boldsymbol{\lambda}$. 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

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

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

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

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

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

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

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 $\boldsymbol{X}\boldsymbol{X}^\top$ is precisely $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$!

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

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

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 $\boldsymbol{w}^\top \boldsymbol{x}$ 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 18.1. 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:

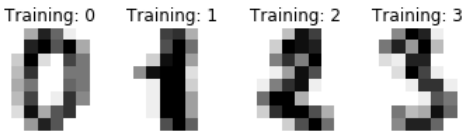
- **Separate** the data set randomly into *training data* and *test data*;
- **Create** a support vector classifier with optional parameters;
- **Train** (using FIT) the classifier with the training data;
- **Predict** the response using the test data and compare with the true response;
- **Report** 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 [15]:
```
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 [16]:
```
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

19

Iterative Algorithms

Most modern optimization methods are iterative: they generate a sequence of points $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots$ in $\mathbb{R}^d$ in the hope that this sequences will converge to a local or global minimizer $\boldsymbol{x}^*$ of a function $f(\boldsymbol{x})$. A typical rule for generating such a sequence would be to start with a vector $\boldsymbol{x}_0$, chosen by an educated guess, and then for $k \geq 0$, move from step k to $k+1$ by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k,$$

in a way that ensures that $f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k)$. The parameter α_k is called the **step length**, 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

$$\boldsymbol{p}_k = -\nabla f(\boldsymbol{x}_k), \tag{19.1}$$

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

$$\boldsymbol{p}_k = -\nabla^2 f(\boldsymbol{x}_k)^{-1} \nabla f(\boldsymbol{x}_k). \tag{19.2}$$

The resulting method is called **Newton's Method**. If applicable, Newton's method tends to converge 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

$$\boldsymbol{p}_k = -\nabla f(\boldsymbol{x}_k).$$

To see why this makes sense, let $\boldsymbol{p}$ be a direction and consider the Taylor expansion

$$f(\boldsymbol{x}_k + \alpha \boldsymbol{p}) = f(\boldsymbol{x}_k) + \alpha \langle \boldsymbol{p}, \nabla f(\boldsymbol{x}_k) \rangle + O(\alpha^2).$$

Considering this as a function of α, the rate of change in direction $\boldsymbol{p}$ at $\boldsymbol{x}_k$ is the derivative of this function at $\alpha = 0$,

$$\frac{\mathrm{d} f(\boldsymbol{x}_k + \alpha \boldsymbol{p})}{\mathrm{d}\alpha}|_{\alpha=0} = \langle \boldsymbol{p}, \nabla f(\boldsymbol{x}_k) \rangle,$$

also known as the **directional derivative** of f at $\boldsymbol{x}_k$ in the direction $\boldsymbol{p}$. This formula indicates that the rate of change is *negative*, and we have a **descent direction**, if $\langle \boldsymbol{p}, \nabla f(\boldsymbol{x}_k) \rangle < 0$.

The Cauchy-Schwarz inequality gives the bounds

$$-\|\boldsymbol{p}\|_2\|\nabla f(\boldsymbol{x}_k)\|_2 \leq \langle \boldsymbol{p}, \nabla f(\boldsymbol{x}_k)\rangle \leq \|\boldsymbol{p}\|_2\|\nabla f(\boldsymbol{x}_k)\|_2.$$

We see that the rate of change is the smallest when the first inequality is an equality, which happens if

$$\boldsymbol{p} = -\alpha\nabla f(\boldsymbol{x}_k)$$

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 $\nabla f(\boldsymbol{x})$ at a point $\boldsymbol{x}$ is given by (see Preliminaries, Page 9)

$$\langle \boldsymbol{x}, \nabla f(\boldsymbol{x})\rangle = \|\boldsymbol{p}\|_2\|\nabla f(\boldsymbol{x})\|_2 \cos(\theta).$$

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.

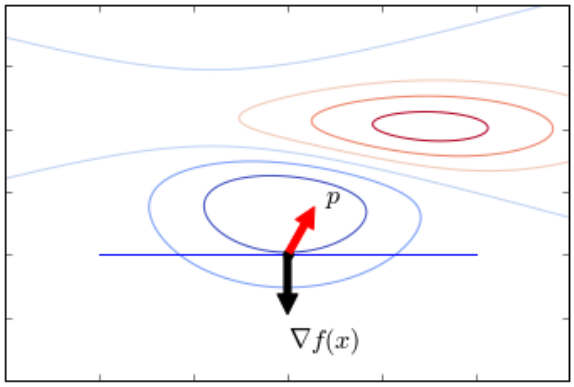


Figure 19.1: A descent direction

Any multiple $\alpha\nabla f(\boldsymbol{x}_k)$ 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 $f(\boldsymbol{x}_k - \alpha_k\nabla f(\boldsymbol{x}_k))$. 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

$$f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2.$$

The Hessian is symmetric and positive semidefinite, with the gradient given by

$$\nabla f(\boldsymbol{x}) = \boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}).$$

The method of gradient descent proceeds as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{A}^\top (\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{b}).$$

To find the best α_k, we compute the minimum of the function

$$\alpha \mapsto \varphi(\alpha) = f(\boldsymbol{x}_k - \alpha \boldsymbol{A}^\top (\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{b})). \tag{19.3}$$

If we set $\boldsymbol{r}_k := \boldsymbol{A}^\top (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k) = -\nabla f(\boldsymbol{x}_k)$ and compute the minimum of (19.3) by setting the derivative to zero,

$$\begin{aligned}
\varphi'(\alpha) &= \frac{d}{d\alpha} f(\boldsymbol{x}_k + \alpha \boldsymbol{r}_k) = \langle \nabla f(\boldsymbol{x}_k + \alpha \boldsymbol{r}_k), \boldsymbol{r}_k \rangle \\
&= \langle \boldsymbol{A}^\top (\boldsymbol{A}(\boldsymbol{x}_k + \alpha \boldsymbol{r}_k) - \boldsymbol{b}), \boldsymbol{r}_k \rangle \\
&= \langle \boldsymbol{A}^\top (\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{b}), \boldsymbol{r}_k \rangle + \alpha^2 \langle \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{r}_k, \boldsymbol{r}_k \rangle \\
&= -\boldsymbol{r}_k^\top \boldsymbol{r}_k + \alpha \boldsymbol{r}_k^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{r}_k = 0,
\end{aligned}$$

we get the step length

$$\alpha_k = \frac{\boldsymbol{r}_k^\top \boldsymbol{r}_k}{\boldsymbol{r}_k^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{r}_k} = \frac{\|\boldsymbol{r}_k\|_2^2}{\|\boldsymbol{A}\boldsymbol{r}_k\|_2^2}.$$

Note also that when we have $\boldsymbol{r}_k$ and α_k, we can compute the next $\boldsymbol{r}_k$ as

$$\begin{aligned}
r_{k+1} &= \boldsymbol{A}^\top (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{k+1}) \\
&= \boldsymbol{A}^\top (\boldsymbol{b} - \boldsymbol{A}(\boldsymbol{x}_k + \alpha_k \boldsymbol{r}_k)) \\
&= \boldsymbol{A}^\top (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k - \alpha_k \boldsymbol{A}\boldsymbol{r}_k) = \boldsymbol{r}_k - \alpha_k \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{r}_k.
\end{aligned}$$

The gradient descent algorithm for the linear least squares problem proceeds by first computing $\boldsymbol{r}_0 = \boldsymbol{A}^\top (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0)$, and then at each step

$$\begin{aligned}
\alpha_k &= \frac{\boldsymbol{r}_k^\top \boldsymbol{r}_k}{\boldsymbol{r}_k^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{r}_k} \\
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{r}_k \\
\boldsymbol{r}_{k+1} &= \boldsymbol{r}_k - \alpha_k \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{r}_k.
\end{aligned}$$

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 $\boldsymbol{x}$ 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, $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2$, becomes smaller than some $\varepsilon > 0$.

Example 19.1. 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.

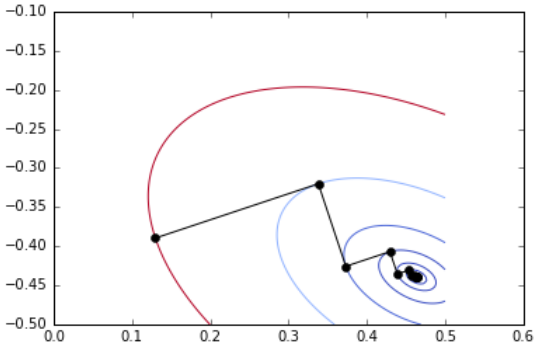


Figure 19.2: Trajectory of gradient descent

Step length selection

While for a quadratic function of the form $\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|^2$ 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.

20

Convergence

Iterative algorithms for solving a problem of the form

$$\text{minimize } f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^d \tag{20.1}$$

generate a sequence of vectors $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots$ in the hope that this sequence converges to a (local or global) minimizer $\boldsymbol{x}^*$ of (20.1). In this lecture we study what it means for a sequence to converge, and how to quantify the speed of convergence. We then study the convergence of of gradient descent for quadratic functions and for convex functions satisfying certain smoothness assumptions.

Convergence of iterative methods

A sequence of vectors $\{\boldsymbol{x}_k\}_{k\in\mathbb{N}_0} \subset \mathbb{R}^d$ *converges* to $\boldsymbol{x}^*$ with respect to a norm $\|\cdot\|$ as $k \to \infty$, written $\boldsymbol{x}_k \to \boldsymbol{x}$, if the sequence of numbers $\|\boldsymbol{x}_k - \boldsymbol{x}^*\|$ converges to zero. Iterative algorithms will typically not find the exact solution to a problem like (20.1). 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 20.1. Assume that a sequence of vectors $\{\boldsymbol{x}_k\}_{k\in\mathbb{N}_0}$ converges to $\boldsymbol{x}^*$. 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,

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\| \leq r\|\boldsymbol{x}_k - \boldsymbol{x}^*\|.$$

(b) **superlinearly**, if

$$\lim_{k\to\infty} \frac{\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\|}{\|\boldsymbol{x}_k - \boldsymbol{x}^*\|} = 0,$$

(c) with order p, if there exists a constant $M > 0$, such that for sufficiently large k,

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\| \leq M\|\boldsymbol{x}_k - \boldsymbol{x}^*\|^p.$$

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 $\{\boldsymbol{x}_k\}$ converges to $\boldsymbol{x}^*$. Therefore, for sufficiently large k, $\|\boldsymbol{x}_k - \boldsymbol{x}^*\| < 1$ and if $\{\boldsymbol{x}_k\}$ converges with order of convergence $p > 1$, then

$$\frac{\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\|}{\|\boldsymbol{x}_k - \boldsymbol{x}^*\|^{p-1}} \leq M\|\boldsymbol{x}_k - \boldsymbol{x}^*\| < M.$$

This shows that convergence of order p implies convergence of any lower order and also superlinear convergence.

