MTH793P – Advanced Machine Learning

Martin Benning

Last updated on: March 16, 2021

Lecture Notes
Semester B 2021

Contents

The lecture notes are under constant re-development and will likely contain errors and mistakes. I very much appreciate the finding and reporting of those (to m.benning@qmul.ac.uk). Thanks!

Chapter 1

Mathematical preliminaries

In this first chapter we briefly revise basic mathematical preliminaries that we are going to use throughout this module. Those preliminaries span topics ranging from linear algebra over calculus to basic probability and statistics.

1.1 Linear algebra

Throughout this module, we will extensively deal with vectors and matrices. The term *vector* refers to a an object that both has a magnitude and a direction. We will usually write vectors $x \in \mathbb{R}^d$ (or $x \in \mathbb{R}^{d\times 1}$) as

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} ,$$

which we generally also refer to as *column-vectors.* Here each individual x_j, for $j \in \{1, \ldots, d\}$, is a real-valued scalar, i.e. $x_j \in \mathbb{R}$ for all $j \in \{1, \ldots, d\}$. In contrast to a column-vector, we can also consider *row-vectors* $x^\top \in \mathbb{R}^{1\times d}$, i.e.

$$x^\top = \begin{pmatrix} x_1 & x_2 & \ldots & x_d \end{pmatrix} .$$

Both are special cases of a *matrix*, which is a rectangular array of scalars. We write a matrix $X \in \mathbb{R}^{d_1\times d_2}$ of size $d_1 \times d_2$ as

$$X = \begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1d_2} \\ x_{21} & x_{22} & \ldots & x_{2d_2} \\ \vdots & & \ddots & \vdots \\ x_{d_11} & x_{d_12} & \ldots & x_{d_1d_2} \end{pmatrix} .$$

The *transpose* $X^\top \in \mathbb{R}^{d_2\times d_1}$ of matrix $X \in \mathbb{R}^{d_1\times d_2}$ is defined by interchanging the rows and columns, i.e.

$$X^\top = \begin{pmatrix} x_{11} & x_{21} & \ldots & x_{d_11} \\ x_{12} & x_{22} & \ldots & x_{d_12} \\ \vdots & & \ddots & \vdots \\ x_{1d_2} & x_{2d_2} & \ldots & x_{d_1d_2} \end{pmatrix} .$$

Note that we can immediately conclude $(X^\top)^\top = X$, and that a row-vector is simply the transpose of a column-vector (and vice versa), which in hindsight explains our use of the notation $x^\top$ for the row-vector. Two very important concepts in the context of matrices are the *range* and the *kernel* (or nullspace) of a matrix. The range $\operatorname{ran}(X)$ of a matrix X is the set of all vectors that can be expressed in terms of X, i.e.

$$\operatorname{ran}(X) := \{Xz \mid z \in \mathbb{R}^n\} \, .$$

The kernel $\ker(X)$ of a matrix X is the set of all vectors that X maps onto the zero vector, i.e.

$$\ker(X) := \{z \in \mathbb{R}^n \mid Xz = 0\} \, .$$

In the following, we want to recall the two main products that are relevant for this module. The most important vector-vector product is the so-called *dot product* or *inner product* of two vectors $x, y \in \mathbb{R}^d$ of identical dimension, defined as

$$\langle x, y \rangle := \sum_{j=1}^{d} x_j y_j \, .$$

Note that we simply multiply the individual coordinates of the vectors x and y and sum over all products. Another common notation for the dot product is $x \cdot y$.

The *matrix-vector product* for a matrix $X \in \mathbb{R}^{d_1 \times d_2}$ and a column-vector $y \in \mathbb{R}^{d_2 \times 1}$ is defined as

$$(Xy)_i := \sum_{j=1}^{d_2} x_{ij} y_j \qquad \text{for all} \quad i \in \{1, \ldots, d_1\} \, .$$

We denote the resulting (column) vector simply as $Xy \in \mathbb{R}^{d_1 \times 1}$. In identical fashion, we can define a *matrix-matrix product* for matrices $X \in \mathbb{R}^{d_1 \times d_2}$ and $Y \in \mathbb{R}^{d_2 \times d_3}$ as

$$(XY)_{ij} := \sum_{l=1}^{d_2} x_{il} y_{lj} \quad \text{for all} \quad i \in \{1, \ldots, d_1\} \text{ and } j \in \{1, \ldots, d_3\} \, .$$

Again, we denote the resulting matrix simply as $XY \in \mathbb{R}^{d_1 \times d_3}$. We want to emphasise that the inner product can be viewed as a special case of the matrix-matrix product, if we consider the matrix-matrix product of a row-vector and a column-vector, i.e. for $x, y \in \mathbb{R}^d$ we have

$$\langle x, y \rangle = x^\top y \, .$$

Having defined an inner product, it is straight-forward to define the *Euclidean norm* of a vector. The Euclidean norm $\| \cdot \| : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ of a vector $x \in \mathbb{R}^d$ is defined as

$$\|x\| := \sqrt{\langle x, x \rangle} = \sqrt{\sum_{j=1}^{d} x_j^2} \, .$$

Please note that sometimes we will denote the Euclidean norm by $\| \cdot \|_2$ rather than $\| \cdot \|$ in order to distinguish them from other norms. Another module-relevant vector norm is the *one-norm* $\| \cdot \|_1 : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$, which is defined as

$$\|x\|_1 := \sum_{j=1}^{d} |x_j| \, .$$

There are numerous ways of defining a *matrix norm*, but the only two relevant matrix norms for this module are the *Frobenius norm*, i.e.

$$\|X\|_{\text{Fro}} := \sqrt{\sum_{i=1}^{d_1}\sum_{j=1}^{d_2} x_{ij}^2}\,,$$

and the standard matrix norm

$$\|X\| := \sup_{\|y\|\leq 1} \|Xy\| = \sup_{y\neq 0} \frac{\|Xy\|}{\|y\|}\,.$$

Here $\|\cdot\|$ denotes the Euclidean norm, and sup is the supremum. Before we move on to recall basic concepts of calculus, we want to briefly address the concepts of *eigenvalues* and *eigenvectors* of square matrices. An eigenvalue λ and an eigenvector w_λ are characterised by the equation

$$Xw_\lambda = \lambda w_\lambda\,. \tag{1.1}$$

This means that w_λ is invariant under matrix multiplication albeit a scaling with factor λ. If we take an inner product of (1.1) with w_λ, we immediately observe that an eigenvalue λ takes on the value

$$\lambda = \frac{\langle Xw_\lambda, w_\lambda\rangle}{\|w_\lambda\|^2}\,.$$

Assuming that all eigenvectors are normalised, i.e. $\|w_\lambda\| = 1$, we conclude $\lambda = \langle Xw_\lambda, w_\lambda\rangle$. For eigenvalues σ^2 and eigenvectors v_σ of $X^\top X$, i.e.

$$X^\top X v_\sigma = \sigma^2 v_\sigma\,,$$

we observe $\sigma = \|Xv_\sigma\|/\|v_\sigma\| \geq 0$. This implies that the matrix norm equals the square-root of the largest eigenvalue of $X^\top X$, i.e.

$$\|X\| = \sup_{v\neq 0}\frac{\|Xv\|}{\|v\|} = \sup\left\{\sigma\in\mathbb{R}_{\geq 0}\,\middle|\, X^\top X v = \sigma^2 v\right\}\,.$$

The square-root of the eigenvalues of $X^\top X$, i.e. σ, are known as *singular values* of X. More information and properties of singular values can be found here. The eigenvectors are known as the *right singular vectors* of X. In identical fashion we can derive eigenvectors u_σ of $XX^\top$, which are known as the *left singular vectors* of X. In this module, the most important properties are that Xw, $X^\top y$, $X^\top Xw$ and $(X^\top X)^{-1}b$ (if $(X^\top X)^{-1}$ exists) can be expressed in terms of the *singular value decomposition* of X, i.e.

$$X = U\Sigma V^\top\,, \tag{1.2}$$

where $U\in\mathbb{R}^{s\times\min(s,d)}$ and $V\in\mathbb{R}^{d\times\min(s,d)}$ are the matrices that contain all singular vectors $\{u_{\sigma_i}\}_{i=1}^{\min(s,d)}$ and $\{v_{\sigma_i}\}_{i=1}^{\min(s,d)}$ as their columns, while $\Sigma\in\mathbb{R}^{\min(s,d)\times\min(s,d)}$ is the diagonal matrix whose diagonal contains all singular vectors $\sigma_1\geq\sigma_2\geq\ldots\geq\sigma_{\min(s,d)}$. With the help of Equation (3.16), we can express Xw, $X^\top y$, $X^\top Xw$ and $(X^\top X)^{-1}b$ as

$$Xw = \sum_{j=1}^{\min(s,d)}\sigma_j u_j\langle v_j, w\rangle\,,\qquad X^\top y = \sum_{j=1}^{\min(s,d)}\sigma_j v_j\langle u_j, y\rangle\,,$$

and

$$X^\top X w = \sum_{j=1}^{\min(s,d)} \sigma_j^2 v_j \langle v_j, w\rangle \,, \qquad (X^\top X)^{-1} b = \sum_{j=1}^{\min(s,d)} \sigma_j^{-2} v_j \langle v_j, b\rangle \,.$$

For more information on singular value decompositions we refer to this page.

1.2 Calculus

We will require the computation of (partial) derivatives, gradients and Hessian matrices during this module. Suppose we are given a continuously differentiable function $f : \mathbb{R} \to \mathbb{R}$, then its *derivative* $f' : \mathbb{R} \to \mathbb{R}$ is defined as

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

The notation f' is known as Lagrange's notation and is the most common notation for derivatives. Another useful notation is Leibniz's notation $\frac{df}{dx}$ that emphasises that the derivative of f with respect to (w.r.t.) x is taken. Many useful differentiation rules exist. If you require a little refresher you can recall many of those rules here. For continuously differentiable functions $f : \mathbb{R}^d \to \mathbb{R}$ in multiple variables we can define *partial derivatives* as

$$\frac{\partial f}{\partial x_j}(x_1, \ldots, x_d) := \lim_{h \to 0} \frac{f(x_1, \ldots, x_{j-1}, x_j + h, x_{j+1}, \ldots, x_d) - f(x_1, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_d)}{h} \,.$$

If we compute the partial derivatives w.r.t. all arguments $x_1, \ldots, x_d$ and store them in a column-vector, we obtain the *gradient* $\nabla f : \mathbb{R}^d \to \mathbb{R}^d$ of a function $f : \mathbb{R}^d \to \mathbb{R}$, i.e.

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(x) & \frac{\partial f}{\partial x_2}(x) & \cdots & \frac{\partial f}{\partial x_d}(x) \end{pmatrix} \,.$$

Here $x = (x_1, \ldots, x_d)$ is a short-hand vector notation for all arguments $x_1, \ldots, x_d$. For function $f : \mathbb{R}^{d_2} \to \mathbb{R}^{d_1}$ with multiple outputs we can define the *Jacobian matrix* $J_f : \mathbb{R}^{d_2} \to \mathbb{R}^{d_1 \times d_2}$ as

$$J_f(x) := \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_{d_2}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{d_1}}{\partial x_1} & \cdots & \frac{\partial f_{d_1}}{\partial x_{d_2}} \end{pmatrix} \,.$$

For many differentiation rules there exist higher-dimensional counterparts. Of particular interest to us is the multi-dimensional chain-rule, which for a composition $f \circ g$ of functions $f : \mathbb{R}^{d_2} \to \mathbb{R}^{d_1}$ and $g : \mathbb{R}^{d_3} \to \mathbb{R}^{d_2}$ reads

$$J_{f \circ g}(x) = J_f(g(x)) J_g(x) \,.$$

In particular, for functions $f : \mathbb{R}^{d_1} \to \mathbb{R}$ and $g : \mathbb{R}^{d_2} \to \mathbb{R}^{d_1}$ we observe

$$\nabla (f \circ g)(x) = \nabla f(g(x))^\top J_g(x) \,.$$

We conclude this section with second-order derivatives for multi-variable functions. A *second partial derivative* is the application of the partial derivative to a partial derivative, assuming that such a partial derivative exists, i.e.

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(x) := \frac{\partial f}{\partial x_i}\frac{\partial f}{\partial x_j}(x)\,, \qquad \text{for} \qquad i, j \in \{1, \ldots, d\}\,,$$

for $x = (x_1, \ldots, x_d)$ and a function $f : \mathbb{R}^d \to \mathbb{R}$. If $i = j$, we simply write $\partial^2 f/\partial x_i^2$. Based on this concept, one can define the *Hessian matrix* $H_f : \mathbb{R}^d \to \mathbb{R}^{d\times d}$ of second-order partial derivatives as

$$H_f(x) := \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{pmatrix},$$

for $x = (x_1, \ldots, x_d)$, assuming that all second-order partial derivatives exist. Note that gradient, Jacobian and Hessian are connected via

$$H_f(x) = J_{\nabla f}(x)\,.$$

This concludes our section on calculus. In the next and final section of the mathematical preliminaries, we revisit some basic rules and notations of probability & statistics.

1.3 Probability & statistics

The *expected value*, or *expectation*, of a random variable X with finite outcomes $\{x_i\}_{i=1}^s$ with probabilities $\{\rho_i\}_{i=1}^s$ is defined as

$$\mathbb{E}_i[x_i] := \sum_{i=1}^{s} x_i \rho_i\,. \tag{1.3}$$

Since probabilities are non-negative and sum up to one, the expected value (1.3) is a *weighted average*. In case all outcomes are equiprobable, i.e. $\rho_i = 1/s$ for all $i \in \{1, \ldots, s\}$, the expected value is the normal *average*, or *arithmetic mean*.
In the absolutely continuous case, the expectation is defined as

$$\mathbb{E}_x[x] = \int_{\mathbb{R}} x\,\rho(x)\,dx\,,$$

assuming that the cumulative distribution function of its underlying random variable X admits a probability density function (PDF) ρ and that the above integral exists.
Please note that expectations can also be computed for measurable functions f, i.e.

$$\mathbb{E}_x[f(x)] := \int_{\mathbb{R}} f(x)\,\rho(x)\,dx\,. \tag{1.4}$$

For the example of an indicator function over the one-dimensional interval $[a, b]$, i.e.

$$\iota_{[a,b]}(x) = \begin{cases} 1 & x \in [a, b] \\ 0 & x \notin [a, b] \end{cases},$$

we instantly observe

$$\mathbb{E}_x\left[\iota_{[a,b]}(x)\right] = \int_{\mathbb{R}} \iota_{[a,b]}(x)\,\rho(x)\,dx = \int_a^b \rho(x)\,dx = P(a \leq X \leq b)\,.$$

The right-hand-side denotes the probability that a random variable takes a value that lies in the interval $[a,b]$.
The *variance* of a random variable X is defined as

$$\begin{aligned}\mathrm{Var}_x[x] &:= \mathbb{E}_x\left[(x - \mathbb{E}_x[x])^2\right]\,, \\ &= \mathbb{E}_x[x^2] - \mathbb{E}_x[x]^2\,.\end{aligned}$$

The square-root of the variance, i.e. $\sigma_x := \sqrt{\mathrm{Var}_x[x]}$, is known as the *standard deviation.*

Note that expectations and variances can easily be extended to multi-variable functions. If we have functions f in two (absolutely continuous) random variables X and Y from a joint cumulative distribution with underlying PDF ρ for example, we simply write

$$\mathbb{E}_{x,y}[f(x,y)] = \int_{\mathbb{R}}\int_{\mathbb{R}} f(x,y)\,\rho(x,y)\,dx\,dy\,.$$

Note that two random variables X and Y are said to be *independent* if their probabilities or their PDFs factor, i.e.

$$\rho(x,y) = \rho_X(x)\rho_Y(y)\,.$$

For an arbitrary number n of random variables $\{X_i\}_{i=1}^n$, we have

$$\rho(x_1,\ldots,x_n) = \prod_{i=1}^n \rho_{X_i}(x_i)\,.$$

A collection of random variables is *independent and identically distributed (iid)* if each random variable has the same probability distribution (and thus the same PDF) and if all random variables are independent. Hence, we can write the joint PDF as

$$\rho(x_1,\ldots,x_n) = \prod_{i=1}^n \tilde{\rho}(x_i)\,,$$

where $\tilde{\rho}$ denotes the PDF of the underlying probability distribution. We conclude this chapter with the definitions of the *likelihood function* and the *posterior probability.* Let X be a (continuous) random variable with probability density function $\rho_\theta(x)$ that depends on parameters θ. Then the function

$$\rho(x\,|\,\theta) := \rho_\theta(x)$$

is the *likelihood function* of θ, given the outcome x of X, or the probability of outcome x for the parameter value θ. The *posterior probability* on the other hand is the probability of the parameters θ given the outcome x of X, i.e. $\rho(\theta\,|\,x)$. Given the *prior* probability distribution function $\rho(\theta)$ for the parameters θ, both are connected via *Bayes' rule* or *Bayes' theorem* for PDFs:

$$\rho(\theta\,|\,x) = \frac{\rho(x\,|\,\theta)\rho(\theta)}{\rho(x)}\,, \tag{1.5}$$

named after Reverend Thomas Bayes. With this we conclude this chapter on mathematical preliminaries and begin our introduction of supervised machine learning.

Chapter 2

Supervised learning

In this module we extensively study numerous aspects of supervised machine learning. The two predominant problems in supervised machine learning are regression and classification problems. Beginning with a statistical motivation for supervised linear regression problems, we spent the first half of this module on regression problems, before moving on to classification problems.

2.1 Statistical motivation

In supervised machine learning, the goal is to find a function mapping $f : \mathbb{R}^n \to \mathbb{R}^m$ that approximately maps a collection of s known input arguments $\{x_i\}_{i=1}^s$, with $x_i \in \mathbb{R}^n$ for all $i \in \{1, \ldots, s\}$, onto a collection of s output elements $\{y_i\}_{i=1}^s$, with $y_i \in \mathbb{R}^m$ for all $i \in \{1, \ldots, s\}$, i.e.

$$f(x_i) \approx y_i \,, \qquad \forall i \in \{1, \ldots, s\} \,. \tag{2.1}$$

The name 'supervised' stems from the fact that a collection of outputs $\{y_i\}_{i=1}^s$ needs to be known in advance. In the following, we want to specify what we mean by 'approximately', and why it is not necessarily desirable to have a strict equality in (2.1).

In Figure 2.1 we see a collection of data points $\{(x_i, y_i)\}_{i=1}^s$ that represent the weight- and height-information for s individuals, so the goal is to find a mapping that approximately returns the height-information when given the weight-information. We observe that the data points seem to follow a linear trend, but with substantial and presumably random deviations from this line. Instead of seeking a function that maps every x_i onto each corresponding y_i, one could rather try to find a line that has minimal distance to each point (x_i, y_i). What distance, you may ask? If we collect the error in prediction for each sample we obtain a plot like the one in Figure 2.1. The error seems to follow a normal distribution, which is why it makes sense to model the deviation as normal-distributed independent random variables with mean zero and variance σ^2, i.e. for

$$\varepsilon_i := y_i - f(x_i) \,, \qquad \forall i \in \{1, \ldots, s\} \,,$$

we know that every ε_i is distributed according to

$$\rho(\varepsilon_i | 0, \sigma) = \mathcal{N}(\varepsilon_i | 0, \sigma) := \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\varepsilon_i^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(f(x_i) - y_i)^2}{2\sigma^2}} \,.$$

Assuming that all ε_i are independent random variables, the joint probability density reads as

$$\rho(\varepsilon | 0, \sigma) = \prod_{i=1}^s \rho(\varepsilon_i | 0, \sigma) = (2\pi\sigma^2)^{-\frac{s}{2}} \prod_{i=1}^s e^{-\frac{(f(x_i) - y_i)^2}{2\sigma^2}} \,.$$

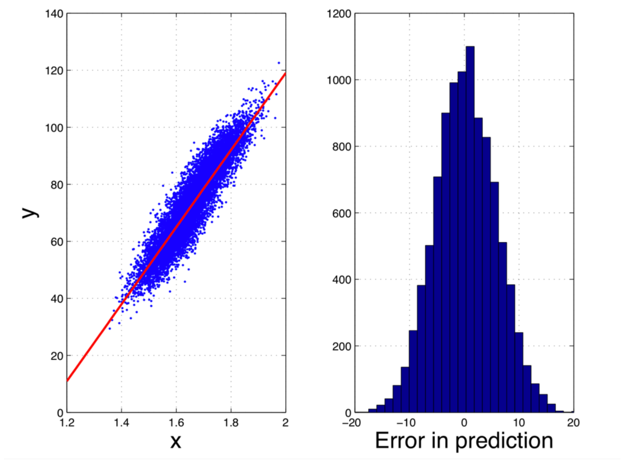


Figure 2.1: A simple linear regression example for data points with weights- (x-axis) and height-information (y-axis). The figure on the left shows a line fitted to the given data points that has minimal mean-squared error w.r.t. all data points in the sense that it minimises (2.3). The plot on the right-hand-side shows the the mean-squared error of the individual samples w.r.t. the fitted line.

If we assume that our model f is parametrised with parameters w, which in the following we denote as f_w, it seems wise to choose those parameters such that the likelihood for the joint probability distribution is maximised, i.e. we look for parameters $\hat{w}$ that satisfy

$$\begin{aligned}
\hat{w} &= \arg\max_w \left\{ \prod_{i=1}^s \rho(\varepsilon_i | 0, \sigma) \right\} , \\
&= \arg\max_w \left\{ (2\pi\sigma^2)^{-\frac{s}{2}} \prod_{i=1}^s e^{-\frac{(f_w(x_i)-y_i)^2}{2\sigma^2}} \right\} .
\end{aligned}$$

Due to the monotonicity of the natural logarithm, we can alternatively estimate $\hat{w}$ by estimating the minimiser of the negative log-likelihood, i.e.

$$\begin{aligned}
\hat{w} &= \arg\min_w \left\{ -\log \left(\prod_{i=1}^s \rho(\varepsilon_i | 0, \sigma) \right) \right\} , \\
&= \arg\min_w \left\{ -\sum_{i=1}^s \log\left(\rho(\varepsilon_i | 0, \sigma) \right) \right\} , \\
&= \arg\min_w \left\{ \frac{s}{2} \log(2\pi\sigma^2) + \frac{1}{2\sigma^2} \sum_{i=1}^s (f_w(x_i) - y_i)^2 \right\} , \\
&= \arg\min_w \left\{ \frac{1}{2} \sum_{i=1}^s (f_w(x_i) - y_i)^2 \right\} , \\
&= \arg\min_w \left\{ \frac{1}{2s} \sum_{i=1}^s (f_w(x_i) - y_i)^2 \right\} . \qquad (2.2)
\end{aligned}$$

Hence, optimal parameters $\hat{w}$ have to be chosen in order to minimise the squared Euclidean norm w.r.t. each sample $\{(x_i, y_i)\}_{i=1}^s$. The function

$$\mathrm{MSE}(w) := \frac{1}{2s} \sum_{i=1}^s (f_w(x_i) - y_i)^2 \tag{2.3}$$

is known as the *mean-squared error* (MSE) and minimising it is known as the *method of least-squares*. In the following we want to address the question of how to parametrise f_w and start with a linear model.

2.2 Linear & polynomial regression

In linear regression, we parametrise our model f_w in terms of a linear transformation, i.e. for a weight $w \in \mathbb{R}^{d+1}$ and $f_w : \mathbb{R}^d \to \mathbb{R}$ we choose

$$f_w(x) := \langle x, w \rangle = \sum_{j=0}^d x_j w_j \,, \tag{2.4}$$

where we define $x_0 = 1$ in order to allow scalar translations w_0. Suppose we are given s pairs of input/output samples $\{(x_i, y_i)\}_{i=1}^s$, we can estimate a weight w following (2.2) by minimising the least-squares error with respect to all samples, i.e.

$$w_t = \operatorname*{arg\,min}_{w \in \mathbb{R}^{d+1}} \left\{ \frac{1}{2s} \sum_{i=1}^s |\langle x_i, w \rangle - y_i|^2 \right\} . \tag{2.5}$$

An alternative way of writing (2.5) is in terms of matrix multiplication and Euclidean norm as

$$w_t = \operatorname*{arg\,min}_{w \in \mathbb{R}^{d+1}} \left\{ \frac{1}{2s} \|Xw - y\|^2 \right\} ,$$

for a matrix

$$X := \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1d} \\ 1 & x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_{s1} & x_{s2} & \cdots & x_{sd} \end{pmatrix} ,$$

and a vector $y := \begin{pmatrix} y_1 & y_2 & \dots & y_s \end{pmatrix}$. Later in this module we will verify that the unique solution to (2.5) are weights w_t that satisfy

$$\nabla \mathrm{MSE}(w_t) = 0 \,. \tag{2.6}$$

It will be left as a coursework exercise to show that the solution of (2.6) is the linear system of equations of the form

$$\sum_{i=1}^s \begin{pmatrix} x_{i0}^2 & x_{i0}x_{i1} & \cdots & x_{i0}x_{id} \\ x_{i1}x_{i0} & x_{i1}^2 & \cdots & x_{i1}x_{id} \\ \vdots & \vdots & \ddots & \vdots \\ x_{id}x_{i0} & x_{id}x_{i1} & \cdots & x_{id}^2 \end{pmatrix} \hat{w} = \sum_{i=1}^s y_i x_i \,,$$

$$\Leftrightarrow \quad \begin{pmatrix} \|x_0\|^2 & \langle x_0, x_1 \rangle & \cdots & \langle x_0, x_d \rangle \\ \langle x_1, x_0 \rangle & \|x_1\|^2 & \cdots & \langle x_1, x_d \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle x_d, x_0 \rangle & \langle x_d, x_1 \rangle & \cdots & \|x_d\|^2 \end{pmatrix} \hat{w} = \sum_{i=1}^s y_i x_i \,. \tag{2.7}$$

Note that here the inner product is with respect to the sample-dimension, i.e. $\langle x_k, x_l\rangle = \sum_{i=1}^s x_{ik}x_{il}$. We want to mention that we can easily extend (2.4) to functions with m-dimensional output via

$$f_W(x) := \begin{pmatrix} \langle x, w_1\rangle \\ \langle x, w_2\rangle \\ \vdots \\ \langle x, w_m\rangle \end{pmatrix} ,$$

or equivalently $f_W(x) = W^\top x$ in matrix-multiplication form, for the matrix $W \in \mathbb{R}^{(d+1)\times m}$ defined as

$$W := \begin{pmatrix} | & | & & | \\ w_1 & w_2 & \dots & w_m \\ | & | & & | \end{pmatrix} .$$

In many applications, the underlying true function f is nonlinear in its argument and linear models are insufficient in terms of their systematic bias. A simple class of nonlinear model functions are polynomials.

2.2.1 Polynomial regression

In polynomial regression the goal is to fit polynomials of degree d to the data samples. This can easily be achieved by equipping (2.4) with a vector of polynomials of degree d, i.e.

$$f_w(x) := \langle \phi(x), w\rangle = \sum_{j=0}^d \phi(x)_j w_j \,, \tag{2.8}$$

with $\phi : \mathbb{R} \to \mathbb{R}^{d+1}$ defined as

$$\phi(x) := \begin{pmatrix} 1 & x & x^2 & \dots & x^d \end{pmatrix}^\top .$$

Note that (2.8) is nonlinear in the argument x, but linear in the weight vector w. Suppose we have s pairs $\{(x_j, y_j)\}_{j=1}^s$ as usual, then for each x_i we have $\phi(x_i) \in \mathbb{R}^{d+1}$. We can, therefore, define the short-hand notations

$$\Phi(X) := \begin{pmatrix} \phi(x_1)^\top \\ \phi(x_2)^\top \\ \vdots \\ \phi(x_s)^\top \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^d \\ 1 & x_2 & x_2^2 & \dots & x_2^d \\ \vdots & & & & \\ 1 & x_s & x_s^2 & \dots & x_s^d \end{pmatrix} \in \mathbb{R}^{s\times(d+1)} , \qquad \text{and} \qquad y := \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_s \end{pmatrix} \in \mathbb{R}^s .$$

As in the linear regression case we can find optimal weights by solving the least-squares problem

$$w_t = \operatorname*{arg\,min}_{w\in\mathbb{R}^{d+1}} \left\{ \frac{1}{2s} \|\Phi(X)w - y\|^2 \right\} , \tag{2.9}$$

where $\|\cdot\|$ denotes the Euclidean norm, or

$$W_t = \operatorname*{arg\,min}_{W\in\mathbb{R}^{(d+1)\times m}} \left\{ \frac{1}{2s} \|\Phi(X)W - Y\|_{\text{Fro}}^2 \right\}$$

in case we want to parametrise a function with m-dimensional output. Here $\|\cdot\|_{\text{Fro}}$ denotes the Frobenius matrix norm, and $Y \in \mathbb{R}^{s\times m}$ is a matrix.

2.2.2 Regression with general basis functions

The previous example of polynomial regression is just a special case of regression with more general (nonlinear) basis functions. The parametrisation of f_w remains the same as in (2.8), but the basis functions $\phi : \mathbb{R}^m \to \mathbb{R}^{d+1}$ can represent more general classes of functions and not just polynomials. A typical example are *radial basis functions*, i.e.

$$\phi(x) := \begin{pmatrix} \varphi(\|x - \mu_0\|) & \varphi(\|x - \mu_1\|) & \dots & \varphi(\|x - \mu_d\|) \end{pmatrix}^\top ,$$

for a function $\varphi : \mathbb{R} \to \mathbb{R}$ and points $\{\mu_i\}_{i=0}^d$ with $\mu_i \in \mathbb{R}^m$ for each $i \in \{0, \dots, d\}$. Classical choices for φ are

$$\varphi(x) := \begin{cases} 1 - \frac{1}{\alpha}|x| & |x| \leq \alpha \\ 0 & |x| > \alpha \end{cases} , \qquad \text{and} \qquad \varphi(x) := \exp\left(-\frac{x^2}{2\sigma^2}\right) .$$

In later sections we want to discuss how to analyse and minimise rather general empirical risk functions with loss functions ℓ that are not necessarily quadratic functions. Before we do so, we want to recall some basic concepts in convex analysis first.

2.3 Convex analysis

In this section we want to more closely investigate cost- or loss-functions such as the MSE (2.3) introduced in the previous section. In particular, we want to define convexity of a function, verify when a function is convex and show why convexity is useful.

Definition 2.1 (Convex set)**.** *A subset $\mathcal{C} \subset \mathbb{R}^n$ is said to be* convex *if*

$$\lambda w + (1 - \lambda)v \in \mathcal{C} ,$$

for all elements $w, v \in \mathcal{C}$ and $\lambda \in [0, 1]$.

Based on this definition we can go ahead and define convex functions.

Definition 2.2 (Convex function)**.** *A function $E : \mathcal{C} \to \mathbb{R}$ is said to be* convex *if for all arguments $w, v \in \mathbb{R}^n$ and scalars $\lambda \in [0, 1]$ we observe*

$$E(\lambda w + (1 - \lambda)v) \leq \lambda E(w) + (1 - \lambda)E(v) .$$

In return, we can define concave functions as follows.

Definition 2.3 (Concave function)**.** *A function $E : \mathcal{C} \to \mathbb{R}$ is said to be* concave *if for all arguments $w, v \in \mathbb{R}^n$ and scalars $\lambda \in [0, 1]$ we observe*

$$E(\lambda w + (1 - \lambda)v) \geq \lambda E(w) + (1 - \lambda)E(v) .$$

Note that for a convex function E we automatically observe that $-E$ is concave, and vice versa. Before we continue, we want to introduce an extremely useful concept for the analysis of convex and continuously differentiable functions: Bregman distances, respectively Bregman divergences.

Definition 2.4 (Bregman distance)**.** *Let $E : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function, i.e. $\nabla E(w)$ exists for all $w \in \mathbb{R}^n$ and is continuous. Then its corresponding* Bregman distance *$D_E : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is defined as*

$$D_E(u, v) := E(u) - E(v) - \langle \nabla E(v), u - v \rangle ,$$

for all arguments $u, v \in \mathbb{R}^n$.

Note that Bregman distances are not necessarily distances in the sense of a metric. They merely describe the distance of a function E at point u to its linearisation around v. Before we continue to show that this distance is non-negative if and only if E is convex, we first want to introduce another useful distance based on the sum of two Bregman distances.

Definition 2.5 (Jensen-Shannon distance). *Suppose $E : \mathcal{C} \subset \mathbb{R}^n \to \mathbb{R}$ is a continuously differentiable function defined over a convex set $\mathcal{C}$. Then its* Jensen-Shannon distance *is defined as*

$$J_E^\lambda(u, v) := \lambda D_E(u, w) + (1 - \lambda) D_E(v, w) \,,$$

for $w := \lambda u + (1 - \lambda) v \in \mathcal{C}$ and $\lambda \in [0, 1]$.

Straight-forward computations reveal that this Jensen-Shannon distance is nothing but the difference between the convex combination of $E(u)$ and $E(v)$ and E applied to the convex combination of u and v, i.e.

$$J_E^\lambda(u, v) = \lambda E(u) + (1 - \lambda) E(v) - E(\lambda u + (1 - \lambda) v) \,. \tag{2.10}$$

Remark 2.1. From the Definition 2.2 we instantly observe that $J_E^\lambda(u, v) \geq 0$ for all $u, v \in \mathcal{C}$ if and only if E is convex.

Remark 2.2. For the special choice $\lambda = \frac{1}{2}$ the Jenson-Shannon distance reads

$$J_E^{\frac{1}{2}}(u, v) = \frac{1}{2} \left(D_E(u, w) + D_E(v, w) \right) = \frac{1}{2}(E(u) + E(v)) - E\left(\frac{u + v}{2} \right) \,,$$

and is known as the *Burbea-Rao distance.* In particular, the Burbea-Rao distance is symmetric, i.e $J_E^{\frac{1}{2}}(u, v) = J_E^{\frac{1}{2}}(v, u)$.

From (2.10) we immediately observe that the Jensen-Shannon distance is non-negative if and only if E is convex, which immediately follows from the definition of convexity, Definition 2.2. The following corollary shows that also the Bregman distance is non-negative if and only if E is convex.

Corollary 2.1. *Let $E : \mathcal{C} \to \mathbb{R}$ be a differentiable function defined over a convex set $\mathcal{C} \subset \mathbb{R}^n$. Then E is convex if and only if its corresponding Bregman distance D_E is non-negative for all arguments, i.e.*

$$D_E(u, v) \geq 0 \,,$$

for all $u, v \in \mathcal{C}$.

Proof (non-examinable). $\Rightarrow$: We first consider the case $n = 1$. Given $u, v \in \mathcal{C}$, we conclude $u + \lambda(v - u) \in \mathcal{C}$ due to the convexity of $\mathcal{C}$. From the convexity of E we observe

$$\begin{aligned} & E(u + \lambda(v - u)) \leq (1 - \lambda) E(u) + \lambda E(v) \,, \\ \Rightarrow \qquad & \lambda E(v) - \lambda E(u) - (E(u + \lambda(v - u)) - E(u)) \geq 0 \,, \\ \Rightarrow \qquad & E(v) - E(u) - \frac{E(u + \lambda(v - u)) - E(u)}{\lambda} \geq 0 \,. \end{aligned}$$

Taking the limit $\lambda \to 0$ of the left-hand-side then yields

$$\lim_{\lambda \to 0} E(v) - E(u) - \frac{E(u + \lambda(v - u)) - E(u)}{\lambda} = E(v) - E(u) - E'(u)(v - u) \,;$$

hence, we can conclude

$$E(v) - E(u) - E'(u)(v-u) \geq 0\,. \tag{2.11}$$

For general n we define the function $g(\lambda) := E((1-\lambda)u + \lambda v)$. By applying the chain rule we obtain

$$g'(\lambda) = \langle \nabla E((1-\lambda)u + \lambda v), v - u \rangle\,.$$

Since E is convex, it is straight-forward to see that g is also convex and that (2.11) therefore implies

$$g(s) - g(t) - g'(t)(s-t) \geq 0\,,$$

for any $s, t \in [0,1]$. For the particular choices $s = 1$ and $t = 0$ this inequality reads as $g(1) - g(0) - g'(0) \geq 0$, which is equivalent to

$$E(v) - E(u) - \langle \nabla E(u), v - u \rangle \geq 0\,.$$

Consequently, the Bregman distance D_E is non-negative for all input arguments in $\mathcal{C}$.
$\Leftarrow$: suppose we have $u, v \in \mathcal{C}$, then $w \in \mathcal{C}$ with $w := u + \lambda(v-u)$ due to the convexity of $\mathcal{C}$. Since the Bregman distance is non-negative for all arguments in $\mathcal{C}$, we can conclude

$$D_E(u,w) \geq 0 \qquad \text{and} \qquad D_E(v,w) \geq 0\,,$$

and therefore also

$$J_E^\lambda(u,v) = \lambda D_E(u,w) + (1-\lambda) D_E(v,w) \geq 0$$

for $\lambda \in [0,1]$. This implies

$$0 \leq J_E^\lambda(u,v) = \lambda E(u) + (1-\lambda)E(v) - E(\lambda u + (1-\lambda)v)\,.$$

Hence, the function E is indeed convex. $\square$

There is a close connection between convexity and global minimisers of a function. A global minimiser is defined as follows.

Definition 2.6 (Global minimiser)**.** *Suppose $E : \mathbb{R}^n \to \mathbb{R}$ is a function for which there exists a constant $\hat{E}$ with*

$$-\infty < \hat{E} \leq E(w)\,, \qquad \forall w \in \mathbb{R}^n\,,$$

i.e. the infimum of E exists and is attained. Then $\hat{E}$ is called the global minimum*. Moreover, let $\hat{w}$ denote the argument for which $E(\hat{w}) = \hat{E}$ and, hence,*

$$E(\hat{w}) \leq E(w)$$

holds true, for all $w \in \mathbb{R}^n$. Then $\hat{w}$ is known as the global minimiser *of E.*

For convex functions $E : \mathcal{C} \subset \mathbb{R}^n \to \mathbb{R}$ that are also differentiable, i.e. $\nabla E(w)$ exists for all $w \in \mathbb{R}^n$, we observe that their gradient can help us to determine a global minimiser of E.

Lemma 2.1. *Suppose $E : \mathcal{C} \subset \mathbb{R}^n \to \mathbb{R}$ is a convex function that is bounded from below and also differentiable. Then an argument $\hat{w}$ satisfies $\nabla E(\hat{w}) = 0$ if and only if $\hat{w}$ is a global minimiser.*

Proof. Since E is convex, we know $D_E(w, \hat{w}) \geq 0$ for all $w \in \mathcal{C}$ due to Corollary 2.1. This is equivalent to

$$E(w) - E(\hat{w}) - \langle \underbrace{\nabla E(\hat{w})}_{=0}, w - \hat{w} \rangle \geq 0 \, ,$$

which proves that $\hat{w}$ is a global minimiser. $\square$

Based on the previous considerations we know that (2.7) can only be a solution of (2.5) if $\nabla E(\hat{w}) = 0$ for $E(w) := \frac{1}{2s} \sum_{i=1}^s |\langle x_i, w \rangle - y_i|^2$ and

$$\hat{w} = \begin{pmatrix} \|x_1\|^2 & \langle x_1, x_2 \rangle & \cdots & \langle x_1, x_n \rangle \\ \langle x_2, x_1 \rangle & \|x_2\|^2 & \cdots & \langle x_2, x_n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle x_n, x_1 \rangle & \langle x_n, x_2 \rangle & \cdots & \|x_n\|^2 \end{pmatrix}^{-1} \sum_{i=1}^s y_i x_i \, .$$

In order to verify this, we need to compute the gradient ∇E and show that it coincides with the previous equation, which is left as a coursework exercise.