Example 20.2. 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

$$\text{minimize } f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2, \tag{20.2}$$

where $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$ is a matrix of full rank. The function $f(\boldsymbol{x})$ 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

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{r}_k,$$

where the *step length* α_k and the *residual* $\boldsymbol{r}_k$ are given by

$$\alpha_k = \frac{\|\boldsymbol{r}_k\|^2}{\|\boldsymbol{A}\boldsymbol{r}_k\|^2}, \quad \boldsymbol{r}_k = \boldsymbol{A}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k) = -\nabla f(\boldsymbol{x}_k).$$

At the minimizer $\boldsymbol{x}^*$, the residual is $\boldsymbol{r} = -\nabla f(\boldsymbol{x}^*) = 0$ and if the sequence $\boldsymbol{x}_k$ converges to $\boldsymbol{x}^*$, the norms of the residuals converge to 0. Conversely, the residual is related to the difference $\boldsymbol{x}_k - \boldsymbol{x}^*$ by

$$\boldsymbol{r}_k = \boldsymbol{A}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k) = \boldsymbol{A}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k - (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^*)) = \boldsymbol{A}^\top\boldsymbol{A}(\boldsymbol{x_k} - \boldsymbol{x}^*). \tag{20.3}$$

Therefore

$$\|\boldsymbol{x}_k - \boldsymbol{x}^*\| = \|(\boldsymbol{A}^\top\boldsymbol{A})^{-1}\boldsymbol{r}_k\| \leq \|(\boldsymbol{A}^\top\boldsymbol{A})^{-1}\|\|\boldsymbol{r}_k\|,$$

where $\|\boldsymbol{B}\| = \max_{\boldsymbol{x}\neq\boldsymbol{0}} \|\boldsymbol{B}\boldsymbol{x}\|/\|\boldsymbol{x}\|$ 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 $\|\boldsymbol{x}_k - \boldsymbol{x}^*\|$. 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

$$\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\| \cdot \|\boldsymbol{A}^\dagger\|,$$

[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.

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 $\boldsymbol{A}^\dagger = (\boldsymbol{A}^\top \boldsymbol{A})^{-1} \boldsymbol{A}^\top$. The condition number is always greater than or equal to one.

Theorem 20.3. *The error in the $k+1$-th iterate is bounded by*

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\| \leq \left(\frac{\kappa^2(\boldsymbol{A}) - 1}{\kappa^2(\boldsymbol{A}) + 1} \right) \|\boldsymbol{x}_k - \boldsymbol{x}^*\|.$$

In particular, the gradient descent algorithm converges linearly. We can deduce Theorem 20.3 as a special case of a more general convergence result from convex function satisfying certain smoothness assumptions.

Notes

21

Gradient Descent

In this lecture we will derive a convergence result for gradient descent applied to a convex function $f \in C^1(\mathbb{R}^d)$. Convergence results in optimization are often stated in terms of the difference $f(\boldsymbol{x}^k) - f^*$, where $f^* = f(\boldsymbol{x}^*)$ and $\boldsymbol{x}^*$ is a minimizer of f. By the convexity of f, we have

$$f(\boldsymbol{x}^k) - f(\boldsymbol{x}^*) \leq \langle \nabla f(\boldsymbol{x}^k), \boldsymbol{x}^k - \boldsymbol{x}^* \rangle \leq \|\nabla f(\boldsymbol{x}^k)\| \cdot \|\boldsymbol{x}^k - \boldsymbol{x}^*\|, \tag{21.1}$$

which allows to relate the convergence of $f(\boldsymbol{x}^k) - f^*$ to the convergence of $\|\boldsymbol{x}^k - \boldsymbol{x}^*\|$. In order to guarantee good convergence rates, we need some additional smoothness and boundedness conditions.

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 21.1. A function $f\colon \mathbb{R}^d \to \mathbb{R}^k$ is called **Lipschitz continuous** with Lipschitz constant $L > 0$, if for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$,

$$\|f(\boldsymbol{x}) - f(\boldsymbol{y})\| \leq L \cdot \|\boldsymbol{x} - \boldsymbol{y}\|.$$

A function $f\colon \mathbb{R}^d \to \mathbb{R}$ is called β**-smooth** for some $\beta > 0$ if the gradient is Lipschitz continuous:

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\| \leq \beta \cdot \|\boldsymbol{x} - \boldsymbol{y}\|$$

for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$.

Lipschitz continuity lies somewhere between continuity and differentiability. If $f \in C^1(\mathbb{R}^d)$ is Lipschitz continuous with Lipschitz constant L, then $\|\nabla f(\boldsymbol{x})\| \leq L$. Similarly, β-smoothness implies that $\|\nabla^2 f(\boldsymbol{x})\| \leq \beta$. Recall from Lecture 14 that a function $f \in C^1(\mathbb{R}^d)$ is convex if and only if

$$f(\boldsymbol{y}) + \langle \nabla f(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \rangle \leq f(\boldsymbol{x}) \tag{21.2}$$

for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$. The following result shows that β-smoothness is equivalent to a quadratic upper bound on the difference between the function value $f(\boldsymbol{x})$ and its linear lower bound (21.2).

Lemma 21.2. *Let $f \in C^1(\mathbb{R}^d)$ be β-smooth and convex. Then for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$,*

$$f(\boldsymbol{x}) \leq f(\boldsymbol{y}) + \langle \nabla f(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \rangle + \frac{\beta}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2. \tag{21.3}$$

Conversely, if a convex function $f \in C^1(\mathbb{R}^d)$ satisfies (21.3), then for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$,

$$\frac{1}{\beta}\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|^2 \leq \langle \nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \rangle. \tag{21.4}$$

In particular, f is β-smooth.

Proof. The first inequality follows from the convexity assumption. For the second inequality, represent $f(\boldsymbol{x}) - f(\boldsymbol{y})$ as an integral:

$$f(\boldsymbol{x}) - f(\boldsymbol{y}) = \int_0^1 \langle \nabla f(\boldsymbol{y} + t(\boldsymbol{x} - \boldsymbol{y})), \boldsymbol{x} - \boldsymbol{y} \rangle \, \mathrm{d}t.$$

We can then write

$$\begin{aligned} f(\boldsymbol{x}) - f(\boldsymbol{y}) - \langle \nabla f(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \rangle &= \int_0^1 \langle \nabla f(\boldsymbol{y} + t(\boldsymbol{x} - \boldsymbol{y})), \boldsymbol{x} - \boldsymbol{y} \rangle \\ &\qquad - \langle \nabla f(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \rangle \, \mathrm{d}t \\ &\leq \int_0^1 \|\nabla f(\boldsymbol{y} + t(\boldsymbol{x} - \boldsymbol{y})) - \nabla f(\boldsymbol{y})\| \\ &\qquad \cdot \|\boldsymbol{x} - \boldsymbol{y}\| \, \mathrm{d}t \\ &\leq \beta \int_0^1 t\|\boldsymbol{x} - \boldsymbol{y}\|^2 \, \mathrm{d}t = \frac{\beta}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2, \end{aligned}$$

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 (21.3). For any $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}^d$ we have

$$\begin{aligned} f(\boldsymbol{x}) - f(\boldsymbol{y}) &= f(\boldsymbol{x}) - f(\boldsymbol{z}) + f(\boldsymbol{z}) - f(\boldsymbol{y}) \\ &\leq \langle \nabla f(\boldsymbol{x}), \boldsymbol{x} - \boldsymbol{z} \rangle + \langle \nabla f(\boldsymbol{y}), \boldsymbol{z} - \boldsymbol{y} \rangle + \frac{\beta}{2}\|\boldsymbol{z} - \boldsymbol{y}\|^2, \end{aligned}$$

where we used the convexity of f to bound $f(\boldsymbol{x}) - f(\boldsymbol{z})$ and the β-smoothness to bound $f(\boldsymbol{z}) - f(\boldsymbol{y})$. If we now set $\boldsymbol{z} = \boldsymbol{y} - (1/\beta)(\nabla f(\boldsymbol{y}) - \nabla f(\boldsymbol{x}))$ and simplify the resulting expression, we get

$$f(\boldsymbol{x}) - f(\boldsymbol{y}) \leq \langle \nabla f(\boldsymbol{x}), \boldsymbol{x} - \boldsymbol{y} \rangle - \frac{1}{2\beta}\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|^2.$$

Adding this expression to the same one with the roles of $\boldsymbol{x}$ and $\boldsymbol{y}$ exchanged, we get

$$0 \leq (\nabla f(\boldsymbol{x}) - \langle \nabla f(\boldsymbol{y})), \boldsymbol{x} - \boldsymbol{y} \rangle - \frac{1}{\beta}\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|^2.$$

The fact that this implies β-smoothness of f follows from the Cauchy-Schwartz inequality. □

We can, and will, use (21.3) and (21.4) as alternative definitions of β-smoothness.

Theorem 21.3. *Let $f \in C^1(\mathbb{R}^d)$ be a function that is β-smooth and convex. Then for any $\boldsymbol{x}^0 \in \mathbb{R}^d$, the iterates $\{\boldsymbol{x}^k\}$ generated by gradient descent with constant step length $1/\beta$ satisfy*

$$f(\boldsymbol{x}^k) - f^* \leq \frac{2\beta}{k}\|\boldsymbol{x}^0 - \boldsymbol{x}^*\|^2,$$

where $f^* = f(\boldsymbol{x}^*)$ and $\boldsymbol{x}^*$ is a minimizer of f.

Proof. Observe first that

$$\begin{aligned}\|\boldsymbol{x}^{k+1}-\boldsymbol{x}^*\|^2 &= \|\boldsymbol{x}^k-(1/\beta)\nabla f(\boldsymbol{x}^k)-\boldsymbol{x}^*\|^2\\ &= \|\boldsymbol{x}^k-\boldsymbol{x}^*\|^2-\frac{2}{\beta}\langle\nabla f(\boldsymbol{x}^k),\boldsymbol{x}^k-\boldsymbol{x}^*\rangle+\frac{1}{\beta^2}\|\nabla f(\boldsymbol{x}^k)\|^2\\ &\overset{(21.4)}{\leq} \|\boldsymbol{x}^k-\boldsymbol{x}^*\|^2-\frac{1}{\beta^2}\|\nabla f(\boldsymbol{x}^k)\|^2\leq\|\boldsymbol{x}^k-\boldsymbol{x}^*\|^2,\end{aligned}$$

where in addition to (21.4) we used the fact that $\nabla f(\boldsymbol{x}^*)=0$. In particular, since $\|\boldsymbol{x}^k-\boldsymbol{x}^*\|$ is non-increasing, we get from (21.1) that

$$f(\boldsymbol{x}^k)-f^*\leq\|\nabla f(\boldsymbol{x}^k)\|\cdot\|\boldsymbol{x}^0-\boldsymbol{x}^*\|. \tag{21.5}$$

Using (21.3) with $\boldsymbol{y}=\boldsymbol{x}^k$ and $\boldsymbol{x}=\boldsymbol{x}^k-(1/\beta)\nabla f(\boldsymbol{x}^k)=\boldsymbol{x}^{k+1}$, we get

$$f(\boldsymbol{x}^{k+1})-f(\boldsymbol{x}^k)\leq-\frac{1}{2\beta}\|\nabla f(\boldsymbol{x}^k)\|^2.$$