Note that we can also write the scalar product as the multiplication of a row with a column vector, i.e. $\langle x, w \rangle = x^\top w$. With this notation we can write the previous problem in a more compact form by defining

$$X := \begin{pmatrix} x_1^\top \\ x_2^\top \\ \vdots \\ x_s^\top \end{pmatrix} \in \mathbb{R}^{s \times n} \qquad \text{and} \qquad y := \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_s \end{pmatrix} \in \mathbb{R}^s \, .$$

Using this notation the linear regression problem (2.5) then reads

$$\hat{w} = \arg\min_{w \in \mathbb{R}^n} \left\{ \frac{1}{2s} \|Xw - y\|^2 \right\} \, .$$

The corresponding optimality condition $\nabla E(\hat{w}) = 0$ reads as

$$X^\top X \hat{w} = X^\top y \, , \tag{2.12}$$

which is also known as the normal equation associated with $X\hat{w} = y$.

2.3.1 A comment on existence and uniqueness

Before we investigate issues of regression problems due to ill-conditioning of the data matrices we want to briefly talk about existence and uniqueness of regression problems of the form (2.12). Do solutions to (2.12) always exist, and if they exist, are they unique? To answer the first question, we require the following lemma.

Lemma 2.2. *For every matrix $X \in \mathbb{R}^{s \times (d+1)}$ we observe*

$$\ker(X^\top X) = \ker(X) \qquad \textit{and} \qquad \operatorname{ran}(X^\top X) = \operatorname{ran}(X^\top) \, .$$

Proof (non-examinable). For $w \in \ker(X)$ we know $Xw = 0$. Multiplying $X^\top$ from the left, i.e. $X^\top X w = 0$, therefore implies $w \in \ker(X^\top X)$ and, thus, $\ker(X) \subset \ker(X^\top X)$. For $w \in \ker(X^\top X)$ we know $X^\top X w = 0$. Taking the inner product with w then yields $0 = \langle X^\top X w, w\rangle = \langle Xw, Xw\rangle = \|Xw\|^2$, which already implies $Xw = 0$. Hence, we concluded $w \in \ker(X)$, and thus, $\ker(X^\top X) \subset \ker(X)$. As an immediate consequence, we conclude $\ker(X) = \ker(X^\top X)$.

A fundamental statement in linear algebra says that $\ker(X)^\perp = \operatorname{ran}(X^\top)$, where $\perp$ denotes the orthogonal complement. Based on the first part of the proof, we verify

$$\operatorname{ran}(X^\top) = \ker(X)^\perp = \ker(X^\top X)^\perp = \ker((X^\top X)^\top)^\perp = \operatorname{ran}(X^\top X)\,,$$

which concludes the proof. □

An immediate consequence of Lemma 2.2 is that a solution of (2.12) always exists.

Theorem 2.1. *There always exists a solution of the Normal Equation* (2.12)*, for all* $X \in \mathbb{R}^{s\times(d+1)}$ *and* $y \in \mathbb{R}^s$*.*

Proof (non-examinable). By definition we have $X^\top y \in \operatorname{ran}(X^\top)$. From Lemma 2.2 we know that $\operatorname{ran}(X^\top) = \operatorname{ran}(X^\top X)$, which implies $X^\top y \in \operatorname{ran}(X^\top X)$. This means that there must exist a $w \in \mathbb{R}^{d+1}$ with $X^\top X w = X^\top y$. □

Now that we know that a solution to (2.12) always exists, we need to ask ourselves when that solution is unique. We immediately see that this is the case if and only if $\ker(X) = \{0\}$. If the kernel of X contains elements v other than 0, we observe

$$X^\top y = X^\top X\hat{w} = X^\top(X\hat{w} + \underbrace{Xv}_{=0}) = X^\top X(\hat{w} + v)\,,$$

due to the linearity of the matrix-matrix- and matrix-vector product. Hence, $\hat{w} + v$ is also a solution of (2.12). Note that this is not a contradiction to $\hat{w}$ being a global minimiser of (2.3), since

$$\operatorname{MSE}(\hat{w} + v) = \operatorname{MSE}(\hat{w})\,.$$

Amongst other challenges, we are going to address the non-uniqueness issue in the following section.

2.4 Ill-conditioned regression problems & regularisation

Solving a finite-dimensional regression problem can be challenging if the underlying data matrix is ill-conditioned. Before we define what this means, we want to lead with a simple example. Suppose we want to fit a line to two data point pars (x_1, y_1) and (x_2, y_2) with $x_1 = 1 - c$, $y_1 = 1$, $x_2 = 1 + c$ and $y_2 = 1$. As in the previous section, we can solve the regression problem via (2.12) for the matrix

$$X = \begin{pmatrix} 1 & 1-c \\ 1 & 1+c \end{pmatrix}$$

and the data vector $y = (y_1, y_2)^\top = \begin{pmatrix} 1 & 1 \end{pmatrix}^\top$. The linear system that we obtain reads

$$\begin{pmatrix} 1 & 1 \\ 1 & 1+c^2 \end{pmatrix} \hat{w} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}\,,$$

or

$$\hat{w} = \begin{pmatrix} 1 + c^{-2} & -c^{-2} \\ -c^{-2} & c^{-2} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} ,$$

if we solve for $\hat{w}$ directly. It is straight-forward to see (or to calculate) that $\hat{w} = \begin{pmatrix} 1 & 0 \end{pmatrix}^\top$ is the unique solution of this linear system, which makes perfect sense as the data points lie on a line with constant shift $\hat{w}_0 = 1$ and slope $\hat{w}_1 = 1$. We now want to investigate how things change if we make an error when taking our measurements. Suppose instead of y we measure $y_\delta = \begin{pmatrix} 1 - \varepsilon & 1 + \varepsilon \end{pmatrix}^\top$, for some constant $\varepsilon > 0$. The linear system remains the same, but the right-hand-side changes, and we have to solve

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 + c^2 \end{pmatrix} \hat{w}_\delta = \begin{pmatrix} 1 \\ 1 + c\varepsilon \end{pmatrix} ,$$

respectively

$$\hat{w}_\delta = \begin{pmatrix} 1 + c^{-2} & -c^{-2} \\ -c^{-2} & c^{-2} \end{pmatrix} \begin{pmatrix} 1 \\ 1 + c\varepsilon \end{pmatrix} ;$$

hence, the solution to this problem reads $\hat{w}_\delta = \begin{pmatrix} 1 - c^{-1}\varepsilon & c^{-1}\varepsilon \end{pmatrix}^\top$. Now we observe something interesting: if ε is reasonably small and the constant c is significantly larger, not much changes in terms of the solution. Consider e.g. the values $c = 1/2$ and $\varepsilon = 1/100$. We immediately compute $\hat{w}_\delta = \begin{pmatrix} 0.98 & 0.02 \end{pmatrix}^\top$, which is not very different from $\hat{w} = \begin{pmatrix} 1 & 0 \end{pmatrix}^\top$. However, we get a completely different picture if c is substantially smaller than ε. If we for instance choose $c = 1/1000$ while ε remains $\varepsilon = 1/100$, the solution $\hat{w}_\delta$ changes to $\hat{w}_\delta = \begin{pmatrix} -9 & 10 \end{pmatrix}^\top$. This is very different from $\hat{w}$. The error in the data $\delta := \|y - y_\delta\| = \sqrt{2}\varepsilon$ is very small ($\sqrt{2}/100 \approx 0.0141$), whereas the error in the reconstruction, i.e. $\|\hat{w} - \hat{w}_\delta\| = \sqrt{164} \approx 12.8063 \gg 0.0141$, is heavily amplified. In this particular example it is probably not surprising to see such a phenomenon: the closer the two inputs x_1 and x_2 move together, the more significant the impact of small variations in y on the slope and shift of the regression line. But can this phenomenon also appear in more complex scenarios?

To address this question we want to look at the solution of (2.12) for more general data matrices X (or matrices $\Phi(X)$) and outputs y and y_δ with the help of the singular value decomposition. For simplicity, we focus on the X notation, but all steps can be carried out for any matrix. Suppose that $X = U\Sigma V^\top$ is the singular value decomposition of X. We observe $X^\top X = V\Sigma^2 V^\top$, where Σ^2 is defined as the diagonal matrix with squared singular values on the diagonal. We assume that $X^\top X$ is invertible and that $d < s$. If we rewrite (2.3) in terms of the singular value decomposition of X, we obtain

$$\hat{w} = \sum_{j=1}^{d+1} \sigma_j^{-1} v_j \langle v_j, X^\top y \rangle \qquad \text{and} \qquad \hat{w}_\delta = \sum_{j=1}^{d+1} \sigma_j^{-1} v_j \langle v_j, X^\top y_\delta \rangle ,$$

respectively

$$\hat{w} - \hat{w}_\delta = \sum_{j=1}^{d+1} \sigma_j^{-1} v_j \langle v_j, X^\top (y - y_\delta) \rangle$$

for their difference. Computing the squared Euclidean norm of this difference yields

$$\begin{aligned}\|\hat{w}-\hat{w}_\delta\|^2 &= \left\|\sum_{j=1}^{d+1}\sigma_j^{-1}v_j\langle v_j, X^\top(y-y_\delta)\rangle\right\|^2 = \sum_{j=1}^{d+1}\sigma_j^{-2}\|v_j\|^2\left|\langle v_j, X^\top(y-y_\delta)\rangle\right|^2\\ &\leq \sum_{j=1}^{d+1}\sigma_j^{-2}\|X^\top(y-y_\delta)\|^2 \leq \sigma_{d+1}^{-2}(d+1)\|X^\top(y-y_\delta)\|^2\\ &\leq \sigma_{d+1}^{-2}(d+1)\|X\|^2\|y-y_\delta\|^2 = \left(\frac{\sigma_1}{\sigma_{d+1}}\right)^2(d+1)\|y-y_\delta\|^2\,.\end{aligned}$$

If we assume that out initial error $\|y-y_\delta\|$ is bounded by a constant $\delta > 0$, we can therefore conclude

$$\|\hat{w}-\hat{w}_\delta\| \leq \kappa\,\delta\,\sqrt{d+1}\,,$$

where $\kappa := \sigma_1/\sigma_{d+1}$ is the so-called *condition number* that determines the amplification of the error in the worst case scenario. The consequence of this exercise is that we now know that the ratio of the largest and smallest singular value of X is important for the amplification of errors in our data. If we have any influence on the data collection process, we should aim to collect data points $\{x_i\}_{i=1}^s$ such that the matrix X has a small condition number, i.e. X is *well-conditioned.* Matrices X with large condition numbers are called *ill-conditioned.* In the following we discuss how to compensate ill-conditioning with what is known to be Tikhonov regularisation or ridge regression.

2.4.1 Ridge regression

In the previous section we have seen that worst-case error amplification is a consequence of data matrices with large condition numbers. A relatively straight-forward idea to combat instability is by approximating the original regression problem by a problem with lower condition number. Writing the left-hand-side of the normal equation (2.12), i.e. $X^\top X\hat{w} = X^\top y$, in terms of its singular value decomposition reads

$$\sum_{j=1}^{d+1}\sigma_j^2 v_j\langle v_j, \hat{w}\rangle\,. \tag{2.13}$$

We now replace (2.13) with a version where we shift the singular values by a constant, positive factor α, i.e.

$$\sum_{j=1}^{d+1}(\sigma_j^2+\alpha)v_j\langle v_j, w_\alpha\rangle\,. \tag{2.14}$$

Reverting back from (2.14) to the matrix-vector-multiplication representation, we have effectively modified the normal equation to

$$\left(X^\top X+\alpha I\right)w_\alpha = X^\top y\,, \tag{2.15}$$

where $I \in \{0,1\}^{(d+1)\times(d+1)}$ denotes the identity matrix. The nice thing about $X^\top X+\alpha I$ is that its condition number is $\kappa(X^\top X+\alpha I) = \sqrt{(\sigma_1^2+\alpha)/(\sigma_{d+1}^2+\alpha)}$, instead of $\kappa(X^\top X) = \sigma_1/\sigma_{d+1}$.

If we have a matrix $X^\top X$ with $\sigma_1 = \sqrt{2}$ and $\sigma_{d+1} = 1/\sqrt{2000000}$ as in the previous section for example, we have a large condition number of $\kappa = 2000$. If we consider (2.15) instead of (2.12) for $\alpha = 1$, we observe $\kappa(X^\top X + \alpha I) \approx \sqrt{3} \ll 2000 = \kappa(X^\top X)$. As a consequence, any worst-case error amplification for the ridge regression is much smaller, at the cost of potentially altering the original problem dramatically. One thing that is also not clear is how to choose the parameter α; we will address this question in the next section. Before we do so, we want to present an alternative characterisation of (2.15).

Theorem 2.2 (Ridge regression). *The solution w_α of* (2.15) *is the unique solution of the optimisation problem*

$$w_\alpha = \operatorname*{arg\,min}_{w \in \mathbb{R}^{d+1}} \left\{ \frac{1}{2}\|Xw - y\|^2 + \frac{\alpha}{2}\|w\|^2 \right\} . \tag{2.16}$$

Proof. This proof is left as an exercise. □

The beauty of formulation (2.16) is that we immediately see that we still try to minimise the mean-squared error, but at the same time also ensure that the squared norm of the weights does not become too large. Both goals have to be balanced with a reasonable choice of α. We discuss this choice in the next section. Note that Problem (2.16) is known as *ridge regression* in the context of machine learning and *Tikhonov regularisation* in the context of inverse & ill-posed problems. The parameter α is known as the *regularisation parameter.* In the following section we want to investigate how to choose hyperparameters such as the regularisation parameter in some optimal way.

2.5 Model selection

Generally speaking, the major goal in most machine learning problems is to approximate (or interpolate) an unknown function $\hat{f} : \mathbb{R}^n \to \mathbb{R}^m$ for a given set of data points sampled from an unknown distribution $\mathcal{D}$. The assumption that we are in a supervised learning setting implies that we sample corresponding pairs (x, y) of input **and** output data points. The goal of supervised machine learning is then to find $\hat{f}$ such that it minimises the *population risk*, *expected risk* or *expected error*, which is defined as

$$E(f) := \mathbb{E}_{x,y}\left[\ell(f(x), y)\right] , \tag{2.17a}$$

$$= \int_{(x,y)\in\mathcal{D}} \ell(f(x), y)\rho(x, y)\, dx\, dy . \tag{2.17b}$$

Here $\ell : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is a so-called loss function that measures the difference between $f(x)$ and y, while $\rho : \mathcal{D} \to \mathbb{R}$ is the unknown joint probability density function. Note that the latter implies

$$\rho(x, y) \geq 0 \quad \text{a.e.} \qquad \text{and} \qquad \int_{(x,y)\in\mathcal{D}} \rho(x, y)\, dx\, dy = 1 .$$

The key problem with computing the population risk E, let alone minimise it, is that we do not have access to it as we do not know ρ. In practice, all we can do is to draw $|S|$ i.i.d. samples from $\mathcal{D}$ and consider minimising the *empirical risk*

$$L_S(f) := \frac{1}{|S|} \sum_{(x_i,y_i)\in S} \ell(f(x_i), y_i) \tag{2.18}$$

instead. Here $|S|$ denotes the cardinality of the set S, respectively the number of elements of S. The difference of (2.17) and (2.18) is known as the *generalisation error*

$$\begin{aligned} G_S(f) &= E(f) - L_S(f)\,, \\ &= \mathbb{E}_{x,y}\left[\ell(f(x), y)\right] - \frac{1}{|S|}\sum_{(x_i,y_i)\in S} \ell(f(x_i), y_i)\,. \end{aligned}$$

Note that we have defined $\hat{f}$ as the function that minimises (2.17), i.e.

$$\hat{f} = \operatorname*{arg\,min}_{f\in\mathcal{F}} \mathbb{E}_{(x,y)\in\mathcal{D}}\left[\ell(f(x), y)\right]\,,$$

where $\mathcal{F}$ denotes some suitable function space. Now we assume that we want to approximate $\hat{f}$ with a parametric function f_{w_t}, whose parameters w_t are computed by minimising the empirical risk on a set of data points S_t that are sampled from the distribution $\mathcal{D}$; we will refer to this set of points as the *training set*. The empirical risk for this function over the set S_t is then known as the *training error*, which reads

$$L_{S_t}(f_{w_t}) = \frac{1}{|S_t|}\sum_{(x_i,y_i)\in S_t} \ell(f_{w_t}(x_i), y_i)\,,$$

and we have f_{w_t} with w_t defined as

$$w_t := \operatorname*{arg\,min}_{w\in\mathbb{R}^{d+1}} L_{S_t}(f_w)\,.$$

If we sample a different set of data points from the same distribution $\mathcal{D}$ and denote this set as the *validation set* or *test set* S_v, then we can define the *validation error*

$$L_{S_v}(f_{w_t}) = \frac{1}{|S_v|}\sum_{(x_i,y_i)\in S_v} \ell(f_{w_t}(x_i), y_i)\,.$$

The validation error is of particular interest, as it approximates the population risk in expectation. An algorithm or method that aims at approximating $\hat{f}$ via a parametric function f_{w_t} is said to generalise, if the generalisation error G_{S_v} converges to zero if the number of samples converges to infinity, i.e $\lim_{|S_v|\to\infty} G_{S_v}(f_{w_t}) = 0$. Since E cannot be computed, the generalisation error cannot be computed either. Instead, the goal of research in statistical learning is to bound the generalisation error in probability.

Example 2.1 (Ridge regression)**.** Suppose we have collected a set of samples and use them as our training dataset $S_t := \{(x_i, y_i) \in \mathcal{D} \,|\, i \in \{1, \ldots, s\}\}$. In linear ridge regression, we recall that the idea is to approximate $\hat{f}$ via $f_{w_t}(x) := \langle \phi(x), w_t\rangle$, where the weights w_t are computed via

$$w_t = \operatorname*{arg\,min}_{w\in\mathbb{R}^{d+1}} \left\{ \frac{1}{2|S_t|}\sum_{i=1}^{s} |\langle \phi(x_i), w\rangle - y_i|^2 + \frac{\alpha}{2}\|w\|^2 \right\}\,.$$

Here, $\phi : \mathbb{R}^m \to \mathbb{R}^{d+1}$ is a the data-augmentation map that we had introduced earlier and $\alpha > 0$ is the regularisation parameter. We can then compute the validation error on a set of different samples $S_v := \{(x_i, y_i) \in \mathcal{D} \setminus S_t \,|\, i \in \{1, \ldots, s\}\}$ via

$$L_{S_v}(f_{w_t}) = L_{S_v}(\langle \phi(x_i), w_t\rangle) = \frac{1}{2|S_v|}\sum_{(x_i,y_i)\in S_v} (\langle \phi(x_i), w_t\rangle - y_i)^2\,.$$

Choosing an optimal regularisation parameter $\hat{\alpha}$ in terms of the validation error can then be formulated as the bilevel optimisation problem

$$\hat{\alpha} = \arg\min_{\alpha \geq 0} L_{S_v}(\langle \phi(x_i), w_t \rangle) \quad \text{subject to} \quad w_t = \arg\min_{w \in \mathbb{R}^{d+1}} \left\{ \frac{1}{2s} \sum_{i=1}^{s} |\langle \phi(x_i), w \rangle - y_i|^2 + \frac{\alpha}{2} \|w\|^2 \right\} . \tag{2.19}$$

In practice, this hyperparameter $\hat{\alpha}$ is often determined by applying a grid-search strategy to approximately solve (2.19).

In the following section we will further analyse the validation error for a new sample in expectation over all possible training sets in the context of ℓ being the mean-squared error function.