Set $\Delta_k=f(\boldsymbol{x}^k)-f^*$, so that $f(\boldsymbol{x}^{k+1})-f(\boldsymbol{x}^k)=\Delta_{k+1}-\Delta^k$. Then

$$\Delta_{k+1}-\Delta_k\leq-\frac{1}{2\beta}\|\nabla f(\boldsymbol{x}^k)\|^2\overset{(21.5)}{\leq}-\frac{1}{2\beta\|\boldsymbol{x}^0-\boldsymbol{x}^*\|^2}\Delta_k^2, \tag{21.6}$$

where we used (21.5) to lower-bound $\|\nabla f(\boldsymbol{x}^k)\|$ in the second inequality. In particular, we see that $\Delta_{k+1}\leq\Delta_k$. We can rearrange the inequality (21.6) to

$$\begin{aligned}\Delta_{k+1}+\frac{\Delta_k^2}{2\beta\|\boldsymbol{x}^0-\boldsymbol{x}^*\|}\leq\Delta_k &\Rightarrow \frac{1}{\Delta_k}+\frac{1}{2\beta\|\boldsymbol{x}^0-\boldsymbol{x}^*\|}\frac{\Delta_k}{\Delta_{k+1}}\leq\frac{1}{\Delta_{k+1}}\\ &\Rightarrow \frac{1}{\Delta_k}+\frac{1}{2\beta\|\boldsymbol{x}^0-\boldsymbol{x}^*\|}\leq\frac{1}{\Delta_{k+1}},\end{aligned}$$

where for the first implication we divided both sides by $\Delta_k\Delta_{k+1}$, and for the second implication we used that $\Delta_k/\Delta_{k+1}\geq 1$. Applying the same bound recursively to $1/\Delta_k$, we get

$$\frac{1}{\Delta_{k+1}}\geq\frac{k+1}{2\beta\|\boldsymbol{x}^0-\boldsymbol{x}^*\|}+\frac{1}{\Delta_0}\geq\frac{k+1}{2\beta\|\boldsymbol{x}^0-\boldsymbol{x}^*\|}.$$

Taking the inverse, and shifting the index from $k+1$ to k, we get

$$f(\boldsymbol{x}^k)-f^*=\Delta_k\leq\frac{2\beta\|\boldsymbol{x}^0-\boldsymbol{x}^*\|}{k},$$

as claimed. □

We can get even better convergence when assuming **strong convexity**.

Definition 21.4. A function $f\in C^1(\mathbb{R}^d)$ is called α**-strongly convex** for some $\alpha>0$ if for every $\boldsymbol{x},\boldsymbol{y}\in\mathbb{R}^d$,

$$f(\boldsymbol{y})+\langle\nabla f(\boldsymbol{y}),\boldsymbol{x}-\boldsymbol{y}\rangle+\frac{\alpha}{2}\|\boldsymbol{x}-\boldsymbol{y}\|^2\leq f(\boldsymbol{x}).$$

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 $f(\boldsymbol{x})-\alpha\|\boldsymbol{x}\|^2/2$ is convex.

Remark 21.5. The difference

$$\mathcal{D}_f(\boldsymbol{x}, \boldsymbol{y}) := f(\boldsymbol{x}) - f(\boldsymbol{y}) - \langle \nabla f(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \rangle$$

is called the **Bregman divergence** associated to f. To say that a function $f \in C^1(\mathbb{R}^d)$ is α-strongly convex and β-smooth is to say that for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$,

$$\frac{\alpha}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2 \leq \mathcal{D}_f(\boldsymbol{x}, \boldsymbol{y}) \leq \frac{\beta}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2.$$

This means that we can, locally, upper and lower bound the function by quadratic functions. In particular, $\beta \geq \alpha$.

Theorem 21.6. *Let $f \in C^1(\mathbb{R}^d)$ be a function that is α-strongly convex and β-smooth. Then for any $\boldsymbol{x}^0 \in \mathbb{R}^d$, the iterates of gradient descent with constant step length $2/(\alpha + \beta)$ satisfy*

$$\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\| \leq \left(\frac{\kappa - 1}{\kappa + 1}\right) \|\boldsymbol{x}^k - \boldsymbol{x}^*\|,$$

where $\kappa = \beta/\alpha$.

Example 21.7. Assume that $\alpha = \beta$ (and therefore $\kappa = 1$) in Theorem 21.6. Then

$$f(\boldsymbol{x}) = \frac{\alpha}{2}\|\boldsymbol{x}\|^2.$$

The gradient is $\nabla f(\boldsymbol{x}) = \alpha \boldsymbol{x}$. Starting with $\boldsymbol{x}^0 \in \mathbb{R}^d$, gradient descent with step length $2/(\alpha + \beta) = 1/\alpha$ gives

$$\boldsymbol{x}_1 = \boldsymbol{x}^0 - \boldsymbol{x}^0 = \boldsymbol{0} = \boldsymbol{x}^*,$$

so that this converges in a single iteration.

Example 21.8. Let $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2$ 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

$$\frac{1}{2}\left(\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2 - \|\boldsymbol{A}\boldsymbol{y} - \boldsymbol{b}\|^2\right) - (\boldsymbol{A}\boldsymbol{y} - \boldsymbol{b})^T \boldsymbol{A}(\boldsymbol{x} - \boldsymbol{y}) = \frac{1}{2}\|\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{y})\|^2. \tag{21.7}$$

The largest and smallest **singular values** of the matrix $\boldsymbol{A}$ are defined as

$$\sigma_1(\boldsymbol{A}) = \max_{\|\boldsymbol{x}\|=1} \|\boldsymbol{A}\boldsymbol{x}\|, \qquad \sigma_n(\boldsymbol{A}) = \min_{\|\boldsymbol{x}\|=1} \|\boldsymbol{A}\boldsymbol{x}\|.$$

The term (21.7) is therefore bounded from above and below by the squares of the *largest singular value* and by the *smallest singular value* of $\boldsymbol{A}$:

$$\sigma_1^2(\boldsymbol{A})\|\boldsymbol{x} - \boldsymbol{y}\|^2 \geq \|\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{y})\|^2 \geq \sigma_n^2(\boldsymbol{A})\|\boldsymbol{x} - \boldsymbol{y}\|^2.$$

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 Lecture 20.

The proof of Theorem 21.6 relies on the following auxiliary result.

Lemma 21.9. *Let f be α-strongly convex and β smooth. Then for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$,*

$$\langle \nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \rangle \geq \frac{\alpha\beta}{\alpha + \beta}\|\boldsymbol{x} - \boldsymbol{y}\|^2 + \frac{1}{\alpha + \beta}\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|^2.$$

Proof. Set $\varphi(\boldsymbol{x}) := f(\boldsymbol{x}) - \frac{\alpha}{2}\|\boldsymbol{x}\|^2$. Since f is α-strongly convex, $\varphi(\boldsymbol{x})$ is convex. Moreover, $\nabla\varphi(x) = \nabla f(\boldsymbol{x}) - \alpha x$. We therefore get

$$\begin{aligned}\varphi(\boldsymbol{x}) - \varphi(\boldsymbol{y}) - \langle\nabla\varphi(\boldsymbol{y}), \boldsymbol{x}-\boldsymbol{y}\rangle &\leq f(\boldsymbol{x}) - f(\boldsymbol{y}) - \langle\nabla f(\boldsymbol{y}), \boldsymbol{x}-\boldsymbol{y}\rangle \\ &\quad - \frac{\alpha}{2}(\|\boldsymbol{x}\|^2 - \|\boldsymbol{y}\|^2) + \alpha\langle\boldsymbol{y}, \boldsymbol{x}-\boldsymbol{y}\rangle \\ &\overset{\text{Lemma } 21.2}{\leq} \frac{\beta}{2}\|\boldsymbol{x}-\boldsymbol{y}\|^2 \\ &\quad - \frac{\alpha}{2}(\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2 - 2\langle\boldsymbol{y}, \boldsymbol{x}\rangle) \\ &= \frac{\beta-\alpha}{2}\|\boldsymbol{x}-\boldsymbol{y}\|^2.\end{aligned} \tag{21.8}$$

From Lemma 21.2 it follows that φ is β-smooth and satisfies the inequality

$$\langle\nabla\varphi(\boldsymbol{x}) - \nabla\varphi(\boldsymbol{y}), \boldsymbol{x}-\boldsymbol{y}\rangle \geq \frac{1}{\beta-\alpha}\|\nabla\varphi(\boldsymbol{x}) - \nabla\varphi(\boldsymbol{y})\|^2.$$

Replacing φ in this expression, we get

$$\langle\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y}) - \alpha\boldsymbol{x} + \alpha\boldsymbol{y}, \boldsymbol{x}-\boldsymbol{y}\rangle \geq \frac{1}{\beta-\alpha}\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y}) - \alpha\boldsymbol{x} + \alpha\boldsymbol{y}\|^2.$$

The left-hand side of this inequality gives

$$\langle\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y}) - \alpha\boldsymbol{x} + \alpha\boldsymbol{y}, \boldsymbol{x}-\boldsymbol{y}\rangle = \langle\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y}), \boldsymbol{x}-\boldsymbol{y}\rangle - \alpha\|\boldsymbol{x}-\boldsymbol{y}\|^2. \tag{21.9}$$

The right-hand side, on the other hand, gives

$$\begin{aligned}\frac{1}{\beta-\alpha}\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y}) - \alpha\boldsymbol{x} + \alpha\boldsymbol{y}\|^2 = \frac{1}{\beta-\alpha}\big(&\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|^2 + \alpha^2\|\boldsymbol{x}-\boldsymbol{y}\|^2 \\ &- 2\alpha\langle\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y}), \boldsymbol{x}-\boldsymbol{y}\rangle\big).\end{aligned} \tag{21.10}$$

Collecting the terms in (27.2) and (21.10) involving $\langle\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y}), \boldsymbol{x}-\boldsymbol{y}\rangle$ on the left, and the terms involving $\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|^2$ and $\|\boldsymbol{x}-\boldsymbol{y}\|^2$ on the right, we get

$$\frac{\alpha+\beta}{\beta-\alpha}\langle\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y}), \boldsymbol{x}-\boldsymbol{y}\rangle \geq \frac{\alpha\beta}{\beta-\alpha}\|\boldsymbol{x}-\boldsymbol{y}\|^2 + \frac{1}{\beta-\alpha}\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|^2.$$

Multiplying this expression with $(\beta-\alpha)/(\alpha+\beta)$ gives the desired inequality. □

Proof of Theorem 21.6. Set $\eta := 2/(\alpha+\beta)$. Since $\nabla f(\boldsymbol{x}^*) = 0$, we will omit this term whenever it appears in the following (so that we can think of $\nabla f(\boldsymbol{x}^k) - \nabla f(\boldsymbol{x}^*)$ whenever we see $\nabla f(\boldsymbol{x}^k)$).

$$\begin{aligned}\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\|^2 &= \|\boldsymbol{x}^k - \eta\nabla f(\boldsymbol{x}^k) - \boldsymbol{x}^*\|^2 = \|(\boldsymbol{x}^k - \boldsymbol{x}^*) - \eta\nabla f(\boldsymbol{x}^k)\|^2 \\ &= \|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2 - 2\eta\langle\nabla f(\boldsymbol{x}^k), \boldsymbol{x}^k - \boldsymbol{x}^*\rangle + \eta^2\|\nabla f(\boldsymbol{x}^k)\|^2 \\ &\leq \left(\frac{\beta-\alpha}{\beta+\alpha}\right)^2\|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2,\end{aligned}$$

where in the inequality we used Lemma 21.9 to bound the term $\langle\nabla f(\boldsymbol{x}^k), \boldsymbol{x}^k - \boldsymbol{x}^*\rangle$, and simplified the resulting expression. The claim now follows by dividing the numerator and the denominator by α. □

Using the bound

$$\left(\frac{\kappa-1}{\kappa+1}\right)^2 = \left(1-\frac{\kappa-2}{\kappa+1}\right)^2 \leq e^{-4/(\kappa+1)},$$

and the inequality $f(\boldsymbol{x}^k) - f^* \leq (\beta/2)\|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2$ from the β-smoothness assumption, we get the convergence bound

$$f(\boldsymbol{x}^k) - f^* \leq \frac{\beta\|\boldsymbol{x}^0 - \boldsymbol{x}^*\|^2}{2} \cdot e^{-\frac{4}{\kappa+1}k},$$

which is a considerable improvement over the $1/k$ convergence bound 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 [4]. A standard reference for many of the tools used in the analysis of gradient descent (and a wealth of additional information) [19].