2.6 Bias-variance decomposition

Following up on the previous section, we now assume the abstract data generation model

$$y_\varepsilon = \hat{f}(x) + \varepsilon ,$$

where $\hat{f}$ is some unknown, deterministic function mapping inputs x onto outputs y. We assume that the outputs y_ε that we measure are $y = \hat{f}(x)$ plus a random variable ε drawn from some distribution $\mathcal{D}_\varepsilon$ with zero expectation, and variance σ^2, i.e.

$$\mathbb{E}_\varepsilon [\varepsilon] = 0 \qquad \text{and} \qquad \mathrm{Var}_\varepsilon[\varepsilon] = \sigma^2 .$$

One can easily verify that this implies

$$\mathbb{E}_\varepsilon[y_\varepsilon] = \mathbb{E}_\varepsilon[\hat{f}(x)] = \hat{f}(x) \qquad \text{and} \qquad \mathrm{Var}_\varepsilon(y_\varepsilon) = \sigma^2$$

in particular. Further, we assume that each pair (x, y_ε) is a sample from an unknown distribution $\mathcal{D}$ and that we have collected a finite number of samples from this distribution in each set S_t. Given a parametrised prediction function f_{w_t} based on one set S_t, we want to investigate the expected squared error of the difference of $y_\varepsilon = \hat{f}(x) + \varepsilon$ and f_{w_t} at a specific pair of points $(\tilde{x}, \tilde{y})$. Hence, we investigate

$$\mathbb{E}_{t,\varepsilon} \left[\left(\hat{f}(\tilde{x}) + \varepsilon - f_{w_t}(\tilde{x}) \right)^2 \right] ,$$

where $\mathbb{E}_t$ denotes the expectation over the sets S_t, while $\tilde{x}$ is a new sample outside of the training set. We have also dropped the factor 1/2 for notational convenience. In the following, we want to show that we can split this expectation into three important components: the so-called *bias*, the variance and the noise variance. In particular, we observe

$$\begin{aligned}
&\mathbb{E}_{t,\varepsilon}\left[\left(\hat{f}(\tilde{x}) + \varepsilon - f_{w_t}(\tilde{x})\right)^2\right] ,\\
&= \mathbb{E}_{t,\varepsilon}\left[\varepsilon^2 + 2\,\varepsilon\left(\hat{f}(\tilde{x}) - f_{w_t}(\tilde{x})\right) + \left(\hat{f}(\tilde{x}) - f_{w_t}(\tilde{x})\right)^2\right] ,\\
&= \mathrm{Var}_\varepsilon[\varepsilon] + \mathbb{E}_t\left[2\,\mathbb{E}_\varepsilon[\varepsilon]\left(\hat{f}(\tilde{x}) - f_{w_t}(\tilde{x})\right)\right] + \mathbb{E}_t\left[\left(\hat{f}(\tilde{x}) - f_{w_t}(\tilde{x})\right)^2\right] ,\\
&= \sigma^2 + \mathbb{E}_t\left[\left(\hat{f}(\tilde{x}) - f_{w_t}(\tilde{x})\right)^2\right] ,\\
&= \sigma^2 + \mathbb{E}_t\left[\left(\hat{f}(\tilde{x}) - \mathbb{E}_t\left[f_{w_t}(\tilde{x})\right] + \mathbb{E}_t\left[f_{w_t}(\tilde{x})\right] - f_{w_t}(\tilde{x})\right)^2\right] ,\\
&= \sigma^2 + \mathbb{E}_t\left[\left(\hat{f}(\tilde{x}) - \mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]\right)^2\right] + \mathbb{E}_t\left[\left(\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right] - f_{w_t}(\tilde{x})\right)^2\right] ,\\
&= \sigma^2 + \left(\hat{f}(\tilde{x}) - \mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]\right)^2 + \mathbb{E}_t\left[\left(\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right] - f_{w_t}(\tilde{x})\right)^2\right] ,\\
&= \sigma^2 + \mathrm{Bias}_t\left[f_{w_t}(\tilde{x})\right]^2 + \mathrm{Var}_t\left[f_{w_t}(\tilde{x})\right] ,
\end{aligned}$$

for

$$\begin{aligned}
\sigma^2 &:= \mathrm{Var}_\varepsilon[\varepsilon] ,\\
\mathrm{Bias}_t\left[f_{w_t}(\tilde{x})\right] &:= \mathbb{E}_t\left[f_{w_t}(\tilde{x})\right] - \hat{f}(\tilde{x}) ,\\
\mathrm{Var}_t\left[f_{w_t}(\tilde{x})\right] &= \mathbb{E}_t\left[\left(f_{w_t}(\tilde{x}) - \mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]\right)^2\right] .
\end{aligned}$$

Intuitively, the noise variance is an irreducible error in the measurements, the bias refers to a systematic model error, while the variance of a learning method indicates how much it will move around its mean. If we for example try to approximate a nonlinear function $\hat{f}$ with a linear function f_{w_t}, we will inevitably observe a systematic bias of our model fit. In the previous lecture we referred to this phenomenon as *underfitting*. If we try to fit a f_{w_t} to even the smallest fluctuation of $\hat{f}$, we observe the phenomenon of *overfitting*.

2.7 The LASSO

In Section 2.4.1 we have introduced the ridge regression model as a way to deal with ill-conditioned regression problems. In the previous two sections we have further explained how ridge regression can be used to prevent overfitting of a model function. In Theorem 2.2 we have characterised the ridge regression problem as an optimisation problem of the form

$$w_\alpha = \arg\min_{w\in\mathbb{R}^{d+1}}\left\{\frac{1}{2}\|\Phi(X)w - y\|^2 + \frac{\alpha}{2}\|w\|^2\right\} .$$

In this section we want to replace the squared Euclidean norm regularisation term in (2.16) with a one-norm, i.e.

$$w_\alpha = \arg\min_{w\in\mathbb{R}^{d+1}}\left\{\frac{1}{2}\|\Phi(X)w - y\|^2 + \alpha\|w\|_1\right\} , \tag{2.20}$$

with $\|w\|_1 := \sum_{j=0}^{d} |w_j|$. The motivation behind using the one-norm instead of the squared Euclidean lies in recovering simpler weights. Simple in this context means sparse, i.e. that we only want some coefficients of w_α to be non-zero while most coefficients are in fact zero. If we think of polynomial regression as an example, we could think of fitting a polynomial with a very high degree. Without knowing which coefficients of a polynomial should be relevant, we could potentially use the minimisation problem in Equation (2.20) to recover a weight vector with only a few non-zero entries. Intuitively, we can prevent overfitting this way, and we do not have to limit the degree in advance in order to do so. A regression problem of the form of (2.20) is known as the *Least Absolute Shrinkage and Selection Operator (LASSO)*. Before we discuss how to solve (2.20) computationally, we want to introduce a very basic and popular method for smooth optimisation, known as gradient descent.

2.7.1 Gradient descent

Gradient descent is an iterative procedure that aims at minimising general, differentiable functions E. Gradient descent is of the form

$$w^{k+1} = w^k - \tau \nabla E(w^k)\,, \tag{2.21}$$

for some energy E, an initial value $w^0 \in \mathbb{R}^n$ and a step-size parameter $\tau > 0$. In case of $E(w) = \frac{1}{2s}\|Xw - y\|^2$ for example, gradient descent reads

$$\begin{aligned} w^{k+1} &= w^k - \frac{\tau}{s} X^\top \left(X w^k - y \right)\,, \\ &= \left(I - \frac{\tau}{s} X^\top X \right) w^k + \frac{\tau}{s} X^\top y\,. \end{aligned} \tag{2.22}$$

The advantage of iteratively solving (2.22) is that we only need to compute matrix multiplications and basic arithmetic operations that are reasonably cheap. Also, with an algorithm like (2.21) we can address minimisation problems more general than minimising the MSE, which is going to be useful for the LASSO problem (2.20). On the downside, we yet have to determine when and under which conditions (2.21) really converges to a solution of the minimisation problem $\hat{w} = \arg\min_{w\in\mathbb{R}^n} E(w)$ and how quickly it converges to that solution. For this, we rewrite Equation (2.21) to

$$\begin{aligned} w^{k+1} &= \operatorname*{arg\,min}_{w\in\mathbb{R}^n} \left\{ E(w^k) + \langle \nabla E(w^k), w\rangle + \frac{1}{2\tau}\|w - w^k\|^2 \right\}\,, \\ &= \operatorname*{arg\,min}_{w\in\mathbb{R}^n} \left\{ E(w) + \frac{1}{2\tau}\|w - w^k\|^2 - E(w) + E(w^k) + \langle \nabla E(w^k), w - w^k\rangle \right\} \\ &= \operatorname*{arg\,min}_{w\in\mathbb{R}^n} \left\{ E(w) + \frac{1}{2\tau}\|w - w^k\|^2 - D_E(w, w^k) \right\}\,. \end{aligned}$$

The objective function $L^k(w) := \langle \nabla E(w^k), w\rangle + \frac{1}{2\tau}\|w - w^k\|^2$ is convex and differentiable with gradient $\nabla L(w) = \nabla E(w^k) + \frac{1}{\tau}(w - w^k)$. Hence, the global minimiser can be determined via $\nabla L(w^{k+1}) = 0$, which yields (2.21). Gradient descent is summarised in Algorithm 1. The questions that we need to ask ourselves now are the following: does Algorithm 1 converge to a minimiser of the objective function E and if yes, under what conditions does it converge? The following definition will come in handy for answering this question.

Algorithm 1 Gradient descent

```
Specify: Differentiable, convex function $E : \mathbb{R}^n \to \mathbb{R}$, step-size $\tau > 0$, index $K$
Initialise: $w^0 \in \mathbb{R}^n$
Iterate:
1: for $k = 0, \ldots, K-1$ do
2:     $w^{k+1} = w^k - \tau \nabla E(w^k)$
3: end for
return $w^K$.
```

Definition 2.7 (L-smooth functions)**.** *A continuously differentiable function $E : \mathcal{C} \subset \mathbb{R}^n \to \mathbb{R}$ is called* L-smooth *if*

$$\|\nabla E(w) - \nabla E(v)\| \leq L\|w - v\|$$

is guaranteed for all $w, v \in \mathcal{C}$ and a positive constant $L > 0$.

Theorem 2.3 (Convergence of Algorithm 1)**.** *Let $E : \mathcal{C} \subset \mathbb{R}^n \to \mathbb{R}$ be a convex and $1/\tau$-smooth function in the sense of Definition 2.7. Suppose $\hat{w}$ denotes a global minimiser of E, i.e. $\hat{w} = \arg\min_{w \in \mathbb{R}^n} E(w)$. Then the iterates of Algorithm 1 satisfy*

$$E(w^k) - E(\hat{w}) \leq \frac{C}{k}, \tag{2.23}$$

for a constant $C > 0$ that is independent of k. As a direct consequence, we observe $\lim_{k\to\infty} E(w^k) = E(\hat{w})$.

As usual, *the proof of this statement is non-examinable* and is left to the interested reader. In order to prove Theorem 2.3 we need to verify the following two lemmas first. The first one verifies a convenient property of smooth functions E: if E is $1/\tau$-smooth, then a function $J := \frac{1}{2\tau}\|\cdot\|^2 - E$ is automatically convex.

Lemma 2.3. *Let $E : \mathcal{C} \to \mathbb{R}$ be a $1/\tau$-smooth function over a convex domain $\mathcal{C} \subset \mathbb{R}^n$, for a positive constant $\tau > 0$. Then $J : \mathcal{C} \to \mathbb{R}$, with*

$$J(w) := \frac{1}{2\tau}\|w\|^2 - E(w),$$

is a convex function, for all $w \in \mathcal{C}$.

Proof (non-examinable): As E is $1/\tau$-smooth we conclude

$$\|\nabla E(w) - \nabla E(v)\| \leq \frac{1}{\tau}\|w - v\|.$$

Multiplying both sides with $\|w - v\|$ and making use of the Cauchy-Schwartz inequality $\langle x, y\rangle \leq \|x\|\|y\|$ leaves us with

$$\langle \nabla E(w) - \nabla E(v), w - v\rangle \leq \frac{1}{\tau}\|w - v\|^2 = \frac{1}{\tau}\langle w - v, w - v\rangle.$$

Subtracting the left-hand-side from the right-hand-side yields the inequality

$$0 \leq \left\langle \frac{1}{\tau}w - \nabla E(w) - \left(\frac{1}{\tau}v - \nabla E(v)\right), w - v \right\rangle = D_J(w, v) + D_J(v, w),$$

for $J := \frac{1}{2\tau}\| \cdot \|^2 - E$. From $D_J(w, v) + D_J(v, w) \geq 0$ for all u, v we can conclude $D_J(y + t(x - y), y) + D_J(y, y + t(x - y)) \geq 0$ for all x, y and $t \in [0, 1]$, which implies

$$\langle \nabla J(y + t(x - y)) - \nabla J(y), x - y \rangle \geq 0 \, . \tag{2.24}$$

If we define

$$f(t) := J(y + t(x - y)) \, ,$$

we observe $f'(t) = \langle \nabla J(y + t(x - y)), x - y \rangle$ and can therefore conclude $f'(t) \geq f'(0)$ from (2.24). Hence, we can estimate

$$\begin{aligned} J(x) = f(1) = f(0) + \int_0^1 f'(t) \, dt &\geq f(0) + f'(0) \\ &= J(y) - \langle \nabla J(y), x - y \rangle \, , \end{aligned}$$

which is true for all x, y. Hence, $D_J(w, v) + D_J(v, w) \geq 0$ for all arguments already implies $D_J(u, v) \geq 0$ for all arguments u, v. Corollary 2.1 then implies convexity of J. □

Before we continue with the actual proof of Theorem 2.3 we also verify the following intermediate result.

Lemma 2.4. *Let the same assumptions hold true as in Theorem 2.3 and suppose w^* is defined as $w^* := \arg\min_{w \in \mathbb{R}^n} \{E(w) + D_J(w, \overline{w})\}$ for some $\overline{w} \in \mathbb{R}^n$. Then the identity*

$$E(w^*) + D_E(w, w^*) + D_J(w, w^*) + D_J(w^*, \overline{w}) = E(w) + D_J(w, \overline{w})$$

holds for any $w \in \mathbb{R}^n$. Here $J : \mathbb{R}^n \to \mathbb{R}$ is defined as $J(w) := \frac{1}{2\tau}\|w\|^2 - E(w)$ and D_E and D_J denote the Bregman distances with respect to the functions E, respectively J.

Proof (non-examinable): As a consequence of Lemma 2.1 we can characterise w^* via the optimality condition

$$0 = \nabla E(w^*) + \nabla J(w^*) - \nabla J(\overline{w}) \, .$$

Taking an inner product with $w^* - w$ then yields

$$\begin{aligned} 0 &= -\langle \nabla E(w^*), w - w^* \rangle - \langle \nabla J(w^*) - \nabla J(\overline{w}), w - w^* \rangle \, , \\ &= D_E(w, w^*) - E(w) + E(w^*) - \langle \nabla J(w^*), w - w^* \rangle + \langle \nabla J(\overline{w}), w - w^* \rangle \, , \\ &= D_E(w, w^*) - E(w) + E(w^*) + D_J(w, w^*) - J(w) + J(w^*) + \langle \nabla J(\overline{w}), w - w^* \rangle \, , \\ &= D_E(w, w^*) - E(w) + E(w^*) + D_J(w, w^*) - J(w) + J(w^*) + \langle \nabla J(\overline{w}), w - \overline{w} + \overline{w} - w^* \rangle \, , \\ &= D_E(w, w^*) - E(w) + E(w^*) + D_J(w, w^*) - D_J(w, \overline{w}) + D_J(w^*, \overline{w}) \, , \end{aligned}$$

which concludes the proof. □

Proof of Theorem 2.3 (non-examinable). Applying Lemma 2.4 for $w^* = w^{k+1}$, $\overline{w} = w^k$ and $w = \hat{w}$ yields

$$\begin{aligned} E(\hat{w}) + D_J(\hat{w}, w^k) &= E(w^{k+1}) + D_E(\hat{w}, w^{k+1}) + D_J(\hat{w}, w^{k+1}) + D_J(w^{k+1}, w^k) \, , \\ &\geq E(w^{k+1}) + D_J(\hat{w}, w^{k+1}) \, , \end{aligned}$$

due to the convexity of E and J that implies $D_E(\hat{w}, w^{k+1}) \geq 0$ and $D_J(w^{k+1}, w^k) \geq 0$ because of Corollary 2.1. Hence, we observe

$$E(w^{k+1}) - E(\hat{w}) \leq D_J(\hat{w}, w^k) - D_J(\hat{w}, w^{k+1}) \, .$$

Summing from $k = 0, \dots, K-1$ then leads to

$$\sum_{k=0}^{K-1} E(w^{k+1}) - KE(\hat{w}) \leq D_J(\hat{w}, w^0) - D_J(\hat{w}, w^K) \, . \tag{2.25}$$

Another application of Lemma 2.4 for $w^* = w^{k+1}$, $\overline{w} = w^k$ and $w = w^k$ gives us

$$\begin{aligned} E(w^k) + \underbrace{D_J(w^k, w^k)}_{=0} &= E(w^{k+1}) + D_E(w^k, w^{k+1}) + D_J(w^k, w^{k+1}) + D_J(w^{k+1}, w^k) \, , \\ &\geq E(w^{k+1}) \, , \end{aligned}$$

again due to the convexity of E and J that implies $D_E(w^k, w^{k+1}) \geq 0$, $D_J(w^k, w^{k+1}) \geq 0$ and $D_J(w^{k+1}, w^k) \geq 0$. Hence, we can conclude $E(w^{k+1}) \leq E(w^k)$ for all $k = 0, \dots, K-1$, and in particular $KE(w^K) \leq \sum_{k=0}^{K-1} E(w^{k+1})$. Inserting this inequality in (2.25) implies (2.23).
Since J is convex we can further estimate

$$E(w^K) \leq E(\hat{w}) + \frac{D_J(\hat{w}, w^0)}{K}$$

for a positive constant $C := D_J(\hat{w}, w^0)$ independent of K, which allows us to conclude both $\lim_{K \to \infty} E(w^K) = E(\hat{w})$ and the proof. □

Example 2.2 (Convergent gradient descent for minimising the MSE)**.** For the MSE function $E(w) := \frac{1}{2s}\|Xw - y\|^2$ we have already verified $\nabla E(w) = \frac{1}{s}X^\top(Xw - y)$. We further verify

$$\|\nabla E(w) - \nabla E(v)\| = \frac{1}{s}\|X^\top X(w - v)\| \leq \frac{\|X^\top X\|}{s}\|w - v\| = \frac{\|X\|^2}{s}\|w - v\| \, ,$$

where $\|X\|$ denotes the operator norm of X (based on the Euclidean vector norm). Hence, E is $1/\tau$-smooth for $\tau \leq s/\|X\|^2$. As a consequence of Theorem 2.3 we know that gradient descent, respectively Algorithm 1, applied to the MSE is convergent for any step-size τ with $\tau \leq s/\|X\|^2$.

Remark 2.3. We want to emphasise that Theorem 2.3 not only guarantees convergence of Algorithm 1, but also provides a rate of convergence. This rate is $1/k$, and often this rate of convergence is highlighted with the big $\mathcal{O}$-notation, i.e.

$$E(w^k) - E(\hat{w}) = \mathcal{O}\left(\frac{1}{k}\right) \, .$$

This means that the left-hand-side is proportional to $1/k$. Suppose $D_J(\hat{w}, w^0) = 10$, then we require approximately $k = 1000$ iterations to ensure $E(w^k) - E(\hat{w}) \leq 10^{-2}$ according to Theorem 2.3. Next semester in advanced machine learning we want to address the question of whether we can have a $1/k^2$-convergence rate (or faster). To illustrate the gain in convergence speed, we would only require $k = 32$ instead of $k = 1000$ iterations in order to get the same accuracy, i.e. $E(w^K) - E(\hat{w}) \leq 10^{-2}$.

2.7.2 Gradient descent and the LASSO

In this section we want to discuss how we can use gradient descent to solve the LASSO problem (2.20). The key challenge is the non-differentiability of the one-norm $\| \cdot \|_1$. We therefore have to make this problem differentiable. We can do this by using the following neat trick. We can rewrite the modulus-function $| \cdot |$ in the one-norm as

$$|z| = \max_{p \in [-1,1]} zp \, ,$$

for any arbitrary scalar $z \in \mathbb{R}$. We now modify the modulus function by subtracting a multiple of a quadratic of the additional variable p and define

$$|z|_\tau := \max_{p \in [-1,1]} zp - \frac{\tau}{2}|p|^2 \, , \tag{2.26}$$

for some scalar parameter $\tau > 0$. A nice thing about this modification is that it has a closed-form solution that we can compute by computing $\hat{p} = \arg\max_{p \in [-1,1]} zp$, which reads

$$\hat{p} = \begin{cases} 1 & z > \tau \\ \frac{z}{\tau} & |z| \leq \tau \\ -1 & z < -\tau \end{cases} \, ,$$

(left as a coursework exercise) and inserting $\hat{p}$ into (2.26). This yields

$$|z|_\tau = \begin{cases} |z| - \frac{\tau}{2} & |z| > \tau \\ \frac{1}{2\tau}|z|^2 & |z| \leq \tau \end{cases} \, ,$$

which is also known as the Huber loss. This function is differentiable, and we can replace the one-norm $\|w\|_1 = \sum_{j=0}^d |w_j|$ in (2.20) with the funtion $H_\tau(w) = \sum_{j=0}^d |w_j|_\tau$, i.e.

$$w_\alpha^\tau = \arg\min_{w \in \mathbb{R}^{d+1}} \left\{ \frac{1}{2}\|\Phi(X)w - y\|^2 + \alpha H_\tau(w) \right\} \, . \tag{2.27}$$

Since all terms in Problem (2.27) are differentiable, we can solve (or approximate a solution of) (2.27) with Algorithm 1. There are obviously a few open questions, such as convexity of the Huber loss, Lipschitz-continuity of the overall gradient and whether minimisers of (2.27) and (2.20) coincide, which we won't address for now. The reason for this is that there is a minor but powerful modification of gradient descent known as proximal gradient descent or forward-backward splitting that is much for suitable for the minimisation of problems of the form (2.20).

2.7.3 Proximal gradient descent

Problems like the LASSO are problems where we minimise the sum of two functions. More precisely, they are of the form

$$w = \arg\min_{w \in \mathcal{C}} \left\{ E(w) + R(w) \right\} \, , \tag{2.28}$$

where $E : \mathbb{R}^n \to \mathbb{R}$ is a convex and continuously differentiable function, while $R : \mathcal{C} \subset \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is a proper, convex and lower semi-continuous function. Please do not worry too much about the assumptions on R; these are obviously important from a mathematical point of view

Algorithm 2 Proximal gradient descent

Specify: Continuously differentiable, convex function $E : \mathbb{R}^n \to \mathbb{R}$, proper, lower semi-continuous and convex function $R : \mathcal{C} \subset \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$, step-size $\tau > 0$, index K

Initialise: $w^0 \in \mathcal{C}^n$

Iterate:

```
1: for $k = 0, \ldots, K-1$ do
2:     $w^{k+1} = (I + \tau\partial R)^{-1}\left(w^k - \tau\nabla E(w^k)\right)$
3: end for
return $w^K$.
```

but are not really relevant for this module. I have included them for the interested reader, but you can very well survive this module without knowing what those assumptions mean. As already mentioned, typical examples of (2.28) include the LASSO problem (2.20), i.e.

$$w = \operatorname*{arg\,min}_{w \in \mathbb{R}^{d+1}} \left\{ \frac{1}{2s}\|Xw - y\|^2 + \alpha\|w\|_1 \right\} ,$$

but also other problems such as constrained MSE minimisation, i.e.

$$w = \operatorname*{arg\,min}_{w \in \mathcal{C}} \left\{ \frac{1}{2s}\|Xw - y\|^2 \right\} = \operatorname*{arg\,min}_{w \in \mathbb{R}^{d+1}} \left\{ \frac{1}{2s}\|Xw - y\|^2 + \chi_{\mathcal{C}}(w) \right\} ,$$

where $\chi_{\mathcal{C}} : \mathbb{R}^{d+1} \to \mathbb{R} \cup \{\infty\}$ is the characteristic function over the convex set $\mathcal{C} \subset \mathbb{R}^{d+1}$, i.e.

$$\chi_{\mathcal{C}}(w) := \begin{cases} 0 & w \in \mathcal{C} \\ \infty & w \notin \mathcal{C} \end{cases} .$$

We aim to minimise (2.28) with a modification of Algorithm 1 of the form

$$w^{k+1} = \operatorname*{arg\,min}_{w \in \mathcal{C}} \left\{ E(w) + R(w) + D_J(w, w^k) \right\} ,$$

again for the function $J(w) := \frac{1}{2\tau}\|w\|^2 - E(w)$. This method yields the so-called proximal gradient method that reads

$$\begin{aligned} w^{k+1} &= \operatorname*{arg\,min}_{w \in \mathcal{C}} \left\{ E(w^k) + \langle \nabla E(w^k), w - w^k \rangle + R(w) + \frac{1}{2\tau}\|w - w^k\|^2 \right\} , \\ &= \operatorname*{arg\,min}_{w \in \mathcal{C}} \left\{ E(w^k) + \langle \nabla E(w^k), w - w^k \rangle + R(w) + \frac{1}{2\tau}\|w - w^k\|^2 \right\} , \\ &= \operatorname*{arg\,min}_{w \in \mathcal{C}} \left\{ \frac{1}{2}\left\| w - \left(w^k - \tau\nabla E(w^k) \right) \right\|^2 + \tau R(w) \right\} , \\ &=: (I + \tau\partial R)^{-1}\left(w^k - \tau\nabla E(w^k) \right) . \end{aligned}$$

and the short-hand notation

$$(I + \tau\partial R)^{-1}(z) := \operatorname*{arg\,min}_{x \in \mathbb{R}^n} \left\{ \frac{1}{2}\|x - z\|^2 + \tau R(x) \right\} .$$

The mapping $(I + \tau\partial R)^{-1} : \mathbb{R}^n \to \mathcal{C}$ is known as the proximal operator or proximal map with respect to R, and the success of the proximal gradient method relies on the simplicity of this map. In the examples mentioned above, the proximal maps read

$$\left((I + \tau\alpha\partial\|\cdot\|_1)^{-1}(z)\right)_j = \begin{cases} z_j - \tau\alpha & z_j > \tau\alpha \\ 0 & |z_j| \leq \tau\alpha \\ z_j + \tau\alpha & z_j < -\tau\alpha \end{cases} \quad \text{and} \quad (I + \tau\alpha\partial\chi_{\mathcal{C}})^{-1}(z) = \text{proj}_{\mathcal{C}}(z)\,,$$

where $\text{proj}_{\mathcal{C}}$ denotes the orthogonal projector onto the set $\mathcal{C}$. If $\mathcal{C} := \{x \in \mathbb{R} \,|\, x \geq 0\}$ for example, then $\text{proj}_{\mathcal{C}}(z) = \max(0, z)$ We have summarised the proximal gradient descent in Algorithm 2. Convergence can be deduced in very similar fashion as for gradient descent, but we leave this for the advanced machine learning module in the next semester where this becomes more relevant.

2.8 Deep learning

So far, all supervised machine learning models that we have considered were regression models that were either linear or nonlinear in the input arguments x but linear in the weights w. We now want to shift our focus to a particular class of models that are nonlinear in both the input arguments and the weights: so-called *deep neural networks*.

Deep neural networks

In mathematical terms, a deep neural network is simply a function that is a composition of simple, parametrised functions and potentially many of them, i.e.

$$f_w(x) := \varphi_L\left(\varphi_{L-1}\left(\cdots \varphi_1\left(\varphi_0(x, w_1), w_2\right) \cdots, w_{L-1}\right), w_L\right)\,. \tag{2.29}$$

Here $\{\varphi_l\}_{l=1}^L$ is a family of L so-called *activation functions* that are parametrised with weights $w := \{w_l\}_{l=1}^L$. To be more precise, (2.29) is a deep neural network with L layers.
Typical activation functions are affine-linear transformations, i.e.

$$\varphi(x, W, b) := W^\top x + b\,.$$

In terms of terminology, $W \in \mathbb{R}^{n\times m}$ is a *weight* matrix and $b \in \mathbb{R}^{m\times 1}$ is the *bias* vector. This way $\varphi : \mathbb{R}^n \to \mathbb{R}^m$ maps inputs $x \in \mathbb{R}^n$ onto outputs $\varphi(x) \in \mathbb{R}^m$.
An example for a nonlinear activation function is the Heaviside function

$$\varphi(x) = H(x) := \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}\,.$$

Together with an affine-linear transformation, Heaviside functions paved the way for the definition of the *perceptron*, a simplified model loosely based on a neuron.

Example 2.3 (Perceptron)**.** An artificial neuron known as *perceptron* is defined as the nonlinear activation function

$$\varphi(x, w, b) := H(w^\top x + b) = \begin{cases} 1 & w^\top x \geq -b \\ 0 & w^\top x < -b \end{cases}\,,$$

for a weight vector $w \in \mathbb{R}^n$ and a bias $b \in \mathbb{R}$. Note that this function is a composition and itself could already be seen as two-layer neural network of the form

$$f(x, w, b) = \varphi_1(\varphi_0(x, w, b))$$

with $\varphi_1(x) := H(x)$ and $\varphi_0(x, w, b) := w^\top x + b$.

Another popular activation function is the so-called *rectifier* $\varphi : \mathbb{R} \to \mathbb{R}_{\geq 0}$, defined as

$$\varphi(x) := \max(0, x)\,.$$

Simple activation functions such as the Heaviside function or the rectifier can easily be extended to activation functions operating on vectors $x \in \mathbb{R}^n$ by defining $\varphi : \mathbb{R}^n \to \mathbb{R}^n$ with

$$\varphi(x) := (\varphi(x_1), \varphi(x_2), \dots, \varphi(x_n))^\top\,.$$

We will often abuse notation and use the same notation for the scalar and vector-valued variants of such simple activation functions. When we write a vector-valued rectifier $\varphi : \mathbb{R}^n \to \mathbb{R}^n$ as

$$\varphi(x) = \max(0, x)\,,$$

we mean $\varphi(x) = (\max(0, x_1), \max(0, x_2), \dots, \max(0, x_n))$ but write $\max(x, 0)$ for the sake of notational simplicity.

In combination with the affine-linear transformations they form what is known as *Rectified Linear Unit (ReLU)*, i.e.

$$\varphi(x, W, b) := \max\left(0,\ W^\top x + b\right)\,,$$

where we have used the simplified notation for the vector-valued rectifier.

Example 2.4 (ReLU neural networks)**.** Based on the previous considerations, an L-layer neural network with ReLU activation functions has the form

$$\varphi(x, w) = \max\left(0,\ W_L^\top \max\left(0,\ W_{L-1}^\top \max\left(\dots \max\left(0, W_1^\top x + b_1\right) \dots\right) + b_{L-1}\right) + b_L\right)\,,$$

or

$$\varphi(x, w) = \max\left(0, W_L^\top x^L + b_L\right)\,,$$

for

$$x^l := \begin{cases} \max\left(0, W_l^\top x^{l-1} + b_l\right) & l \in \{2, \dots, L\} \\ \max\left(0, W_1^\top x + b_1\right) & l = 1 \end{cases}\,.$$

Here the parameters w are defined as the collection of all weight matrices and bias vectors, i.e. $w = \left\{\{W_l\}_{l=1}^L, \{b_l\}_{l=1}^L\right\}$.

One last notable activation function that we want to discuss is the *softmax* activation function, which is defined as $\varphi : \mathbb{R}^n \to \mathbb{R}^n$ with

$$\varphi(x_1, \dots, x_n) = \left(\frac{\exp(x_1)}{\sum_{j=1}^n \exp(x_j)}, \dots, \frac{\exp(x_n)}{\sum_{j=1}^n \exp(x_j)}\right)^\top\,.$$

This activation function is extremely useful as it allows us to map arguments onto the (probability) simplex, i.e. $\varphi(x_1, \ldots, x_n)_i \geq 0$ for all $i \in \{1, \ldots, n\}$ and $\sum_{i=1} \varphi(x_1, \ldots, x_n)_i = 1$. The name stems from the fact that this activation function can be seen as a smooth approximation of the argmax function.
A general nonlinear regression model with deep neural networks then simply reads as minimising the empirical risk (2.18) of the form

$$w_t = \operatorname*{arg\,min}_{w \in \mathbb{R}^n} \left\{ \frac{1}{s} \sum_{i=1}^{s} \ell_i(f_w(x_i), y_i) \right\} , \tag{2.30}$$

for a family of loss functions $\{\ell_i\}_{i=1}^s$ with $\ell_i : \mathbb{R}^n \to \mathbb{R}^m$ for each $i \in \{1, \ldots, s\}$. As for the previous examples we can for instance choose $\ell_i(z) := \frac{1}{2}|z - y_i|^2$ in order to minimise the mean-squared error, but with a deep neural network as the model function, i.e.

$$\begin{aligned} w_t &= \operatorname*{arg\,min}_{w \in \mathbb{R}^n} \left\{ \frac{1}{2s} \sum_{i=1}^{s} |f_w(x_i) - y_i|^2 \right\} , \\ &= \operatorname*{arg\,min}_{w \in \mathbb{R}^n} \left\{ \frac{1}{2s} \sum_{i=1}^{s} |x_i^L - y_i|^2 \right\} , \end{aligned}$$

for

$$x_i^l = \varphi_l(x_i^{l-1}, w^l) , \qquad \text{for} \qquad \forall i \in \{1, \ldots, s\}, \forall l \in \{1, \ldots, L\} ,$$

assuming $x_i^0 = x_i$ for all $i \in \{1, \ldots, s\}$. Other choices such as the mean absolute error can certainly be used as well, and some will also be explored next semester. An important question that we want to address in the following section is the optimisation of the parameters w_t, also referred to as training or empirical risk minimisation as discussed in Section 2.5.

2.8.1 Training deep learning models

In the previous section we learned that training the parameters of a deep neural network is equivalent to minimising objective functions of the form (2.30). In principle, there is no reason why we cannot use algorithms such as gradient descent (Algorithm 1) and modifications such as stochastic gradient descent or subgradient descent that we will learn more about in the next semester. However, the key difference is that the objective in (2.30) is in general not a convex function anymore. This means that the convergence results that we have derived in earlier sections do not apply. This does not mean that there do not exist other conditions that could guarantee convergence, but this is beyond the scope of this module. We nevertheless will apply the same algorithms, keeping in mind that we do not necessarily have convergence guarantees as in the convex setting.

If we have a differentiable deep neural network with differentiable activation functions, we can (try to) train a deep neural network simply by minimising (2.30) via gradient descent, i.e. Algorithm 1. In order to do so, we are required to compute the gradient of the objective function w.r.t. the network parameters w. This can be done via backpropagation, which is a fancy name for applying the chain rule to the particular network architecture.
In the following, we focus on architectures of the form

$$x_i^l = \sigma(z_i^l) , \tag{2.31a}$$
$$z_i^l = W_l^\top x_i^{l-1} + b_l , \tag{2.31b}$$

for $x_i^0 = x_i$, nonlinear activation functions σ and all $i \in \{1, \ldots, s\}$ and $l \in \{1, \ldots, L\}$ and the empirical risk minimisation problem for the empirical risk function

$$L(W_1, \ldots, W_L, b_1, \ldots, b_L) = \frac{1}{s} \sum_{i=1}^{s} \ell(x_i^L, y_i) \,. \tag{2.32}$$

Other backpropagation rules for more general architectures can be derived in similar fashion. The following lemma characterises the partial derivatives of the empirical risk function with respect to the parameters.

Lemma 2.5. *When we define the quantity*

$$\delta_j^l := \frac{\partial L}{\partial x_j^l}$$

for $j \in \{1, \ldots, n_l\}$ and $l \in \{2, \ldots, L\}$, we can show that the partial derivatives of L with respect to the weights and biases satisfy

$$\begin{aligned} \delta_i^l &= \begin{cases} \sigma'(z_i^L) \odot \frac{1}{s} \nabla_1 \ell(x_i^L, y_i) & l = L \\ \sigma'(z_i^l) \odot W_{l+1} \delta^{l+1} & l \in \{1, \ldots, L-1\} \end{cases}, \\ \frac{\partial L}{\partial b_j^l} &= \delta_j^l \,, \\ \frac{\partial L}{\partial w_{jk}^l} &= \delta_j^l x_k^{l-1} \,. \end{aligned}$$

Here $\odot$ denotes the Hadamard product, which is simply a pointwise multiplication, and $\nabla_1 \ell$ is the gradient of ℓ with respect to the first argument.

Theorem 2.4 (Backpropagation)**.** *The gradient of the function* (2.32) *subject to the neural network constraint* (2.31) *with respect to the parameters $\{W_l\}_{l=1}^L$ and $\{b_l\}_{l=1}^L$ can be computed via the backpropagation Algorithm 3.*

As a consequence, we can train the network parameters via Algorithm 1 aka gradient descent where we use the gradient that we have computed with Algorithm 3.

2.9 Classification

For the remainder of this chapter we move on from regression problems to classification problems. Classification is the task of associating a certain class from a number of pre-defined classes to the input of a function. Suppose we have a set of s input and output samples $\{(x_i, y_i)\}_{i=1}^s$, then the goal of classification is to find a function $f : \mathbb{R}^d \to \{C_1, C_2, \ldots, C_n\}$ that approximately satisfies

$$f(x_i) \approx y_i \,,$$

for all $i \in \{1, \ldots, s\}$. You may say: hold on, this looks exactly like the general supervised learning formulation that we have introduced in (2.1) and this is correct! The only difference compared to supervised regression is that the function f no longer maps onto continuous values, but onto a discrete set of values $\{C_1, C_2, \ldots, C_n\}$. Here $\{C_j\}_{j=1}^n$ are the so-called *class labels* that are numerical values associated with the n individual classes. Note that each y_i has to take on one of

Algorithm 3 Backpropagation

```
Specify: Activation function $\sigma$, samples $\{(x_i, y_i)\}_{i=1}^s$, weight and bias dimensions, and no. of layers $L$
Iterate:
 1: for $i = 1, \dots, s$ do
 2:     for $l = 1, \dots, L$ do
 3:         Forward pass: compute $z_i^l = W_l^\top x_i^{l-1} + b_l$
 4:         Forward pass: compute $x_i^l = \sigma(z_i^l)$
 5:     end for
 6: end for
 7: for $i = 1, \dots, s$ do
 8:     for $l = L, \dots, 1$ do
 9:         Backward pass: compute $\delta_i^l = \begin{cases} \sigma'(z_i^L) \odot \frac{1}{s}\nabla_1 \ell(x_i^L, y_i) & l = L \\ \sigma'(z_i^l) \odot W_{l+1}\delta^{l+1} & l \in \{1, \dots, L-1\} \end{cases}$
10:     end for
11: end for
12: Partial derivatives: compute $\frac{\partial L}{\partial b_j^l} = \delta_j^l$, for all $j \in \{1, \dots, n_l\}$
13: Partial derivatives: compute $\frac{\partial L}{\partial w_{jk}^l} = \delta_j^l x_k^{l-1}$, for all $j \in \{1, \dots, n_l\}$ and $k \in \{1, \dots, n_{l-1}\}$.
return $\{W_l\}_{l=1}^L$ and $\{b_l\}_{l=1}^L$.
```

those values as well, i.e. $y_i \in \{C_1, \dots, C_n\}$ for all $i \in \{1, \dots, s\}$. If there are only two classes to map to, i.e. the range of the function is $\{C_1, C_2\}$, then we speak of a *binary classification* problem. For more than two classes we speak of a *multiclass classification* problem. In the following, we introduce a very basic classification method known as nearest neighbour classification, discuss its limitations and then continue to introduce other classification models such as logistic regression and support vector machines.