22

Extensions of Gradient Descent

We have seen that for a β-smooth convex function $f \in C^1(\mathbb{R}^d)$, the sequence of iterates $\{\boldsymbol{x}^k\}_{k\geq 0}$ generated by gradient descent with step length $1/\beta$ satisfies

$$f(\boldsymbol{x}^{k+1}) - f(\boldsymbol{x}^*) = O(1/k),$$

where $\boldsymbol{x}^*$ is a minimizer of f. This implies that we need a number of iterations of order $O(1/\epsilon)$ to reach accuracy ϵ. If in addition the function f is α-strongly convex, then we get linear convergence with ratio $(\kappa-1)/(\kappa+1)$, where $\kappa = \beta/\alpha$. For the convergence of the function value, this implies

$$f(\boldsymbol{x}^{k+1}) - f(\boldsymbol{x}^*) = O(\mathrm{e}^{-4k/(\kappa+1)})$$

shows that only $O(\log(1/\epsilon))$ iterations are needed to reach accuracy ϵ. In this lecture we will have a look at two extensions of gradient descent. The first is an accelerated version, while the second extension covers common situations in which the function to be minimized is not differentiable.

Acceleration

Accelerated gradient descent, proposed by Y. Nesterov, begins with initial values $\boldsymbol{y}^0 = \boldsymbol{x}^0 = \boldsymbol{x}_{-1} \in \mathbb{R}^d$ and the proceeds as follows for $k \geq 0$:

$$\begin{aligned} \boldsymbol{y}^k &= \boldsymbol{x}^k + \frac{k-1}{k+2}(\boldsymbol{x}^k - \boldsymbol{x}^{k-1}) \\ \boldsymbol{x}^{k+1} &= \boldsymbol{y}^k - \eta \nabla f(\boldsymbol{y}^k) \end{aligned}$$

The method can be interpreted as carrying over some *momentum* from previous iterates: instead of only taking into account the current iterate, the gradient step is based on a combination of the current and the previous step. This method has favourable convergence properties.

Theorem 22.1. *Let $f \in C^1(\mathbb{R}^d)$ be convex and β-smooth. Then accelerated gradient descent with step length $1/\beta$ converges to a minimizer $\boldsymbol{x}^*$ of f with rate*

$$f(\boldsymbol{x}^k) - f(\boldsymbol{x}^*) \leq \frac{2\|\boldsymbol{x}^0 - \boldsymbol{x}^*\|^2}{\beta(k+1)^2}.$$

There are lower bounds that show that this rate is optimal for gradient methods.

Proximal gradients

The objective functions arising in machine learning often have the form

$$f(\boldsymbol{x}) = g(\boldsymbol{x}) + h(\boldsymbol{x}),$$

where both g and h are convex, but only g is differentiable. The term $h(\boldsymbol{x})$ is typically a regularization term.

Example 22.2. Consider the function

$$f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2 + \lambda\|\boldsymbol{x}\|_1,$$

where $\|\boldsymbol{x}\|_1 = \sum_{i=1}^{d} |x_i|$ and $\lambda > 0$. The problem of minimizing this function is often referred to as the LASSO problem in statistics. The purpose of the regularization term is to encourage **sparse** solutions. These are solutions that have only few entries that are significantly larger than 0.

Definition 22.3. Let f be a convex function. The **subdifferential** of f at $\boldsymbol{x}$ is the set

$$\partial f(\boldsymbol{x}) = \{\boldsymbol{g} \in \mathbb{R}^d \colon \forall \boldsymbol{y} \in \mathbb{R}^d, f(\boldsymbol{x}) + \boldsymbol{g}^T(\boldsymbol{y} - \boldsymbol{x}) \leq f(\boldsymbol{y})\}.$$

The elements of $\partial f(\boldsymbol{x})$ are called **subgradients**.

If f is differentiable at $\boldsymbol{x}$, then there exists only one subgradient, which coincides with the gradient.

Example 22.4. Consider the function $f(x) = |x|$. The differential is then

$$\partial f(x) = \begin{cases} 1 & x > 0 \\ [-1, 1] & x = 0 \\ -1 & x < 0 \end{cases}$$

Example 22.5. The subdifferential is additive, in the sence that if $\boldsymbol{A}$ and $\boldsymbol{B}$ are matrices and $f(\boldsymbol{x}) = g(\boldsymbol{A}\boldsymbol{x}) + h(\boldsymbol{B}\boldsymbol{x})$, then

$$\partial f(\boldsymbol{x}^0) = \boldsymbol{A}^T \partial g(\boldsymbol{A}\boldsymbol{x}) + \boldsymbol{B}^T \partial h(\boldsymbol{B}\boldsymbol{x}).$$

A special case is the 1-norm. Here, we can write

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{d} |\Pi_i(\boldsymbol{x})|,$$

where $\Pi_i(\boldsymbol{x}) = x_i$ is the projection on the i-th coordinate. It follows that the subdifferential of the 1-norm can be described as

$$\partial\|\boldsymbol{x}\|_1 = \{\boldsymbol{z} \colon z_i = \operatorname{sign}(x_i) \text{ if } x_i \neq 0, z_j \in [-1, 1] \text{ if } x_j = 0\}.$$

Using the subdifferential, we have the following optimality condition.

Theorem 22.6. *Let $f \colon \mathbb{R}^d \to \mathbb{R}$ be a convex function. Then $\boldsymbol{x}$ is a global minimizer of f if and only if $\boldsymbol{0} \in \partial f(\boldsymbol{x})$.*

For composite functions $f(\boldsymbol{x}) = g(\boldsymbol{x}) + h(\boldsymbol{x})$ with $g \in C^1(\mathbb{R}^d)$, this means that

$$-\nabla g(\boldsymbol{x}) \in \partial h(\boldsymbol{x}).$$

There are different possible strategies for generalizing gradient descent to make it work with non-smooth functions. One would be to simply pick a subgradient at each step and follow that direction. Note that this may not be a descent direction. One common strategy for composite functions is to perform a gradient descent step based on the smooth function g, and then project onto the subgradient of h. Projection onto the subgradient is done via the **proximal operator**

$$\operatorname{prox}_h(\boldsymbol{x}) = \arg\min_{\boldsymbol{y}} \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2 + h(\boldsymbol{y}).$$

Note that $\boldsymbol{x}^* := \operatorname{prox}_h(\boldsymbol{x})$ satisfies $\boldsymbol{x}^* - \boldsymbol{x} \in \partial h(\boldsymbol{x}^*)$. The **proximal gradient method** for minimizing a function of the form $f(\boldsymbol{x}) = g(\boldsymbol{x}) + h(\boldsymbol{x})$ starts with a vector $\boldsymbol{x}^0$, and then for $k \geq 0$ proceeds by computing

$$\boldsymbol{x}^{k+1} = \operatorname{prox}_{\eta h}\left(\boldsymbol{x}^k - \eta \nabla g(\boldsymbol{x}^k)\right).$$

Example 22.7. Recall the image inpainting problem from Lecture 13. 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. 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.

Now let $\boldsymbol{U}$ be an image with entries u_{ij}, and let $\Omega \subset [m] \times [n] = \{(i,j) \mid 1 \leq i \leq m, 1 \leq j \leq n\}$ 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

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

Alternatively (see Exercise 8.3), one can solve a *regularized* problem

$$\text{minimize } \|\mathcal{A}\boldsymbol{X} - \boldsymbol{U}_\Omega\|^2 + \lambda \|\boldsymbol{X}\|_{\mathrm{TV}},$$

where $\mathcal{A}$ represents the linear map projects $\boldsymbol{X}$ onto the entries indexed by Ω. This problem can be solved using proximal gradient methods.

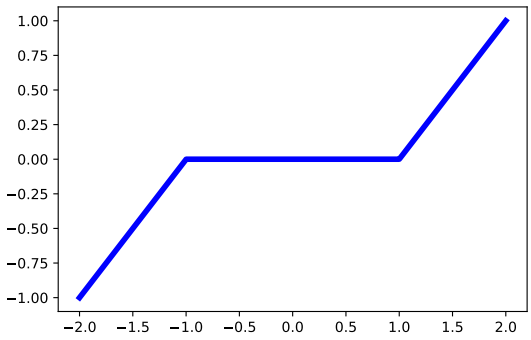


Figure 22.1: Soft thresholding

Even though it seems that a proximal gradient method would require solving an optimization problem within an optimization problem, closed form expressions are known in many cases.

Example 22.8. Consider the function $h(|x|) = \lambda|x|$ for some $\lambda > 0$. Then

$$\text{prox}_h(\boldsymbol{x}) = \mathcal{T}_\lambda(x) := \begin{cases} x - \lambda & x \geq \lambda \\ 0 & x \in [-\lambda, \lambda] \\ -x + \lambda & x \leq -\lambda \end{cases}$$

This is known as **soft thresholding** (see Figure 22.1).

If a function h has the form

$$h(\boldsymbol{x}) = \sum_{i=1}^{d} h_i(x_i),$$

then the proximal mapping associated to h has the form

$$\text{prox}_h(\boldsymbol{x}) = (\text{prox}_{h_1}(x_1), \ldots, \text{prox}_{h_d}(x_d))^T.$$

It follows that if $h(\boldsymbol{x}) = \lambda\|\boldsymbol{x}\|_1$, then we can apply the proximal operator by applying the soft thresholding operator $\mathcal{T}_\lambda$ to each coordinate of $\boldsymbol{x}$.

For the proximal gradient method it is possible to obtain similar convergence results as for gradient descent.

Notes

Accelerated gradient descent goes back to Nesterov's work [20]. A more in depth analysis can be found in [19] and [4]. An interesting interpretation of accelerated gradient descent in terms of differential equations is given in [26]. The proximal operator is discussed in detail in Chapter 6 of [1].