2.9.1 Nearest neighbour classification

One of the most simple classification ideas is to classify a new data sample x based on classes of the K-nearest neighbours of that sample in the training set. We can do this by assigning a probability to the unknown output label based on the labels of the K nearest neighbours. This probability is of the form

$$\rho(y = c \,|\, x, K) := \frac{1}{K} \sum_{l \in \mathcal{N}_K(x)} \iota(y_l = c)\,, \tag{2.33}$$

with ι defined as

$$\iota(z) := \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{if } z \text{ is false} \end{cases}. \tag{2.34}$$

Here $\mathcal{N}_K$ denotes the neighbourhood of x, which includes the K nearest neighbours of x. There are obviously many different ways of measuring distances between data points. One example is to simply measure the Euclidean distances between data points. In that case the neighbourhood can be defined as $\mathcal{N}_K(x) := \left\{x_{i(1)}, \dots, x_{i(K)} \;\middle|\; \|x - x_{i(j)}\| \leq \|x - x_l\|\,, \forall\, l \in \{1, \dots, s\} \setminus \{i(1), \dots, i(K)\}\right\}$.

The definition (2.34) assigns a probability to the event that a label y equals a class label $c \in \{C_0, \ldots, C_n\}$. This is done by computing the averages among the K nearest neighbours with identical class label. You can find particular examples in the corresponding video lecture that accompanies these lecture notes. Once we have computed all probabilities for the different class labels, we assign the class label with the highest probability to the output of our classifier f, i.e.

$$f(x) := \underset{c \in \{C_0, C_1, \ldots, C_n\}}{\arg\max} \rho(y = c \,|\, x, K) \,. \tag{2.35}$$

The entire strategy is known as the *K-nearest neighbours classification*. The number of neighbours K is a hyperparameter that has to be determined with model selection strategies such a cross validation.

In the following we want to shed some light on why nearest neighbour classification works well for lower dimensional problems (small d) but fails to deliver meaningful results when the dimension d increases.

The curse of dimensionality

Nearest neighbour classification is a very simple classification strategy that, unfortunately, falls victim to the so-called *curse of dimensionality*. In the context of supervised machine learning, the curse of dimensionality usually takes the following form.

(a) "Generalising correctly becomes exponentially harder as the dimensionality grows because fixed-size training sets cover a dwindling fraction of the input space."

(b) In high-dimensions, data-points are far from each other. Consequently, "as the dimensionality increases, the choice of nearest neighbour becomes effectively random."

These quotes are taken from Pedro Domingos review article "A few useful things to know about machine learning". In the following, we want to dissect what those claims mean mathematically. For the first claim, imagine that we have m data inputs $\{x_i\}_{i=1}^m$ that all lie in the d-dimensional unit cube $[0, 1]^d$ with a total volume of $1^d = 1$. Now we consider a sub-cube $[a, a+r]^d \subset [0, 1]^d$ with $0 \leq a$ and $a+r \leq 1$ with volume $r^d \leq 1$. This sub-cube can be seen as our training set, containing s training samples. The question we want to address here is the following: how large does this cube have to be in order to cover a certain fraction α of the m data samples (in expectation)? In other words: how do we need to choose the length r of the sub-cube such that $\alpha = r^d$? The straight-forward answer to this question is

$$r = \sqrt[d]{\alpha} \,.$$

If we are in a $d = 10$ dimensional space for example, covering only $\alpha = 1\%$ of the data already requires a cube with length $r \approx 0.63$. To cover $\alpha = 10\%$ of the data in the overall cube, a length of $r \approx 0.8$ is required. In other words, for a fixed sub-cube with fixed length $r < 1$ increasing the dimension d dramatically reduces the fraction of data samples that the sub-cube covers.

For the second claim, we consider the d-dimensional unit cube again, but this time our sub-cube is located around the centre point $(\frac{1}{2}, \frac{1}{2}, \ldots, \frac{1}{2})$, i.e. we have $[(1-r)/2, (1+r)/2]^d \subset [0, 1]^d$ with $0 \leq r \leq 1$. Suppose m input samples $\{x_i\}_{i=1}^s$ are uniformly distributed in the cube $[0, 1]^d$. What is the chance that a random sample is in the overall cube $[0, 1]^d$ but not in the sub-cube $[(1-r)/2, (1+r)/2]^d$? This chance is simply $1 - r^d$. Considering m i.i.d. samples, the chance that none of these m samples is in the sub-cube then becomes

$$\left(1 - r^d\right)^m \,.$$

For a fixed probability ρ, we can solve this equation for r, i.e.

$$r = \sqrt[d]{1 - \sqrt[m]{\rho}}\,.$$

To have a probability of 50% (which means $\rho = 0.5$) that none of $m = 500$ samples in a $d = 10$ dimensional space is in the sub-cube, the length of the sub-cube only has to be $r \approx 0.52$. Again, keeping r fixed and increasing the dimension d will quickly increase the probability ρ to values close to 100%, rendering the concept of nearest neighbours useless in higher dimensions.

All these considerations tell us that the K-nearest neighbours classification strategy suffers from the curse of dimensionality, which is why we have to consider other classification strategies that do not rely on the concept of neighbours. In the following sections we will consider three alternative classification strategies.

2.9.2 Logistic regression

One could think of solving classification problems with the same tools that we have used for tackling regression problems, i.e. mean-squared-error regression. After all, the only difference is that the output of the trained prediction function $f(x, w)$ is supposed to be discrete and not continuous, which we could achieve by thresholding the function output. However, as you find out in the weekly lecture videos, this strategy is not working in practice, since the mean-squared error is not really related to the objective of minimising the number of misclassified samples. Starting with the concept of binary classification, it seems reasonable to transform the prediction $f(x, w)$ into a probability. In order, to do so, we consider $\sigma(f(x, w))$ instef $f(x, w)$, where $\sigma : (-\infty, \infty) \to [0, 1]$ is the so-called *logistic function* defined as

$$\sigma(z) := \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z}\,. \tag{2.36}$$

We can then define the following probabilities for the events that the output $f(x, w)$ belongs the class with class label zero or the class with class label one:

$$\rho(1|x) := \sigma(f(x, w))\,, \tag{2.37a}$$
$$\rho(0|x) := 1 - \sigma(f(x, w))\,. \tag{2.37b}$$

From the definition of σ we instantly observe $\rho(1|x) \geq 0$, $\rho(0|x) \geq 0$ and $\rho(1|x) + \rho(0|x) = 1$. Hence, when we speak of ρ we can indeed speak of a probability. Now assume we have s pairs of samples $\{(x_i, y_i)\}_{i=1}^s$ where $y_i \in \{0, 1\}$ for all $i \in \{1, \ldots, s\}$ that are iid and follow the probability density function that we have just defined in (2.37). The corresponding likelihood then reads

$$\rho\,(y \mid X, w) = \prod_{i=1}^{s} \rho(y_i|x_i)\,, \tag{2.38}$$

where X and y are abbreviations of the matrix and vector that we obtain from the samples $\{y_i\}_{i=1}^s$ and $\{x_i\}_{i=1}^s$. Note that we can rewrite (2.38) as

$$\begin{aligned}
\rho\,(y \mid X, w) &= \prod_{\{i \,|\, y_i = 1\}} \rho(y_i = 1|x_i) \prod_{\{i \,|\, y_i = 0\}} \rho(y_i = 0|x_i)\,, \\
&= \prod_{i=1}^{s} \sigma(f(x_i, w))^{y_i} \left(1 - \sigma(f(x_i, w)\right)^{1 - y_i}\,.
\end{aligned}$$

As in the regression case with the normal distribution, we can obtain parameters $\hat{w}$ that maximise the likelihood (2.38) by minimising the negative log-likelihood, i.e.

$$\begin{aligned}
\hat{w} &= \arg\min_w \left\{ -\log\left(\rho\left(y \mid X, w\right)\right)\right\} ,\\
&= \arg\min_w \left\{ -\log\left(\prod_{i=1}^s \sigma(f(x_i, w))^{y_i} \left(1 - \sigma(f(x_i, w))\right)^{1-y_i}\right)\right\} ,\\
&= \arg\min_w \left\{ -\sum_{i=1}^s \left[y_i \log\left(\sigma(f(x_i, w))\right) + (1 - y_i) \log\left(\left(1 - \sigma(f(x_i, w))\right)\right)\right]\right\} ,\\
&= \arg\min_w \left\{ \sum_{i=1}^s \left[\log\left(1 + \exp\left(f(x_i, w)\right)\right) - y_i f(x_i, w)\right]\right\} .
\end{aligned}$$

This alternative form of regression is known as *logistic regression* as it is based on the logistic function (2.36). We can choose any model function f – linear or polynomial basis function, or even a neural network – that we like, as long as it maps onto the real numbers. To determine a unique minimiser, a model linear with respect to the weights w has its advantages, as we will discuss later. Before we discuss how to compute an argument that minimises this expression numerically, we want to discuss how to extend logistic regression to classification problems with more than two classes first.

In order to derive a logistic regression problem that can deal with more than two classes, we need to come up with a probability model that can associate the highest probability to the label that corresponds to the correct class. We can do this with the help of the so-called *softmax*-function. Assume for $K > 2$ classes that we have a model function $f(x, w_1, \ldots, w_K)$ that depends on multiple weight vectors $\{w_k\}_{k=1}^K$, and more importantly, that maps onto a K-dimensional vector rather than a scalar output. Given such a function, we compose it with the softmax function $\sigma : \mathbb{R}^K \to [0, 1]^K$ that is defined as

$$\sigma(z_1, z_2, \ldots, z_K)_k := \frac{\exp(z_k)}{\sum_{j=1}^K \exp(z_j)} , \qquad \forall\, k \in \{1, \ldots, K\} .$$

We can then associate a probability for each class based on this composition, i.e.

$$\rho(y_i = k \,|\, x_i, w_1, \ldots, w_K) := \sigma\left(f(x_i, w_1, \ldots, w_K)\right)_k = \frac{\exp(f(x_i, w_1, \ldots, w_K)_k)}{\sum_{j=1}^K \exp(f(x_i, w_1, \ldots, w_K)_j)} ,$$

for all $k \in \{1, \ldots, K\}$. By definition of the softmax-function, we observe $\rho(y_i = k \,|\, x_i, w_1, \ldots, w_K) \geq 0$ for all $k \in \{1, \ldots, K\}$ as well as $\sum_{k=1}^K \rho(y_i = k \,|\, x_i, w_1, \ldots, w_K) = 1$; hence, we can talk of ρ as a probability. We can then proceed as in the binary logistic regression case and define a likelihood for s data samples $\{(x_i, y_i)\}_{i=1}^s$ via

$$\rho(\hat{y} = y \,|\, X, W) := \prod_{i=1}^s \rho(\hat{y}_i = y_i \,|\, x_i, w_1, \ldots, w_K) ,$$

for the short-hand notations $y = (y_1, \ldots, y_s)^\top$, $X = \begin{pmatrix} x_1 & \ldots & x_s \end{pmatrix}^\top$ and $W = \begin{pmatrix} w_1 & w_2 & \ldots & w_K \end{pmatrix}$. We can simplify this likelihood to

$$\rho(\hat{y} = y \,|\, X, W) = \prod_{\{i \,|\, y_i = 1\}} \rho(\hat{y}_i = 1 \,|\, x_i, w_1, \ldots, w_K) \cdots \prod_{\{i \,|\, y_i = K\}} \rho(\hat{y}_i = K \,|\, x_i, w_1, \ldots, w_K) ;$$

We can use the indicator function $\mathbf{1}_{y_i=k}$ defined as

$$\mathbf{1}_{y_i=k} := \begin{cases} 1 & y_i = k \\ 0 & \text{otherwise} \end{cases}$$

to further simplify the likelihood to

$$\rho(\hat{y} = y \,|\, X, W) := \prod_{i=1}^{s} \prod_{k=1}^{K} \rho(\hat{y}_i = k \,|\, x_i, w_1, \ldots, w_K)^{\mathbf{1}_{y_i=k}} \, .$$

As before, we can maximise this likelihood by choosing the parameters W such that they minimise the negative log-likelihood, i.e.

$$\begin{aligned}
\hat{W} &= \arg\min_W -\log\left(\rho(\hat{y} = y \,|\, X, W)\right) \, , \\
&= \arg\min_W -\log\left(\prod_{i=1}^{s}\prod_{j=1}^{K} \rho(\hat{y}_i = k \,|\, x_i, w_1, \ldots, w_K)^{\mathbf{1}_{y_i=k}}\right) \, , \\
&= \arg\min_W -\sum_{i=1}^{s}\sum_{k=1}^{K} \mathbf{1}_{y_i=k} \log\left(\rho(\hat{y}_i = k \,|\, x_i, w_1, \ldots, w_K)\right) \, , \\
&= \arg\min_W -\sum_{i=1}^{s}\sum_{k=1}^{K} \mathbf{1}_{y_i=k} \log\left(\frac{\exp(f(x_i, w_1, \ldots, w_K)_k)}{\sum_{j=1}^{K} \exp(f(x_i, w_1, \ldots, w_K)_j)}\right) \, , \\
&= \arg\min_W \sum_{i=1}^{s}\sum_{k=1}^{K} \mathbf{1}_{y_i=k} \left(\log\left(\sum_{j=1}^{K} \exp(f(x_i, w_1, \ldots, w_K)_j)\right) - f(x_i, w_1, \ldots, w_K)_k\right) \, , \\
&= \arg\min_W \sum_{i=1}^{s}\sum_{k=1}^{K} \mathbf{1}_{y_i=k} \log\left(\sum_{j=1}^{K} \exp(f(x_i, w_1, \ldots, w_K)_j)\right) - \sum_{i=1}^{s}\sum_{k=1}^{K} \mathbf{1}_{y_i=k} f(x_i, w_1, \ldots, w_K)_k \, , \\
&= \arg\min_W \sum_{i=1}^{s} \log\left(\sum_{k=1}^{K} \exp(f(x_i, w_1, \ldots, w_K)_k)\right) - \sum_{i=1}^{s}\sum_{k=1}^{K} \mathbf{1}_{y_i=k} f(x_i, w_1, \ldots, w_K)_k \, .
\end{aligned}$$

Hence, with the more compact notation $f(x, W)$ we can estimate optimal parameters $\hat{W}$ via

$$\hat{W} = \arg\min_W \sum_{i=1}^{s} \log\left(\sum_{k=1}^{K} \exp(f(x_i, W)_k)\right) - \sum_{i=1}^{s}\sum_{k=1}^{K} \mathbf{1}_{y_i=k} f(x_i, W)_k \, .$$

A typical choice for f based on polynomial augmentation would be

$$\begin{aligned}
f(x, W) &= \begin{pmatrix} \langle \phi(x), w_1 \rangle & \langle \phi(x), w_2 \rangle & \ldots & \langle \phi(x), w_K \rangle \end{pmatrix} \\
&= \phi(x)^\top W \, .
\end{aligned}$$

If we store all input samples $\{x_i\}_{i=1}^{s}$ in a matrix $X = \begin{pmatrix} x_1 & \ldots & x_s \end{pmatrix}^\top$, we can write

$$f(X, W) = \Phi(X) W$$

in matrix form. The *multinomial logistic regression* problem then reads

$$\hat{W} = \operatorname*{arg\,min}_{W \in \mathbb{R}^{(1+d)\times K}} \sum_{i=1}^{s} \log \left(\sum_{k=1}^{K} \exp(\langle \phi(x_i), w_k \rangle) \right) - \sum_{i=1}^{s} \sum_{k=1}^{K} \mathbf{1}_{y_i = k} \langle \phi(x_i), w_k \rangle \, ,$$

or

$$\hat{W} = \operatorname*{arg\,min}_{W \in \mathbb{R}^{(1+d)\times K}} \sum_{i=1}^{s} \log \left(\sum_{k=1}^{K} \exp((\Phi(X)W)_{ik}) \right) - \sum_{i=1}^{s} \sum_{k=1}^{K} \mathbf{1}_{y_i = k} (\Phi(X)W)_{ik} \rangle$$

in matrix form. The key question that remains for both the binary and the multinomial logistic regression problem is: how do we solve these minimisation problems computationally?

We will focus on the binary logistic regression case with polynomial data model $f(x, w) = \langle \phi(x), w \rangle$; the multinomial logistic regression case can be covered in almost identical fashion, just with different computations. The first thing that we observe is that the objective- (or cost-)function

$$L(w) := \sum_{i=1}^{s} \left[\log\left(1 + \exp\left(\langle \phi(x_i), w \rangle\right)\right) - y_i \langle \phi(x_i), w \rangle \right] \tag{2.39}$$

is differentiable. So can we maybe compute the gradient, set the gradient to zero and solve for the weights w as we did for the mean-squared error regression? Computing the partial derivative of L with respect to w_l, for $l \in \{0, \ldots, d\}$, yields

$$\begin{aligned} \frac{\partial L}{\partial w_l} &= \sum_{i=1}^{s} \phi(x_i)_l \left(\sigma\left(\langle \phi(x_i), w \rangle \right) - y_i \right) \, , \\ &= \sum_{i=1}^{s} \Phi(X)_{il} \left(\sigma\left((\Phi(X)w)_i \right) - y_i \right) \, . \end{aligned}$$

Here σ denotes the logistic function as defined in (2.36). Hence, the entire gradient of L reads

$$\nabla L(w) = \Phi(X)^\top \left(\sigma(\Phi(X)w) - y \right)$$

in column-vector form. Here, $\sigma(\Phi(X)w)$ is short-hand notation for applying the logistic function point-wise to every component of the vector $\Phi(X)w$. Setting the gradient to zero and solving it for the corresponding argument $\hat{w}$ would therefore require the solution of the equation

$$0 = \Phi(X)^\top \left(\sigma(\Phi(X)\hat{w}) - y \right) \, . \tag{2.40}$$

Because of the non-linearity of σ, we cannot simply solve (2.40) for $\hat{w}$ and a fixed but arbitrary matrix $\Phi(X)$. Having computed the gradient, we can, however, try to approximate a solution of

$$\hat{w} = \operatorname*{arg\,min}_{w \in \mathbb{R}^{1+d}} \left\{ L(w) \right\} \tag{2.41}$$

via gradient descent, as defined in Algorithm 1. For this particular choice of cost function L, the gradient descent iterate becomes

$$w^{k+1} = w^k - \tau \, \Phi(X)^\top \left(\sigma(\Phi(X)w^k) - y \right) \, .$$

The question that we need to address is whether gradient descent will convergence to a solution of (2.41). Note that in order to apply Theorem 2.3, we only need to verify convexity of L and $1/\tau$-smoothness.

Corollary 2.2 (Second order convexity)**.** *A twice differentiable one-dimensional function $f : \mathcal{C} \to \mathbb{R}$ over a convex set $\mathcal{C} \subset \mathbb{R}$ is convex if and only if $f''(x) \geq 0$ for all $x \in \mathcal{C}$.*

With the previous corollary we can verify that the logistic regression function is convex.