23

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. Hence, as usual when dealing only with optimization problems, we switch notation and denote the variables to be optimized over by $\boldsymbol{x}$.

Stochastic Gradient Descent

We consider an objective function of the form

$$f(\boldsymbol{x}) = \frac{1}{n} \sum_{i=1}^{n} f_i(\boldsymbol{x}). \tag{23.1}$$

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 $f(\boldsymbol{x})$ is an estimator of the generalization risk $\mathbb{E}_{\boldsymbol{\xi}}[f_{\boldsymbol{\xi}}(\boldsymbol{x})]$ 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 $f_j(\boldsymbol{x})$ gives such an unbiased estimator *by definition*:

$$\mathbb{E}_U[f_U(\boldsymbol{x})] = \frac{1}{n} \sum_{i=1}^{n} f_i(\boldsymbol{x}),$$

where $\mathbb{P}\{U = j\} = 1/n$ for $j \in [n] = \{1, \ldots, n\}$. The Stochastic Gradient Descent (SGD) algorithm proceeds as follows. Begin with $\boldsymbol{x}^0 \in \mathbb{R}^d$. At each step k, compute an unbiased estimator $\boldsymbol{g}^k$ of the gradient at $\boldsymbol{x}^k$:

$$\mathbb{E}[\boldsymbol{g}^k \mid \boldsymbol{x}^k] = \nabla f(\boldsymbol{x}^k).$$

Next, take a step in direction $\boldsymbol{g}^k$:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \eta_k \boldsymbol{g}^k,$$

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 $\nabla f_i(\boldsymbol{x})$ 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 $I \subset \{1, \ldots, n\}$ 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

$$\mathbb{E}_U[\nabla f_U(\boldsymbol{x})] = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\boldsymbol{x}) = \nabla f(\boldsymbol{x}),$$

so that ∇f_U is an unbiased estimator of ∇f. Assuming that f has a unique minimizer $\boldsymbol{x}^*$, we define the empirical **variance** at the optimal point $\boldsymbol{x}^*$ as

$$\sigma^2 = \mathbb{E}_U[\|\nabla f_U(\boldsymbol{x}^*)\|^2] = \frac{1}{n} \sum_{i=1}^{n} \|\nabla f_i(\boldsymbol{x}^*)\|^2. \tag{23.2}$$

We can now state the convergence result for stochastic gradient descent.

Theorem 23.1. *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 $\boldsymbol{x}^*$ and define the variance as in (23.2). Then for any staring point $\boldsymbol{x}^0$, the sequence of iterates $\{\boldsymbol{x}^k\}$ generated by SGD with uniform sampling and step length $\eta = 1/(2\beta)$ satisfies*

$$\mathbb{E}[\|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2] \leq \left(1 - \frac{\alpha}{4\beta}\right)^k \|\boldsymbol{x}^0 - \boldsymbol{x}^*\|^2 + \frac{2\sigma^2}{\alpha\beta}.$$

Proof. As in the analysis of gradient descent, we get

$$\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\|^2 = \|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2 - 2\eta\langle \nabla f_U(\boldsymbol{x}^k), \boldsymbol{x}^k - \boldsymbol{x}^*\rangle + \eta^2 \|\nabla f_U(\boldsymbol{x}^k)\|^2.$$

Taking the expectation conditional on $\boldsymbol{x}^k$, we get

$$\begin{aligned} \mathbb{E}[\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\|^2 \mid \boldsymbol{x}^k] = \|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2 &- 2\eta\langle \nabla f(\boldsymbol{x}^k), \boldsymbol{x}^k - \boldsymbol{x}^*\rangle \\ &+ \eta^2\, \mathbb{E}[\|\nabla f_U(\boldsymbol{x}^k)\|^2 \mid \boldsymbol{x}^k], \end{aligned} \tag{23.3}$$

where we used the fact that the expectation satisfies $\mathbb{E}[\nabla f_U(\boldsymbol{x})] = \nabla f(\boldsymbol{x})$. For the last term we use the bound

$$\begin{aligned} \mathbb{E}[\|\nabla f_U(\boldsymbol{x}^k)\|^2 \mid \boldsymbol{x}^k] &= \mathbb{E}[\|\nabla f_U(\boldsymbol{x}^k) - \nabla f_U(\boldsymbol{x}^*) + \nabla f_U(\boldsymbol{x}^*)\|^2] \\ &\leq 2\,\mathbb{E}[\|\nabla f_U(\boldsymbol{x}^k) - \nabla f_U(\boldsymbol{x}^*)\|^2] + 2\,\mathbb{E}[\|\nabla f_U(\boldsymbol{x}^*)\|^2] \\ &= 2\,\mathbb{E}[\|\nabla f_U(\boldsymbol{x}^k) - \nabla f_U(\boldsymbol{x}^*)\|^2] + 2\sigma^2. \end{aligned}$$

Using the characterization of β smoothness from Lecture 21, namely

$$\frac{1}{\beta}\|\nabla f_i(\boldsymbol{x}) - \nabla f_i(\boldsymbol{y})\|^2 \leq \langle \nabla f_i(\boldsymbol{x}) - \nabla f_i(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y}\rangle, \tag{23.4}$$

we get that

$$\begin{aligned}
\mathbb{E}[\|\nabla f_U(\boldsymbol{x}^k) - \nabla f_U(\boldsymbol{x}^*)\|^2 \mid \boldsymbol{x}^k] &= \frac{1}{n}\sum_{i=1}^n \|\nabla f_i(\boldsymbol{x}^k) - \nabla f_i(\boldsymbol{x}^*)\|^2 \\
&\overset{(23.4)}{\leq} \frac{\beta}{n}\sum_{i=1}^n \langle \nabla f_i(\boldsymbol{x}^k) - \nabla f_i(\boldsymbol{x}^*), \boldsymbol{x}^k - \boldsymbol{x}^*\rangle \\
&= \frac{\beta}{n}\sum_{i=1}^n \langle \nabla f_i(\boldsymbol{x}^k), \boldsymbol{x}^k - \boldsymbol{x}^*\rangle \\
&\qquad - \frac{\beta}{n}\sum_{i=1}^n \langle \nabla f_i(\boldsymbol{x}^*), \boldsymbol{x}^k - \boldsymbol{x}^*\rangle \\
&= \beta\langle \nabla f(\boldsymbol{x}), \boldsymbol{x}^k - \boldsymbol{x}^*\rangle,
\end{aligned}$$

where we used that $\nabla f(\boldsymbol{x}^*) = \boldsymbol{0}$ for the last equality. Hence, we get the bound

$$\mathbb{E}[\|\nabla f_U(\boldsymbol{x}^k)\|^2 \mid \boldsymbol{x}^k] \leq 2\beta\langle \nabla f(\boldsymbol{x}), \boldsymbol{x}^k - \boldsymbol{x}^*\rangle + 2\sigma^2.$$

Plugging this into (23.3), we get

$$\mathbb{E}[\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\|^2 \mid \boldsymbol{x}^k] \leq \|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2 - (2\eta - 2\eta^2\beta)\langle \nabla f(\boldsymbol{x}^k), \boldsymbol{x}^k - \boldsymbol{x}^*\rangle + 2\eta^2\sigma^2.$$

Using the α-strong convexity, we get the bound

$$\langle \nabla f(\boldsymbol{x}^k), \boldsymbol{x}^k - \boldsymbol{x}^*\rangle \geq f(\boldsymbol{x}^k) - f(\boldsymbol{x}^*) + \frac{\alpha}{2}\|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2 \geq \frac{\alpha}{2}\|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2,$$

since $f(\boldsymbol{x}^k) \geq f(\boldsymbol{x}^*)$, so that we get

$$\mathbb{E}[\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\|^2 \mid \boldsymbol{x}^k] \leq (1 - \eta\alpha(1 - \eta\beta))\|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2 + 2\eta^2\sigma^2.$$

With step length $\eta = 1/(2\beta)$, and taking the expected value over all previous iterates, we get

$$\mathbb{E}[\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\|^2] \leq \left(1 - \frac{\alpha}{4\beta}\right)\mathbb{E}[\|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2] + \frac{\sigma^2}{2\beta^2}.$$

Applying this bound recursively (and moving the index down), we get

$$\begin{aligned}
\mathbb{E}[\|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2] &\leq \left(1 - \frac{\alpha}{4\beta}\right)^k \|\boldsymbol{x}^0 - \boldsymbol{x}^*\|^2 + \frac{\sigma^2}{2\beta^2}\sum_{j=0}^{k-1}\left(1 - \frac{\alpha}{4\beta}\right)^j \\
&\leq \left(1 - \frac{\alpha}{4\beta}\right)^k \|\boldsymbol{x}^0 - \boldsymbol{x}^*\|^2 + \frac{2\sigma^2}{\alpha\beta},
\end{aligned}$$

where we used that $4\beta > \alpha$. □

Example 23.2. 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 (Exercise 7.6(a)), and the gradient is

$$\nabla f(\boldsymbol{x}) = -\boldsymbol{X}^T (\boldsymbol{y} - \boldsymbol{p}(\boldsymbol{w})),$$

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).

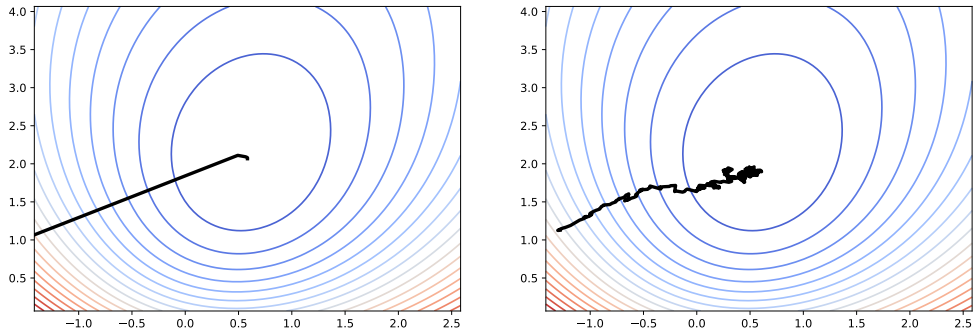


Figure 23.1: The path of gradient descent and of stochastic gradient descent with constant step length.

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.

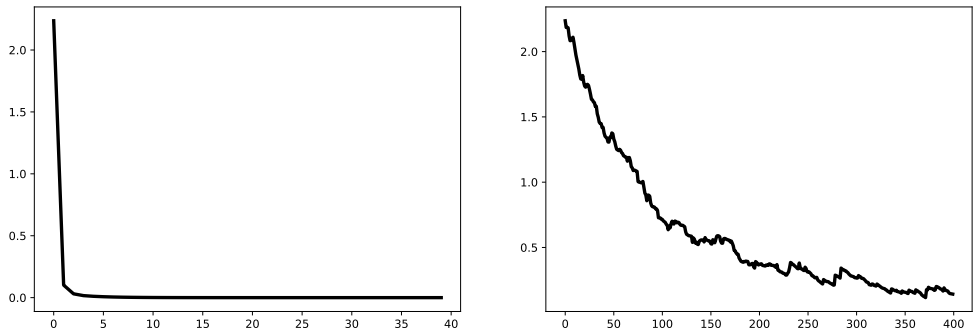


Figure 23.2: Convergence of gradient descent and SGD.

Notes

The origins of stochastic gradient descent go back to the work of Robbins and Monro in 1951 [21]. 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 [11] and [2]. A more general and systematic analysis of SGD that includes non-smooth objectives is given in [10]. These works also discuss general sampling techniques, not just uniform sampling.

Part III

Deep Learning

24

Neural Networks

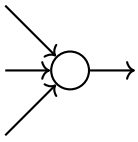
Neural Networks are a powerful class of functions with a wide range of applications in machine learning and data science. Originally introduced as simplified models of neurons in the brain, nowadays the biological motivation plays a less prominent role. Instead, the popularity of neural networks owes to their ability to combine generality with computational tractability: while neural networks can approximate most reasonable functions to arbitrary accuracy, their structure is still simple enough so that they can be trained efficiently by gradient descent.

Connectivity and Activation

We begin by considering 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 classifier

$$h_{\boldsymbol{w},b}(\boldsymbol{x}) = \mathbf{1}\{\boldsymbol{w}^T\boldsymbol{x} + b\} = \begin{cases} 1 & \boldsymbol{w}^T\boldsymbol{x} + b > 0 \\ 0 & \boldsymbol{w}^T\boldsymbol{x} + b \leq 0 \end{cases}$$

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**. One interpretation is that the node represents a neuron that *fires* if a certain linear combination of inputs, $\boldsymbol{w}^T\boldsymbol{x}$, exceeds a threshold $-b$. It is sometimes useful to approximate the indicator with a smooth function, and a convenient candidate is 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

$$\nabla g(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b)(1 - \sigma(\boldsymbol{w}^T\boldsymbol{x} + b)) \cdot \boldsymbol{w}.$$

Other activation functions that are commonly used are the hyperbolic tangent, $\tanh(x)$, and the **rectifiable linear unit** (**ReLU**), $\max\{x, 0\}$. Figure 24.1 illustrates these different activations functions.

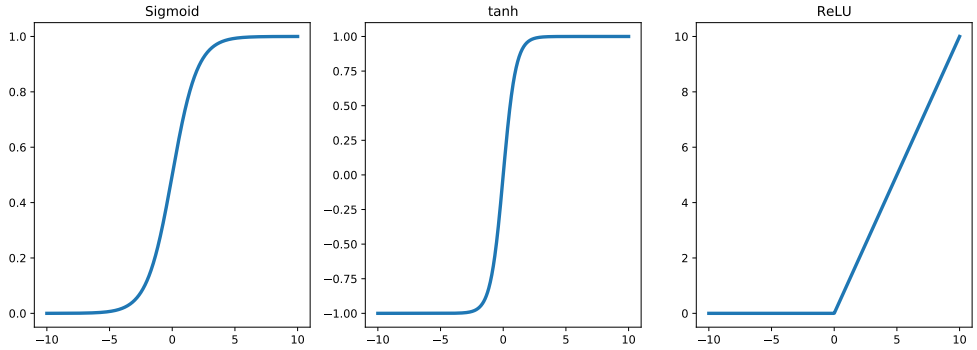


Figure 24.1: Activation functions

A **feedforward neural network** arises by combining various perceptrons by feeding the outputs of a series of perceptrons into a new perceptrons, see Figure 25.5.

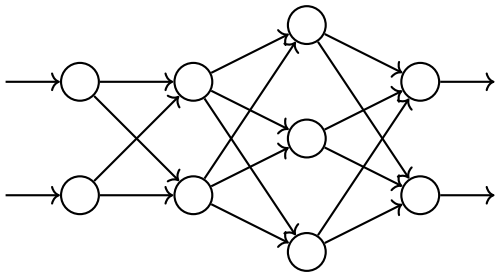


Figure 24.2: A fully connected neural network.

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 $\boldsymbol{b}^k \in \mathbb{R}^{d_k}$. One then applies the activation function componentwise, to 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

$$\begin{aligned} F^1(\boldsymbol{x}) &= \sigma(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \\ F^{k+1}(\boldsymbol{x}) &= \sigma(\boldsymbol{W}^{(k+1)}F^k(\boldsymbol{x}) + \boldsymbol{b}^{k+1}), \end{aligned}$$

and $\boldsymbol{W}^k \in \mathbb{R}^{d_i \times d_{k-1}}$, $\boldsymbol{b}^k \in \mathbb{R}^{d_k}$ for $1 \leq k \leq \ell$ (with $d_0 = d$). The layers between the input and output layer are called the **hidden layers**.

A neural network is therefore just a parametrized function that depends on a possibly large number of parameters. If we fix the **architecture**, that is, the number of layers ℓ and the number of nodes

$\boldsymbol{d} = (d_0, d_1, \ldots, d_\ell)$, where d_i represents the number of nodes in each layer, we get a a hypothesis class

$$\mathcal{H}_{\boldsymbol{d}} = \{F \colon \mathbb{R}^d \to \mathbb{R}^{d_\ell} \colon F \text{ is a NN with format } \boldsymbol{d}\}.$$

We can use neural networks for binary classification tasks, for example, by setting $d_\ell = 1$ (only one output layer), and declaring an output to be of class 1 if $F(\boldsymbol{x}) > 1/2$ and 0 otherwise. Alternatively, we can have two outputs and classify according to whether the first output is larger than the second. We can train the neural network on data $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, $i \in \{1, \ldots, n\}$, using our favourite loss function. Neural networks are particularly attractive for two reasons:

1. The class of neural networks is rich enough to capture almost all functions of interest (see Lecture 25).

2. The structure is simple enough to train using gradient descent, by a computational implementation of the chain rule known as **backpropagation**.

We discuss the computational aspect, backpropagation, first.

Backpropagation

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.

Given n data points $\{\boldsymbol{x}_i\}_{i=1}^n$ with corresponding outputs $\{\boldsymbol{y}_i\}_{i=1}^n$ and a smooth loss function L, the task is to minimize the function

$$f(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{n} \sum_{i=1}^{n} L(F^\ell(\boldsymbol{x}_i), \boldsymbol{y}_i).$$

Gradiend descent, or stochastic gradient descent, requires evaluating the gradient of a function of the form

$$f_i(\boldsymbol{W}, \boldsymbol{b}) = L(F^\ell(\boldsymbol{x}_i), \boldsymbol{y}_i).$$

We will describe a method for computing the gradient of such a function efficiently. In what follows, set $\boldsymbol{x} = \boldsymbol{x}_i$ and $\boldsymbol{y} = \boldsymbol{y}_i$. Also write $\boldsymbol{a}^0 = \boldsymbol{x}$, and for $k \in \{1, \ldots, \ell\}$,

$$\boldsymbol{z}^k = \boldsymbol{W}^k \boldsymbol{a}^{k-1} + \boldsymbol{b}^k, \quad \boldsymbol{a}^k = \sigma(\boldsymbol{z}^k). \tag{24.1}$$

In particular, $\boldsymbol{a}^\ell = F^\ell(\boldsymbol{x}_i)$ 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 $j \in \{1, \ldots, d_k\}$, 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 $j \in \{1, \ldots, d_k\}$. 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 24.1. *For a neural network with ℓ layers and $k \in \{1, \dots, \ell\}$, we have*

$$\frac{\partial C}{\partial w_{ij}^k} = \delta_i^k a_j^{k-1}, \qquad \frac{\partial C}{\partial b_i^k} = \delta_i^k \tag{24.2}$$

for $i, j \in \{1, \dots, d_k\}$. Moreover, the sensitivities δ_i^k can be computed as follows:

$$\boldsymbol{\delta}^\ell = \sigma'(\boldsymbol{z}^\ell) \circ \nabla_{\boldsymbol{a}^\ell} L(\boldsymbol{a}^\ell, \boldsymbol{y}), \qquad \boldsymbol{\delta}^k = \sigma'(\boldsymbol{z}^k) \circ \left(\boldsymbol{W}^{k+1}\right)^T \boldsymbol{\delta}^{k+1} \tag{24.3}$$

for $k \in \{1, \dots, \ell - 1\}$.

Proof. We begin by showing (24.2). For $\boldsymbol{\delta}^\ell$, note that by the chain rule, we have

$$\delta_i^\ell = \frac{\partial L(\boldsymbol{a}^\ell, \boldsymbol{y})}{\partial z_i^k} = \sum_{j=1}^{d_\ell} \frac{\partial L(\boldsymbol{a}^\ell, \boldsymbol{y})}{\partial a_j^\ell} \frac{\partial a_j^\ell}{\partial z_i^k} = \frac{\partial L(\boldsymbol{a}^k, \boldsymbol{y})}{\partial a_i^k} \cdot \sigma'(z_i^\ell).$$

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

$$\delta_i^k = \sum_{j=1}^{d_{k+1}} \delta_j^{k+1} \cdot w_{ji}^{k+1} \cdot \sigma'(z_i^k) = \sigma'(z_i^k) \cdot \left((\boldsymbol{W}^{k+1})^T \boldsymbol{\sigma}^{k+1}\right)_i.$$

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 (24.1) recursively. We can then compute the sensitivities δ_j^k using (24.3), and the partial derivatives of the loss function using (24.2), starting at layer ℓ. This way of computing the gradients is called **backpropagation**. We note that choosing the sigmoid δ 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 24.2. Consider the following setting with $n = 10$ points. We train a neural network with four layers and dimensions $\boldsymbol{d} = (d_0, d_1, d_2, d_3) = (2, 2, 3, 2)$ 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 24.3 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.

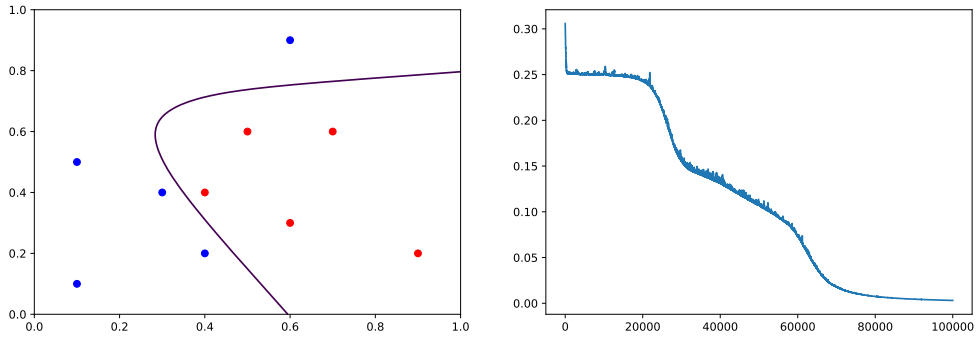


Figure 24.3: Training a neural network for classification and the error of stochastic gradient descent.

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 [22]. The idea of backpropagation was explicitly named in [23] and is closely related to **automatic differentiation**. Automatic differentiation has been discovered independently many times, and an interesting overview is given here: [12]. The content of this lecture is based on the excellent tutorial [13]. A comprehensive modern treatment of the subject can be found in the book [8].