Lemma 2.6 (Convexity of the binary logistic regression problem)**.** *The function $L : \mathbb{R}^{1+d} \to \mathbb{R}$ as defined in Equation* (2.39) *is convex.*

Proof. We basically only need to show that the function $f(z) := \log(1 + \exp(z))$ is convex; then we could conclude that L is a sum of convex functions (and therefore convex itself), as linear functions are convex and since compositions of convex functions and linear functions are convex. We verify that f is convex by showing that $f''(z) \geq 0$ for all $z \in \mathbb{R}$. We had already computed the first derivative that reads

$$f'(z) = \frac{1}{1 + \exp(-z)} = \sigma(z) \, ;$$

It can be easily verified that the second derivative reads

$$f''(z) = \sigma(z) \left(1 - \sigma(z)\right) \, .$$

Since $\sigma(z) \in [0, 1]$ for all $z \in \mathbb{R}$, we immediately see that $f''(z) \geq 0$ for all z. Hence, f is convex and as a consequence, L is convex. □

In order to converge successfully to a minimiser of the logistic regression problem with gradient descent, we only need to specify a step-size $\tau > 0$ that is sufficiently small so that the objective values decrease monotonically (if such a step-size exists). Without proof, we use that the logistic function σ is a $\frac{1}{4}$-Lipschitz continuous function, i.e.

$$|\sigma(x) - \sigma(y)| \leq \frac{1}{4}|x - y| \, ,$$

for all $x, y \in \mathbb{R}$. Then we can conclude that the gradient $\nabla L(w)$ is $1/\tau$-Lipschitz continuous, i.e.

$$\begin{aligned} \|X^\top (\sigma(Xw) - y) - X^\top (\sigma(Xv) - y)\| &= \|X^\top (\sigma(Xw) - \sigma(Xv))\| \, , \\ &\leq \|X\| \|\sigma(Xw) - \sigma(Xv)\| \, , \\ &\leq \frac{\|X\|}{4} \|Xw - Xv\| \, , \\ &\leq \frac{\|X\|^2}{4} \|w - y\| \, , \end{aligned}$$

for $\tau = 4/\|X\|^2$ and all $w, v \in \mathbb{R}^{1+d}$. Hence, gradient descent is guaranteed to converge to a global minimum of L for $\tau < 4/\|X\|^2$ as a consequence of Lemma 2.3 and Theorem 2.3.

2.9.3 Support-vector machines (SVMs)

One limitation of logistic regression is that the hyperplane that spans the decision boundary is not necessarily optimal in the sense that it maximises the distance between the closest data points on each side of the decision boundary. This feature can, however, be achieved in the context of binary classification with *Support Vector Machines (SVMs)* that utilise a different data model. For a more detailed motivation we refer to the lecture videos & slides. For a set of data points $\{(x_i, y_i)\}_{i=1}^s$, with $y_i \in \{-1, 1\}$ for all $i \in \{1, \ldots, s\}$, and a linear model function $f(x, w)$ with parameters $w \in \mathbb{R}^{1+d}$, the key idea is to maximise the distance r of the closest data points to the hyper-plane, but to ensure that the each data point ends up on the correct side of the decision boundary at the same time. Mathematically, for a linear model $f(x, w) = \langle \phi(x), w \rangle$ this distance r can be characterised via

$$r = \frac{f(x, w)}{\|w\|} ,$$

while ensuring that each data point is on the correct side of the decision boundary can mathematically be described as the constraint

$$y_i \, f(x_i, w) - 1 \geq 0 ,$$

for all $i \in \{1, \ldots, s\}$. In order to maximise r, we can minimise $\|w\|$ (or $\|w\|^2$) subject to the previous constraint, i.e.

$$\min_{w_0, w_2, \ldots, w_d} \|w\|^2 \qquad \text{subject to} \qquad y_i \, f(x_i, w) - 1 \geq 0 , \quad \forall i \in \{1, \ldots, s\} . \tag{2.42}$$

If the data points cannot be separated linearly, Problem (2.42) doesn't have a solution. To overcome this limitation, we can relax Problem (2.42) to

$$\min_{w \in \mathbb{R}^{1+d}} \sum_{i=1}^{s} \max\left(0, 1 - y_i \langle x_i, w \rangle\right) + \frac{\alpha}{2} \|w\|^2 , \tag{2.43}$$

for $w = \begin{pmatrix} w_0 & w_2 & \ldots & w_d \end{pmatrix}^\top$ and a balancing (or regularisation) parameter $\alpha > 0$. The solution to Problem (2.43) is known as the *soft-margin SVM*; the solution to Problem (2.42) as *hard-margin SVM*. In the following we want to discuss how to simplify (2.43) to make it computationally more tractable.

We follow a similar trick as in Section 2.7.2 where we have reformulated the absolute value function in terms of a dual variable. We can do the same trick for the ramp function $\max(0, z)$, i.e. we can write $\max(0, z)$ as

$$\max(0, z) = \max_{\lambda \in [0,1]} \lambda z .$$

Replacing the ramp function in (2.43) with this dual characterisation yields the min-max problem

$$\min_{w \in \mathbb{R}^{1+d}} \left\{ \max_{\lambda \in [0,1]^s} \sum_{i=1}^{s} \lambda_i \left(1 - y_i \langle x_i, w \rangle\right) + \frac{\alpha}{2} \|w\|^2 \right\} . \tag{2.44}$$

Note that the underlying function $L(w, \lambda)$ defined as

$$L(w, \lambda) := \sum_{i=1}^{s} \lambda_i \left(1 - y_i \langle x_i, w \rangle\right) + \frac{\alpha}{2} \|w\|^2 - \chi_{[0,1]^s}(\lambda)$$

with

$$\chi_{[0,1]^s}(\lambda) = \begin{cases} 0 & \forall i \in \{1, \ldots, s\} \, : \, \lambda_i \in [0, 1] \\ \infty & \exists i \in \{1, \ldots, s\} \, : \, \lambda_i \notin [0, 1] \end{cases} ,$$

is convex in the first argument (for fixed second argument) and concave in the second argument (for fixed first argument), where convexity and concavety are defined as in Definition 2.2, respectively Definition 2.3. This statement is left as an exercise to the curious reader. The following theorem states that it makes no difference for such functions whether one first maximises and then minimises the function, or if one first minimises and then maximises the function.

Theorem 2.5 (Minimax Theorem, von Neumann 1928)**.** *Let $X \subset \mathbb{R}^m$ and $Y \subset \mathbb{R}^n$ be compact, convex sets. If $f : X \times Y \to \mathbb{R}$ is a continuous function that is convex-concave, i.e.*

$$f(\cdot, y) : X \to \mathbb{R} \textit{ is convex for fixed } y\,,$$
$$f(x, \cdot) : Y \to \mathbb{R} \textit{ is concave for fixed } x\,.$$

Then the max-min inequality is an equality, i.e.

$$\min_{x\in X} \max_{y\in Y} f(x, y) = \max_{y\in Y} \min_{x\in X} f(x, y)\,.$$

Thanks to Theorem 2.5, we can replace (2.44) with the equivalent problem

$$\max_{\lambda\in[0,1]^s} \left\{ \min_{w\in\mathbb{R}^{1+d}} \sum_{i=1}^{s} \lambda_i \left(1 - y_i\langle x_i, w\rangle\right) + \frac{\alpha}{2}\|w\|^2 \right\}\,. \tag{2.45}$$

What is the advantage of (2.45) over (2.44), you may ask? The advantage is that (2.44) can be solved more easily, as the inner problem becomes differentiable. The gradient of L with respect to w reads

$$\nabla_w L(w, \lambda) = -\sum_{i=1}^{s} \lambda_i y_i x_i + \alpha w\,.$$

Computing $\hat{w}$ with $\nabla_w L(\hat{w}, \lambda) = 0$ yields

$$\hat{w} = \frac{1}{\alpha} \sum_{i=1}^{s} \lambda_i y_i x_i\,. \tag{2.46}$$

Inserting $\hat{w}$ into $L(w, \lambda)$ then turns (2.45) into

$$\max_{\lambda\in[0,1]^s} \left\{ \langle \lambda, \mathbf{1}\rangle - \frac{1}{2\alpha}\|X^\top Y\lambda\|^2 \right\}\,. \tag{2.47}$$

Here Y is short-hand notation for the $s \times s$ diagonal matrix that contains the elements of the vector y on its diagonal, i.e.

$$Y = \operatorname{diag}(y) := \begin{pmatrix} y_1 & 0 & 0 & \cdots & 0 \\ 0 & y_2 & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & 0 & \cdots & y_s \end{pmatrix}\,,$$

and $\mathbf{1}$ is short-hand-notation for the s-dimensional vector of ones, i.e. $\mathbf{1} = \begin{pmatrix}1 & 1 & 1 & \dots & 1\end{pmatrix}^\top \in \mathbb{R}^s$. We can reformulate (2.47) equivalently as minimisation problem; hence, computing the argument $\hat{\lambda}$ that minimises this expression is

$$\hat{\lambda} = \operatorname*{arg\,min}_{\lambda\in[0,1]} \left\{ \frac{1}{2\alpha}\|X^\top Y\lambda\|^2 - \langle \lambda, \mathbf{1}\rangle \right\}\,. \tag{2.48}$$

This convex problem can be solved computationally with various algorithms; examples include the proximal gradient method as described in Algorithm 2 and coordinate descent. Both methods utilise proximal maps, which in this case reduce to the orthogonal projection onto the convex set $[0, 1]$, i.e.

$$\left((I + \partial\chi_{[0,1]^s})^{-1}(z)\right)_i = \min(1, \max(0, z_i))\,,$$

for all $i \in \{1, \dots, s\}$. Note that by solving (2.48) we automatically find the argument that minimises the original soft-margin SVM problem (2.43) via (2.46).

2.9.4 Semi-supervised binary classification with graphs

In this section we discuss how to model semi-supervised classification problems with the help of undirected, weighted graphs. An undirected, weighted graph is defined as follows.

Definition 2.8. *An undirected* graph *G is a pair $G = (V, E)$, where V is a set of elements called* vertices*, and $E = \{x, y \,|\, (x, y) \in V^2 \wedge x \neq y\}$ is a set of* edges*. A* weighted graph *(or* network*) is a graph in which a number, known as* weight*, is assigned to each edge.*

An example of an undirected, weighted graph can be seen in Figure 2.2.

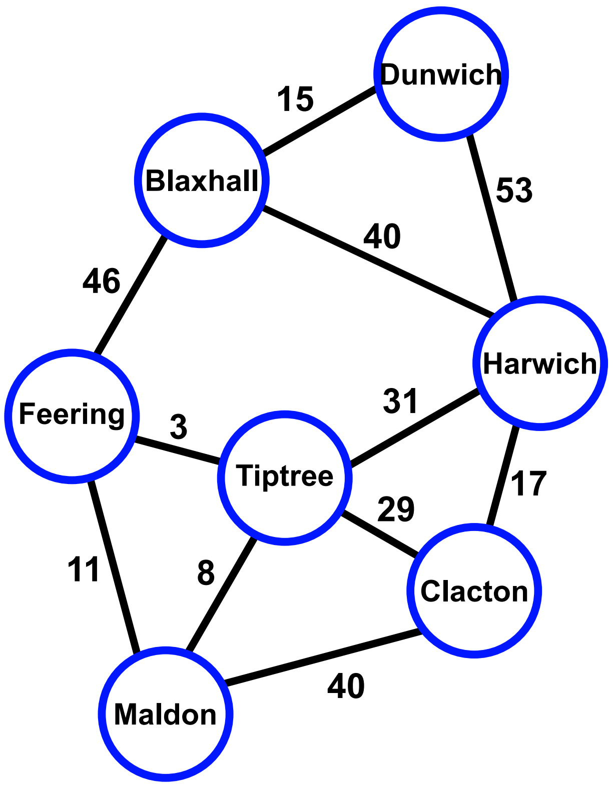


Figure 2.2: An example of a weighted graph of several connected towns in the south east of England. This graph has seven vertices (or nodes), representing different towns, connected by eleven edges. The weights represent the distances between the connected towns. ©Wikimedia commons.

For a (weighted) graph (with weights w) we define a so-called incidence matrix $M_w \in \mathbb{R}^{|E| \times |V|}$, where $|E|$ denotes the number of edges and $|V|$ the number of vertices, as

$$(M_w)_{ev} := \begin{cases} -\sqrt{w_{ev}} & \text{if } v = i \\ \sqrt{w_{ev}} & \text{if } v = j \\ 0 & \text{otherwise} \end{cases},$$

where every edge $e = (i, j)$ connects vertices i and j, with $i < j$. The corresponding *graph-*

Laplacian $L_w \in \mathbb{R}^{|V|\times|V|}$ is then defined as

$$L_w := M_w^\top M_w \, .$$

It is always best to give a concrete example for such an incidence matrix as well as the graph-Laplacian.

Example 2.5. The incidence matrix for the graph in Figure 2.2 reads

$$M_w = \begin{pmatrix} -\sqrt{15} & \sqrt{15} & 0 & 0 & 0 & 0 & 0 \\ -\sqrt{53} & 0 & \sqrt{53} & 0 & 0 & 0 & 0 \\ 0 & -\sqrt{40} & \sqrt{40} & 0 & 0 & 0 & 0 \\ 0 & -\sqrt{46} & 0 & 0 & \sqrt{46} & 0 & 0 \\ 0 & 0 & 0 & -\sqrt{3} & \sqrt{3} & 0 & 0 \\ 0 & 0 & -\sqrt{31} & \sqrt{31} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\sqrt{29} & 0 & \sqrt{29} & 0 \\ 0 & 0 & -\sqrt{17} & 0 & 0 & \sqrt{17} & 0 \\ 0 & 0 & 0 & 0 & -\sqrt{11} & 0 & \sqrt{11} \\ 0 & 0 & 0 & -\sqrt{8} & 0 & 0 & \sqrt{8} \\ 0 & 0 & 0 & 0 & 0 & -\sqrt{40} & \sqrt{40} \end{pmatrix} ,$$

while the corresponding graph Laplacian is then given as

$$L_w = M_w^\top M_w = \begin{pmatrix} 68 & -15 & -53 & 0 & 0 & 0 & 0 \\ -15 & 101 & -40 & 0 & -46 & 0 & 0 \\ -53 & -40 & 141 & -31 & 0 & -17 & 0 \\ 0 & 0 & -31 & 71 & -3 & -29 & -8 \\ 0 & -46 & 0 & -3 & 60 & 0 & -11 \\ 0 & 0 & -17 & -29 & 0 & 86 & -40 \\ 0 & 0 & 0 & -8 & -11 & -40 & 59 \end{pmatrix} .$$

The graph Laplacian can also be decomposed into a so-called degree matrix D_w and an adjacency matrix A_w via $L_w = M_w - A_w$, which we in the interest of time will not explore any further throughout this lecture.

We can use weighted graphs to model and exploit similarities in datasets. Suppose we are given a set $S := \{x_i\}_{i=1}^s$ of s samples, for which only $r \ll s$ samples in the set $R := \{x_{i(j)}\}_{j=1}^r$ have corresponding (binary) output labels $\{y_j\}_{j=1}^r$ with values in $\{0, 1\}$. Here $i : \{1, \ldots, r\} \to \{1, \ldots, s\}$ denotes an index function that picks the indices for which labels are known. If r is very small, supervised learning on just r samples may not lead to mappings that have satisfactory predictive powers when applied to the classification of new samples x. However, we can assume that all data points $\{x_i\}_{i=1}^s$ are nodes in a connected graph. The weights between each nodes are determined by a similarity measure, for example of the form

$$w_{ij} = \begin{cases} \exp\left(-\gamma \|x_i - x_j\|^2\right) & \|x_i - x_j\| \leq \text{threshold} \\ 0 & \|x_i - x_j\| > \text{threshold} \end{cases} ,$$

for a constant $\gamma > 0$ and a threshold value that guarantees that not all weights are non-zero. This way we create an undirected weighted graph with connections between nodes where there is similarity in terms of the Euclidean norm. Based on this graph with weights w, we can construct a corresponding incidence matrix M_w. This type of supervised machine learning is usually referred to as semi-supervised, as we only require a smaller subset R of a set of (training) samples S instead of the entire set of training samples in order to perform the binary classification task.

One could now try to pursue a supervised classification task by solving the following optimisation problem:

$$\hat{v} = \operatorname*{arg\,min}_{v \in [0,1]^s} \left\{ \|M_w v\|^2 \,\text{subject to}\, (P_R v)_j = y_j \,, \text{ for all}\, j \in \{1, \dots, r\} \right\} . \tag{2.49}$$

Here $P_R : \mathbb{R}^s \to \mathbb{R}^r$ denotes the projection of a vector on the indices specified by the index function i, i.e.

$$(P_R v)_j = v_{i(j)} \,, \qquad \forall j \in \{1, \dots, r\} \,.$$

The label vector $\hat{v}$ is constrained to have values in $[0,1]^s$ and to take on the correct values for the indices with corresponding output labels. The remaining values are determined by ensuring minimal $\|M_w \hat{v}\|^2$ for $\hat{v}$ amongst all possible label vectors $v \in \mathbb{R}^s$.

Note that given the sets S and R, we can easily define the complement $R^\perp := S \setminus R$ of R. If we denote the projection onto this set with $P_{R^\perp} : \mathbb{R}^s \to \mathbb{R}^{s-r}$, we can rewrite v as

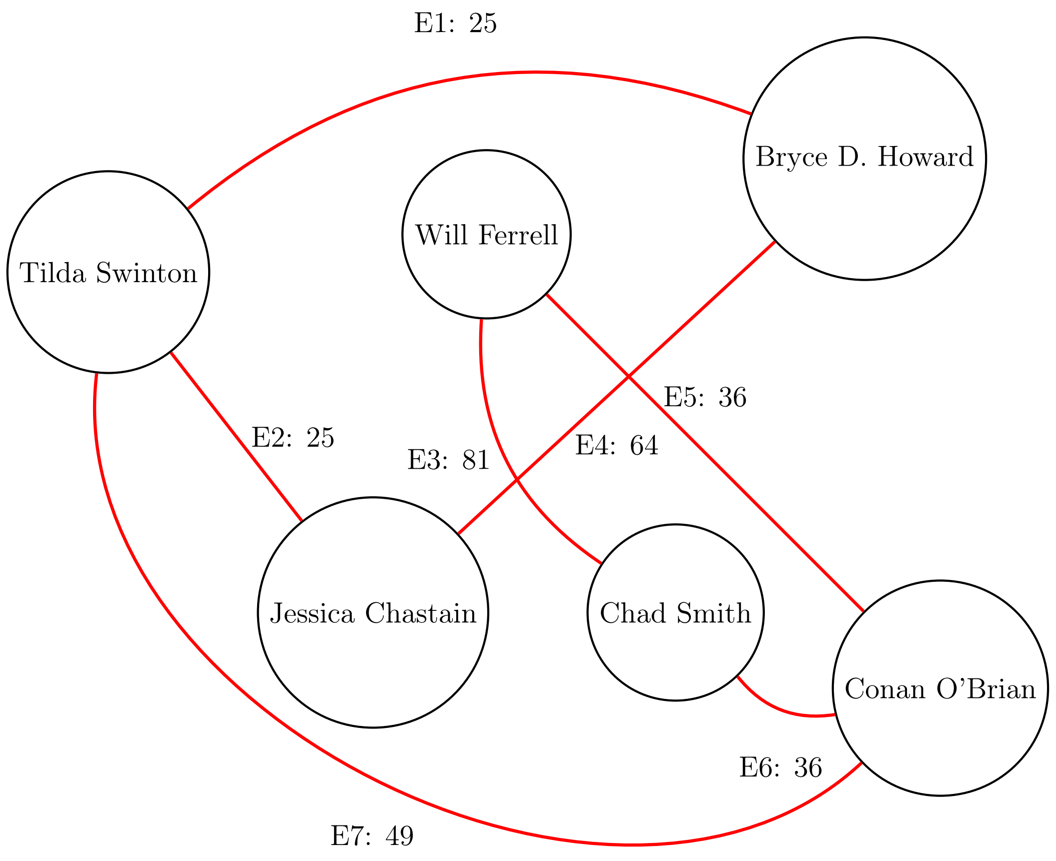
$$v = P_{R^\perp}^\top P_{R^\perp} v + P_R^\top P_R v = P_{R^\perp}^\top P_{R^\perp} v + P_R^\top y \,.$$

Note that we only need to compute $P_{R^\perp} \hat{v}$ instead of $\hat{v}$ as $P_R \hat{v} = y$ is already known. As a direct consequence, we can reformulate (2.49) to

$$\begin{aligned} P_{R^\perp} \hat{v} &= \operatorname*{arg\,min}_{\tilde{v} \in [0,1]^{s-r}} \left\{ \left\| M_w \left(P_{R^\perp}^\top \tilde{v} + P_R^\top y \right) \right\|^2 \right\} \\ &= \operatorname*{arg\,min}_{\tilde{v} \in \mathbb{R}^{s-r}} \left\{ \left\| M_w \left(P_{R^\perp}^\top \tilde{v} + P_R^\top y \right) \right\|^2 \right\} \,, \end{aligned} \tag{2.50}$$

where the last equality holds when $y \in [0,1]^r$. In the following, we want to give a small example to illustrate this approach of binary classification.