25

Universal Approximation

In the last lecture we saw that neural networks can be trained efficiently using backpropagation. We now turn to studying the expressive power of neural networks.

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 $\|f - F\|_\infty = \sup_{x\in[0,1]} |f(x) - F(x)| < \epsilon$?

We begin by noting that any continuous function on $[0,1]$ can be approximated by step functions, as shown in Figure 25.1

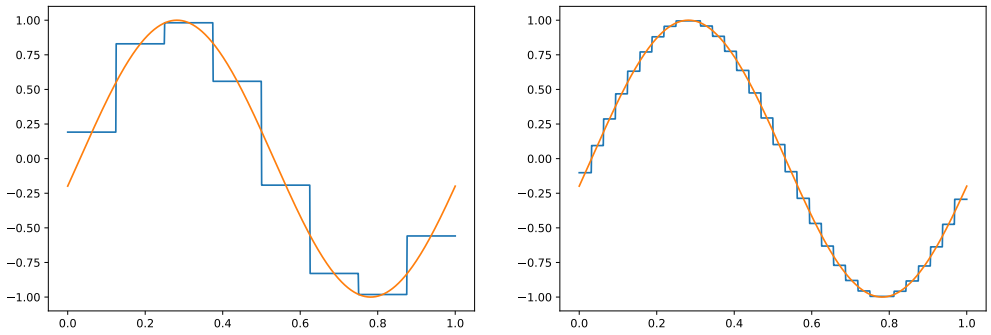


Figure 25.1: Approximation of a continuous function by a step function.

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 $s = -b/w$. Thus for a given weight w and location s, the function

$$g_{w,s}(x) = \sigma(w(x - s))$$

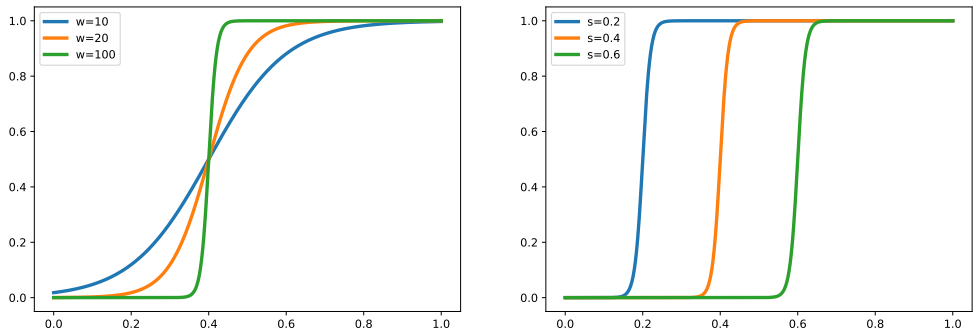


Figure 25.2: Sigmoid approximating a step function at various location and with various weights.

approximates the step function from 0 to 1 at location s. The approximation improves when increasing w, see Figure 25.2.

We can now form linear combinations of such functions to approximate step functions (piecewise constant functions) with multiple steps, as shown in Figure 25.3.

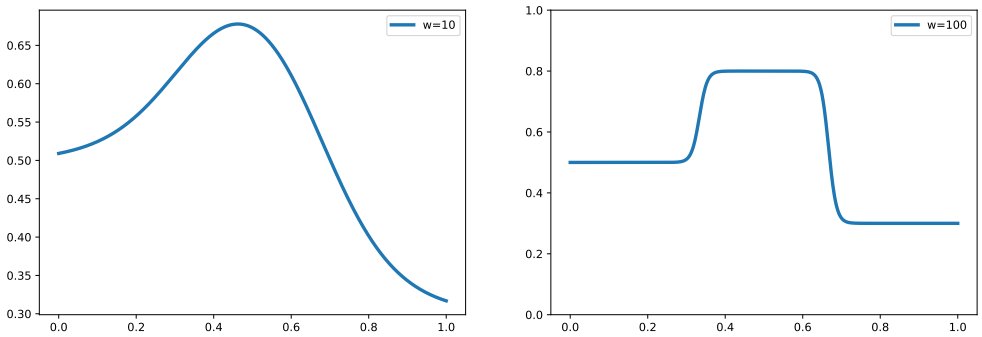


Figure 25.3: The function $0.5g_{w,-0.167} + 0.3g_{w,0.33} - 0.5g_{w,0.67}$ with weights $w = 10$ and $w = 100$.

We can therefore use the approximation of the step function by sigmoids and the approximation of an arbitrary continuous function on a closed interval by sigmoids to approximate any such function by sigmoids, see Figure 25.4.

We note that 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 of the approximation error at the left boundary, possibly by centering the first sigmoids at a negative value. All these considerations are not essential, and in the end one arrives at a function of the form

$$\sum_{i=1}^{m} \alpha_i \sigma_i(w_i x + b_i), \tag{25.1}$$

and 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.

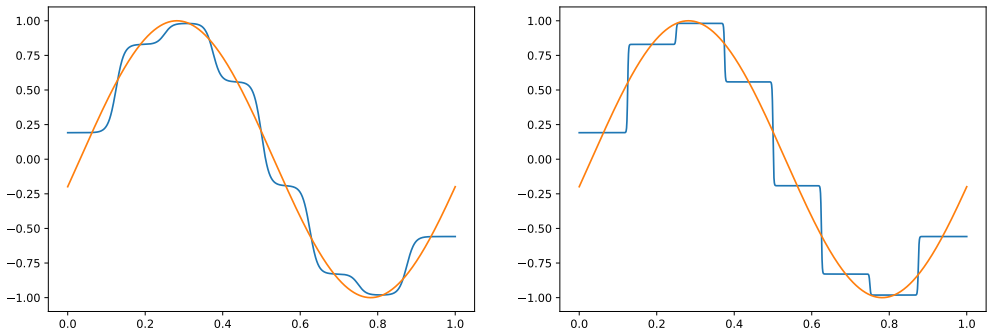


Figure 25.4: Approximation of a function by sigmoids with weights $w = 100$ and $w = 1000$.

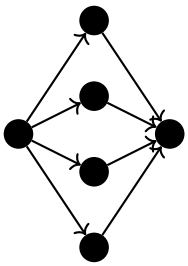


Figure 25.5: A neural network with one hidden layer.

Universal Approximation

Let $I^d = [0,1]^d \subset \mathbb{R}^d$ denote the unit cube and let $C(I^d)$ be the set of continuous functions on I^d with the norm $\|f\|_\infty = \sup_{\boldsymbol{x} \in I^d} |f(\boldsymbol{x})|$.

Theorem 25.1. *(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 $C(I^d)$. 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.

Theorem 25.2. *((Consequence of) Hahn-Banach) Let X be a normed vector space, $S \subset X$ a linear subspace, and $x_0 \in X$. Then $\boldsymbol{x}_0 \in \overline{S}$ if and only if for all linear functionals L on 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.

[1] If you took Measure Theory or Functional Analysis and ever wondered "what is this good for", here you go!

Theorem 25.3. *(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*. The proof relies on a Fourier Transform argument.

Lemma 25.4. *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 $C(I^d)$, 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 $\mu = 0$.

Proof of Theorem 25.1. 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 25.2, there exists a non-zero linear functional L on $C(I^d)$ such that $L|_{\overline{S}} = 0$. By the Riesz Reprentation Theorem 25.3, 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 25.4, 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 [6]. 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.

26

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**, which will be discussed in this lecture. 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 26.1. 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 wide-spread 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 26.2. 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

$$\boldsymbol{D} = \begin{pmatrix} -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \cdots & 0 \\ 0 & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -1 \end{pmatrix}.$$

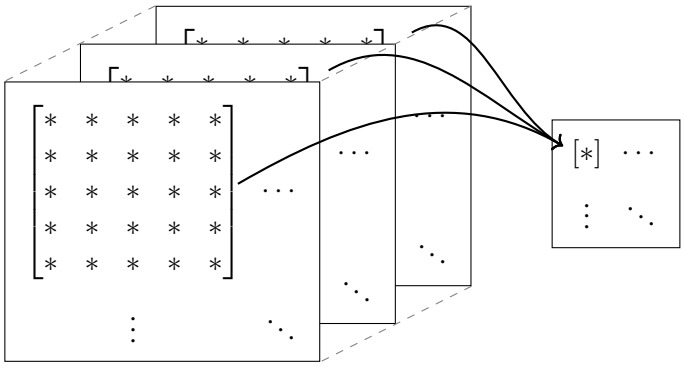
This matrix maps a vector $\boldsymbol{x}$ to the vector of differences $x_{i+1} - x_i$. This transformation detects *changes* in the vector, and a vector that is constant over a large range of coordinates is mapped to a **sparse** vector. Again, computing $\boldsymbol{y} = \boldsymbol{D}\boldsymbol{x}$ 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 26.1 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.

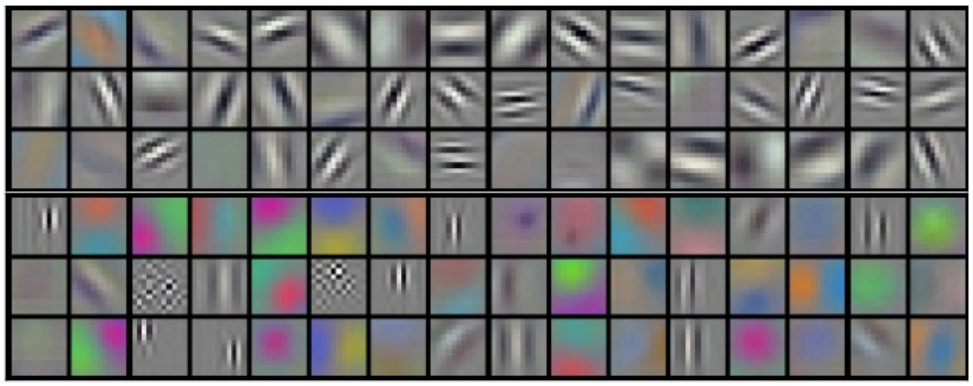


Figure 26.1: Filters for a convolutional layer from an image classification tasks, Krizhevsky et. al.

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 26.2. The following code shows an

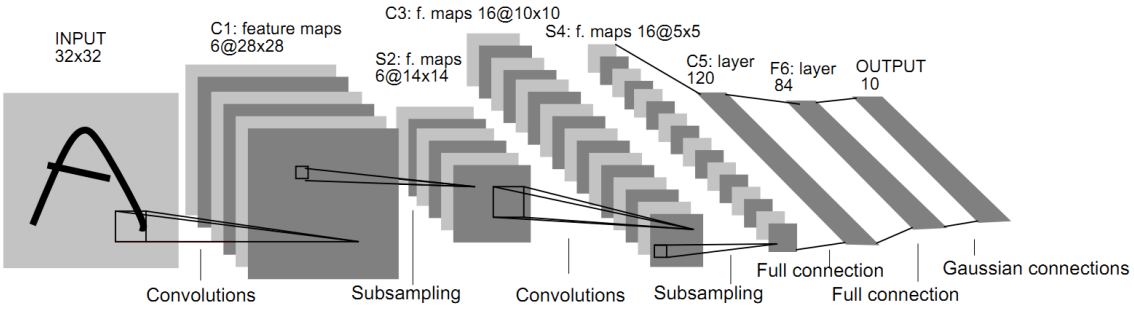


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Figure 26.2: 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 [8] and Chapter 7 of the survey [13] which also contains a worked through example in MATLAB. Two seminal papers in the field are [15] and [14].

27

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

$$\Delta(\boldsymbol{x}) = \inf_{\boldsymbol{r}} \{\|\boldsymbol{r}\|\colon h(\boldsymbol{x}+\boldsymbol{r}) \neq h(\boldsymbol{x})\}. \tag{27.1}$$

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 in Lecture 17 in the context of linear support vector machines (Figure 27.1).

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

$$\boldsymbol{r}^* = \operatorname{argmin} \frac{1}{2}\|\boldsymbol{r}\|^2 \qquad \text{subject to} \quad \boldsymbol{w}^T(\boldsymbol{x}+\boldsymbol{r}) + b = 0.$$

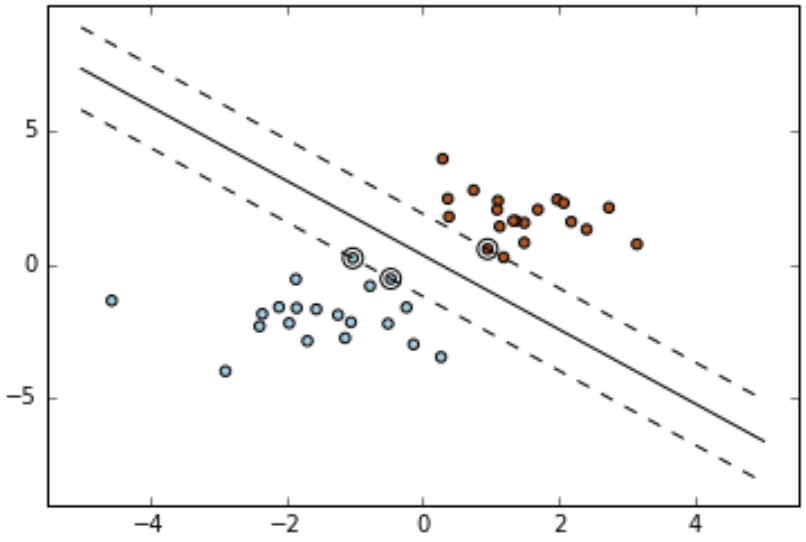


Figure 27.1: A hyperplane separating two sets of points with margin and support vectors.

We can obtain a closed form solution either geometrically (as in Lecture 17), 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

$$\nabla_{\boldsymbol{r}}\mathcal{L}(\boldsymbol{r}, \lambda) = \boldsymbol{r} + \lambda\boldsymbol{w} = \boldsymbol{0} \;\Rightarrow\; \boldsymbol{r}^* = -\lambda^*\boldsymbol{w}. \tag{27.2}$$

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 (27.2), 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 Lecture 17, 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 27.2).

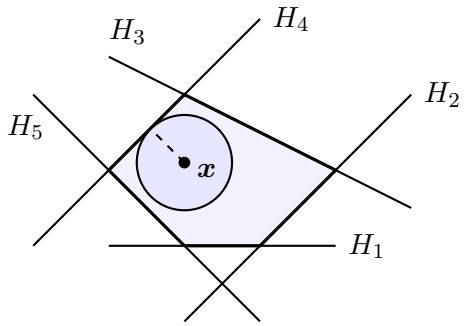


Figure 27.2: The **distance to misclassification** is the radius of the largest enclosed ball in a polyhedron P.

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

$$\hat{j} := \arg\min_j \frac{|g_j(\boldsymbol{x})|}{\|\nabla g_j(\boldsymbol{w})\|}, \quad \boldsymbol{r}^* = -\frac{g_{\hat{j}}(\boldsymbol{x})}{\|\nabla g_{\hat{j}}(\boldsymbol{x})\|^2} \cdot \nabla g_{\hat{j}}(\boldsymbol{x}). \tag{27.3}$$

The formulation (27.3) 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 27.3 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 27.3.

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 (27.1), 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 27.3: 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 27.1. *(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 [18] and [7]. Additional material can be found in [17].

28

Generative Adversarial Nets

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 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 conditional distribution $\rho_{X|Y=y}$, or more generally, the joint distribution $\rho_{X,Y}$ on the product of the input with the output space. 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

$$V(D,G) = \mathbb{E}_X[\log(D(X))] + \mathbb{E}_Z[\log(1 - D(G(Z)))], \tag{28.1}$$

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 (28.1) amounts to maximizing the function

$$\int_{\mathcal{X}} [\log(D(x))\rho_X(x) + \log(1 - D(x))\rho_G(x)] \, \mathrm{d}x.$$

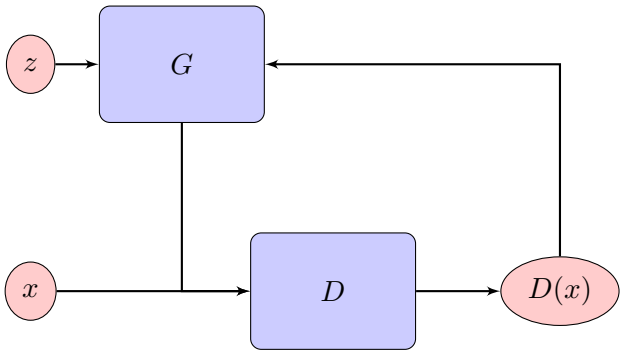


Figure 28.1: Structure of a GAN

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

$$D^*(x) = \frac{\rho_X(x)}{\rho_X(x) + \rho_G(x)} = 1 - \frac{\rho_G(x)}{\rho_X(x) + \rho_G(x)}. \tag{28.2}$$

Remark 28.1. 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 (see Lecture 3)

$$\begin{aligned}
\mathbb{E}[Y \mid X = x] &= \mathbb{P}\{Y = 1 \mid X = x\} \\
&= \frac{\mathbb{P}\{X = x \mid Y = 1\}\mathbb{P}\{Y = 1\}}{\mathbb{P}\{X = x \mid Y = 1\}\mathbb{P}\{Y = 1\} + \mathbb{P}\{X = x \mid Y = 0\}\mathbb{P}\{Y = 0\}} \\
&= \frac{\rho_X(x)}{\rho_X(x) + \rho_G(x)} = D^*(x).
\end{aligned}$$

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

$$-\frac{1}{n}\sum_{i=1}^{n} y_i \log(D(x_i)) + (1-y_i)\log(1 - D(x_i)), \tag{28.3}$$

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 28.2. 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 28.2. In this example, the function of the form (28.2) can be realized within the class of neural networks with

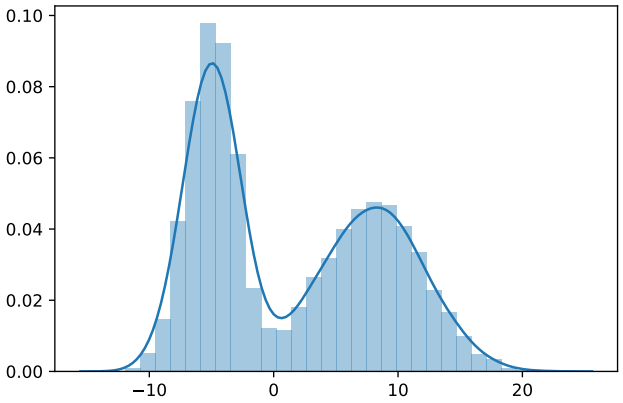


Figure 28.2: A Gaussian mixture distribution with histogram.

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 28.3. *The function $V(D^*, G)$, where D^* is defined as in (28.1), 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 (28.3). 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 28.3.

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 28.3: 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 28.4 shows the result of training such a GAN to generate realistic digits.

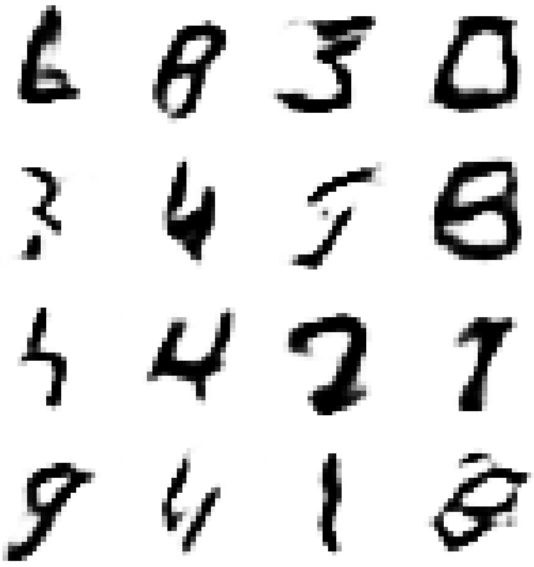


Figure 28.4: Digits generated by a GAN trained on MNIST digits for 2500 epochs with batch size $m = 128$.

Notes

Generative adversarial nets, in the way described here, were introduced by Ian Goodfellow et al in [9]. The GAN setup is also related to the concept of Predictability Minimization, introduced by Schmidhuber in the 1990s. For an overview, see [25]. 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/

Bibliography

[1] A. Beck. *First-Order Methods in Optimization*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, 2017.

[2] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[3] Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press, 2013.

[4] Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[5] Felipe Cucker and Ding Xuan Zhou. *Learning theory: an approximation theory viewpoint*, volume 24. Cambridge University Press, 2007.

[6] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[7] Alhussein Fawzi, Hamza Fawzi, and Omar Fawzi. Adversarial vulnerability for any classifier. In *Advances in Neural Information Processing Systems*, pages 1178–1187, 2018.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[9] 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.

[10] 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.

[11] 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.

[12] Andreas Griewank. Who invented the reverse mode of differentiation. *Documenta Mathematica, Extra Volume ISMP*, pages 389–400, 2012.

[13] Catherine F Higham and Desmond J Higham. Deep learning: An introduction for applied mathematicians. *SIAM Review*, 61(4):860–891, 2019.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] Yurii Nesterov. *Lectures on convex optimization*, volume 137. Springer, 2018.

[20] Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

[21] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

[22] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[23] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[24] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, pages 71–105, 1959.

[25] Juergen Schmidhuber. Unsupervised minimax: Adversarial curiosity, generative adversarial networks, and predictability minimization. *arXiv preprint arXiv:1906.04493*, 2019.

[26] Weijie Su, Stephen Boyd, and Emmanuel Candes. A differential equation for modeling nesterov's accelerated gradient method: Theory and insights. In *Advances in Neural Information Processing Systems*, pages 2510–2518, 2014.

[27] Terence Tao. *Topics in Random Matrix Theory*. Graduate studies in mathematics. American Mathematical Society, 2012.

[28] Lloyd N Trefethen. *Approximation theory and approximation practice*, volume 128. Siam, 2013.

[29] Vladimir Vapnik. *The nature of statistical learning theory*. Springer, 2013.

[30] Roman Vershynin. *High-dimensional probability: An introduction with applications in data science*, volume 47. Cambridge University Press, 2018.

[31] 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.