Example 2.6. We consider the following graph where its weights mimic a similarity measure between the individuals in the nodes:



Wherever there is no edge between two nodes, the corresponding weight, respectively the similarity measure, is zero. Computing the incidence matrix M_w of this weighted graph yields

$$\left(\begin{array}{c|cccccc}
\text{E1} & -5 & 0 & 0 & 0 & 5 & 0 \\
\text{E2} & 0 & 0 & 0 & -5 & 5 & 0 \\
\text{E3} & 0 & -9 & 0 & 0 & 0 & 9 \\
\text{E4} & -8 & 0 & 0 & 8 & 0 & 0 \\
\text{E5} & 0 & 0 & -6 & 0 & 0 & 6 \\
\text{E6} & 0 & -6 & 6 & 0 & 0 & 0 \\
\text{E7} & 0 & 0 & -7 & 0 & 7 & 0 \\
\hline
 & \text{B. D. Howard} & \text{C. Smith} & \text{C. O' Brian} & \text{J. Chastain} & \text{T. Swinton} & \text{W. Ferrell}
\end{array}\right).$$

The corresponding graph Laplacian reads

$$L_w = M_w^\top M_w = \begin{pmatrix} 89 & 0 & 0 & -64 & -25 & 0 \\ 0 & 117 & -36 & 0 & 0 & -81 \\ 0 & -36 & 121 & 0 & -49 & -36 \\ -64 & 0 & 0 & 89 & -25 & 0 \\ -25 & 0 & -49 & -25 & 99 & 0 \\ 0 & -81 & -36 & 0 & 0 & 117 \end{pmatrix}.$$

Suppose we want to classify each node according to biological sex, and already know that Jessica Chastain is female (class label $\hat{v}_4 = 1$) and Will Ferrell is male (class label $\hat{v}_6 = 0$). We can therefore formulate (2.50) with the projections

$$P_{R^\perp} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \quad \text{and} \quad P_R = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} ,$$

and the data $y = \begin{pmatrix} 1 & 0 \end{pmatrix}^\top$. This leaves us with the solution of the linear system

$$P_{R^\perp} L_w P_{R^\perp}^\top \underbrace{\tilde{v}}_{=P_{R^\perp}\hat{v}} = -P_{R^\perp} L_w P_R^\top y \, ,$$

which for this example reads

$$\begin{pmatrix} 89 & 0 & 0 & -25 \\ 0 & 117 & -36 & 0 \\ 0 & -36 & 121 & -49 \\ -25 & 0 & -49 & 99 \end{pmatrix} \tilde{v} = \begin{pmatrix} 64 \\ 0 \\ 0 \\ 25 \end{pmatrix} .$$

The solution to this linear system is (approximately) $\tilde{v} \approx \begin{pmatrix} 0.8912 & 0.0840 & 0.2732 & 0.6128 \end{pmatrix}^\top$, respectively

$$\hat{v} = \begin{pmatrix} 0.8912 & 0.0840 & 0.2732 & 1 & 0.6128 & 0 \end{pmatrix}^\top .$$

We can use this result to classify the remaining nodes by setting all values below 1/2 to zero and above 1/2 to one. In this example, we would (correctly) determine the biological sex of Bryce Dallas Howard as female, of Chad Smith as male, of Conan O' Brian as male, and of Tilda Swinton as female.

2.9.5 From semi-supervised to unsupervised classification

To fully transition from (semi-)supervised to unsupervised machine learning, we want to address the case when r is chosen to be zero. In this case we have just a lot of samples $\{x_i\}_{i=1}^s$ and no labels. Solving (2.50) for $r = 0$, i.e.

$$\hat{v} = \operatorname*{arg\,min}_{v \in [0,1]^s} \left\{ \|M_w v\|^2 \right\} ,$$

is pointless, as it is easy to see that the solution to this problem is simply $\hat{v} = 0$, or any vector $\hat{v}$ with $M_w \hat{v} = 0$ and $\hat{v} \in [0,1]^s$, and therefore not particularly interesting. Things change, however, if we at the same time ensure that the norm of $\hat{v}$ is reasonably large. We can achieve this by solving the optimisation problem

$$\hat{v} = \operatorname*{arg\,min}_{v \in [0,1]^s} \frac{\|M_w v\|}{\|v\|_p} , \tag{2.51}$$

where $\| \cdot \|_p$ denotes the p-norm. This problem is a generalised eigenproblem, i.e. we compute the smallest eigenvector of the graph-Laplacian $L_w = M_w^\top M_w$ subject to the constraint that the eigenvector has to lie in $[0,1]^s$. If M_w has a non-trivial null space, i.e. there exist $v \in \mathbb{R}^s \setminus \{0\}$

with $M_w v = 0$, the smallest eigenvector is simply an element in the null space of M_w with minimal p-norm, which often is not particularly interesting for unsupervised classification either. If we denote such a ground state with v_0, we can formulate the modified eigenproblem

$$v_\lambda = \operatorname*{arg\,min}_{v\in[0,1]^s} \frac{\|M_w(v - v_0)\|}{\|v - v_0\|_p} = \operatorname*{arg\,min}_{v\in[0,1]^s} \frac{\|M_w v\|}{\|v - v_0\|_p} \,,$$

where v_0 is a solution of (2.51). This way we can obtain a non-trivial generalised eigenvector (with eigenvalue $\lambda = \|M_w v_\lambda\|/\|v_\lambda - v_0\|_p$) of the graph-Laplacian L_w. In the following chapter we are going to explore unsupervised machine learning problems more generally.

Example 2.7. If we consider the Euclidean norm ($p = 2$) in (2.51) and forget about the $[0,1]^s$ constraint for a moment, we easily compute

$$\begin{aligned} L_w \hat{v} &= \frac{\|M_w \hat{v}\|^2}{\|\hat{v}\|^2} \hat{v} \\ &= \lambda \hat{v} \,, \end{aligned}$$

for $\lambda := \|M_w \hat{v}\|^2/\|\hat{v}\|^2$. Hence, we deal with a conventional eigenproblem of the graph-Laplacian L_w. If we take the graph Laplacian from Example 2.6, we compute the following (orthonormal) eigenvectors and corresponding eigenvalues:

$$V \approx \begin{pmatrix} 0.4082 & -0.4986 & -0.2759 & 0.7071 & 0.0928 & 0 \\ 0.4082 & 0.4577 & -0.2901 & 0 & -0.1993 & -0.7071 \\ 0.4082 & 0.2416 & 0.4265 & 0 & 0.7701 & 0 \\ 0.4082 & -0.4986 & -0.2759 & -0.7071 & 0.0928 & 0 \\ 0.4082 & -0.1597 & 0.7054 & 0 & -0.557 & 0 \\ 0.4082 & 0.4577 & -0.2901 & 0 & -0.1993 & 0.7071 \end{pmatrix} \quad \text{and} \quad \lambda \approx \begin{pmatrix} 0 \\ 17 \\ 88.93 \\ 153 \\ 175.08 \\ 198 \end{pmatrix} .$$

The first eigenvector, as predicted, is simply a constant vector with eigenvalue zero and not particularly interesting. The second eigenvector, however, is extremely interesting. Let us multiply the vector with -1 and subtract the smallest value (≈ -0.4577) from this product, so that we obtain a vector in $[0,1]^s$ for $s = 6$, namely

$$v \approx \begin{pmatrix} 0.9563 \\ 0 \\ 0.2161 \\ 0.9563 \\ 0.6173 \\ 0 \end{pmatrix} .$$

If we set all values larger than $1/2$ to one and all values smaller than $1/2$ to zero, we obtain $v = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}^\top$, which matches the binary classification result from Example 2.6, but in contrast to Example 2.6 has been obtained in completely unsupervised fashion.

Chapter 3

Unsupervised learning

The last section saw a transitioning from using pairs of input/output samples $\{(x_i, y_i)\}_{i=1}^s$ to having samples of input data $\{x_i\}_{i=1}^s$ only. Learning only from input samples, or training a model based on input samples only, is known as *unsupervised learning.* Unsupervised learning naturally appears in a variety of applications, like clustering of data points or making recommendations to users or customers, where reliable output data is unavailable. In the following, we want to describe several unsupervised learning tasks in greater detail and discuss how to approach these tasks mathematically and computationally.

3.1 Clustering

The term *clustering* describes the process of partitioning a dataset into several subsets, also known as *clusters.* A typical application can be the clustering of academic papers into clusters that represent different research areas or different fields within a research area.

A key challenge in clustering is to infer the correct clusters from the given data, without any additional information in form of labels. In the next section, we want to start with a very basic but hugely popular model that can take care of such a challenge, assuming that we know how many clusters we are looking for.

3.1.1 k-means clustering

The arguably most popular and simple clustering model is *k-means clustering.* Its idea is incredibly simple and yet powerful: we assume that the input data can be partitioned into k clusters, such that every data point only belongs to exactly one cluster. To be more precise, every data point belongs to the particular cluster for which the Euclidean distance of that point to the cluster centre is smaller compared to all other clusters. Mathematically, we can formulate the following optimisation problem that can help to achieve this goal. Suppose we have s data points $\{x_i\}_{i=1}^s$, where every $x_i \in \mathbb{R}^n$ is n-dimensional, and we aim to identify two matrices $z \in \mathbb{R}^{s\times k}$ and $\mu \in \mathbb{R}^{n\times k}$ that minimise the function

$$L(z, \mu) = \sum_{i=1}^{s} \sum_{j=1}^{k} z_{ij} \|x_i - \mu_j\|^2 \, . \tag{3.1}$$

Here $\mu_j \in \mathbb{R}^n$ denotes the j-th column of μ, which is the matrix of *prototype vectors* or *centres* or *centroids* of the cluster. The matrix z contains the cluster assignments, i.e. the information which

Algorithm 4 k-means clustering.

```
Specify: the number of clusters $k$.
Initialise: $\mu^0 \in \mathbb{R}^{n \times k}$
Iterate:
1: for $l = 0, \ldots, N-1$ do
2:     $z_{ij}^{l+1} = \begin{cases} 1 & j = \arg\min_{r \in \{1,\ldots,k\}} \|x_i - \mu_r^l\|^2 \\ 0 & \text{otherwise} \end{cases}$,  for all  $i \in \{1,\ldots,s\}$
3:     $\mu_j^{l+1} = \frac{\sum_{i=1}^s z_{ij}^{l+1} x_i}{\sum_{i=1}^s z_{ij}}$,  for all  $j \in \{1,\ldots,k\}$
4: end for
return $z^N$, $\mu^N$.
```

data point belongs to which cluster. Note that without any additional constraints, the function L in (3.1) is not bounded from below, and attempting to minimise it will result in parameters that send the function value to $-\infty$. Since the squared Euclidean norm is always bounded from below by zero, we have to impose additional constraints on z in order to ensure boundedness from below of L. Ensuring $z_{ij} \in \{0, 1\}$ for all $i \in \{1, \ldots, s\}$ and $j \in \{1, \ldots, k\}$ for example guarantees that L is bounded from below by zero. However, we also want to ensure that every data point is assigned to exactly one class, which is why we also have to enforce $\sum_{j=1}^k z_{ij} = 1$ for all $i \in \{1, \ldots, s\}$. Together with the constraint we obtain the following minimisation problem: find assignments $\hat{z}$ and centroids $\hat{\mu}$ that satisfy

$$(\hat{z}, \hat{\mu}) = \mathop{\arg\min}_{z \in \mathbb{R}^{s \times k}, \mu \in \mathbb{R}^{n \times k}} \left\{ \sum_{i=1}^s \sum_{j=1}^k z_{ij} \|x_i - \mu_j\|^2 \text{ subject to } z_{ij} \in \{0,1\}, \sum_{j=1}^k v_{ij} = 1, \forall i, j \right\}. \tag{3.2}$$

Solving (3.2) is not trivial, as the problem is not convex and there exists no closed-form solution for it. We therefore attempt to solve (3.2) iteratively in an alternating fashion also known as coordinate descent, i.e. via

$$z^{l+1} = \mathop{\arg\min}_{z \in \mathbb{R}^{s \times k}} \left\{ \sum_{i=1}^s \sum_{j=1}^k z_{ij} \|x_i - \mu_j^l\|^2 \quad \text{subject to} \quad z_{ij} \in \{0,1\}, \sum_{j=1}^k z_{ij} = 1, \forall i, j \right\}, \tag{3.3a}$$

$$\mu^{l+1} = \mathop{\arg\min}_{\mu \in \mathbb{R}^{n \times k}} \left\{ \sum_{i=1}^s \sum_{j=1}^k z_{ij}^{l+1} \|x_i - \mu_j\|^2 \right\}, \tag{3.3b}$$

where $l \in \mathbb{N}$ denotes the iteration index and where μ^0 is some suitable initialisation. The nice thing about updating the variables in an alternating fashion is that the individual updates of (3.3) now have closed-form solutions, which read

$$z_{ij}^{l+1} = \begin{cases} 1 & j = \arg\min_{r \in \{1,\ldots,k\}} \|x_i - \mu_r^l\|^2 \\ 0 & \text{otherwise} \end{cases}, \quad \text{for all} \quad i \in \{1, \ldots, s\}, \tag{3.4a}$$

$$\mu_j^{l+1} = \frac{\sum_{i=1}^s z_{ij}^{l+1} x_i}{\sum_{i=1}^s z_{ij}}, \quad \text{for all} \quad j \in \{1, \ldots, k\}. \tag{3.4b}$$

The k-means clustering algorithm is summarised in Algorithm 4. A detailed convergence proof is beyond the scope of this lecture, but we can easily establish a decrease of the objective function with the following lemma.

Lemma 3.1. *Suppose we are given a function $F : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ in two variables, two arguments x^*, y^* in the domain of F, and we compute*

$$\hat{x} = \arg\min_{x \in \mathbb{R}^m} F(x, y^*) \,,$$
$$\hat{y} = \arg\min_{y \in \mathbb{R}^n} F(\hat{x}, y) \,.$$

Then we already know

$$F(\hat{x}, \hat{y}) \leq F(x^*, y^*) \,.$$

Proof. The proof is trivial, since by definition of $\hat{x}$ and $\hat{y}$ we have $F(\hat{x}, y^*) \leq F(x^*, y^*)$ and $F(\hat{x}, \hat{y}) \leq F(\hat{x}, y^*) \leq F(x^*, y^*)$. □

As a consequence of Lemma 3.1, the inequality

$$L(z^{l+1}, \mu^{l+1}) \leq L(z^l, \mu^l)$$

is guaranteed for all iterates (3.4), L as defined in (3.1) and all $l \in \mathbb{N}$.

Despite its simplicity and considerable success (which you will explore in form of coursework), k-means clustering also has some downsides. First of all, in order to use k-means clustering, we have to know the number of clusters k in advance. Secondly, every data sample can only be associated to one cluster, which is not necessarily suitable for some types of application. Thirdly, the clusters are sphere-shaped as a consequence of the Euclidean distance between data points and the cluster centroids, with equal radii for each cluster. For some of these drawbacks there are obvious remedies. Some of them we will discuss in the upcoming section on clustering with Gaußian mixture models.

3.1.2 Gaußian mixture models

A popular alternative to k-means clustering is clustering with *Gaußian mixture models*. Usually spelled Gaussian mixture models in English literature, I take the liberty to use the original German spelling Gauß with the letter ß, the so-called sharp s.

In Gaußian mixture models, the idea for each data sample x_i is to maximise the likelihood of a probability density function that is the some of Gaußians, i.e.

$$p(x_i | \mu_1, \ldots, \mu_k, \Sigma_1, \ldots, \Sigma_k, \rho_1, \ldots, \rho_k) = \sum_{j=1}^{k} \rho_j \, \mathcal{N}(x_i | \mu_j, \Sigma_j) \,,$$

where $\mathcal{N}(x_i | \mu_j, \Sigma_j)$ denotes the multivariate normal distribution density function, i.e.

$$\mathcal{N}(x_i | \mu_j, \Sigma_j) := \frac{1}{\sqrt{(2\pi)^n \det(\Sigma_j)}} \exp\left(-\frac{1}{2} \langle \Sigma_j^{-1}(x_i - \mu_j), x_i - \mu_j \rangle \right) \,.$$

Assuming that all samples are independent and identically distributed (iid), and using the short-hand notations

$$X := \begin{pmatrix} | & | & & | \\ x_1 & x_2 & \ldots & x_s \\ | & | & & | \end{pmatrix} , \; M := \begin{pmatrix} | & | & & | \\ \mu_1 & \mu_2 & \ldots & \mu_k \\ | & | & & | \end{pmatrix} , \; \Sigma := \begin{pmatrix} \Sigma_1 & \Sigma_2 & \ldots & \Sigma_k \end{pmatrix} \,,$$

and $\rho := \big(\rho_1, \ldots, \rho_k\big)^\top$, we can write the density function for all samples as

$$p(X|M, \Sigma, \rho) = \prod_{i=1}^{s} \sum_{j=1}^{k} \rho_j \mathcal{N}(x_i|\mu_j, \Sigma_j)\,. \tag{3.5}$$

The vector ρ is considered to be a discrete probability mass vector, i.e. it satisfies $\rho_j \geq 0$ and $\sum_{j=1}^{k} \rho_j = 1$. Given (3.5), the goal is to find parameters M, Σ and ρ that maximise (3.5). As we know from earlier sections as well as the previous semester, instead of maximising (3.5) we can also minimise the negative log-likelihood, i.e.

$$\begin{aligned}\mathrm{NNL}(M, \Sigma, \rho) &:= -\log\left(\prod_{i=1}^{s} \sum_{j=1}^{k} \rho_j \mathcal{N}(x_i|\mu_j, \Sigma_j)\right)\\ &= -\sum_{i=1}^{s} \log\left(\sum_{j=1}^{k} \rho_j \mathcal{N}(x_i|\mu_j, \Sigma_j)\right)\,.\end{aligned}$$

We now pursue the same strategy that we always pursue when it comes to minimising a negative log-likelihood: we compute the derivatives of NNL with respect to the individual parameters and set them to zero. Starting with the prototype vectors μ_l, we observe

$$\frac{\partial \mathrm{NNL}}{\partial \mu_l} = \sum_{i=1}^{s} \frac{\rho_j\, \mathcal{N}(x_i|\mu_l, \Sigma_l)}{\sum_{j=1}^{k} \rho_j\, \mathcal{N}(x_i|\mu_j, \Sigma_j)} \Sigma_l^{-1}(x_i - \mu_l) = 0\,,$$

which can be concluded with the identity $\frac{\partial}{\partial x_i} \log\left(f(x_1, \ldots, x_n)\right) = \left(\frac{\partial}{\partial x_i} f(x_1, \ldots, x_n)\right) / f(x_1, \ldots, x_n)$. Hence, we can re-write $\partial\mathrm{NNL}/\partial\mu_l = 0$ to

$$\mu_l = \frac{1}{s_l} \sum_{i=1}^{s} \gamma(x_i, M, \Sigma)_l\, x_i\,, \tag{3.6}$$

where we have defined

$$\gamma(x_i, M, \Sigma)_l := \frac{\rho_j\, \mathcal{N}(x_i|\mu_l, \Sigma_l)}{\sum_{j=1}^{k} \rho_j\, \mathcal{N}(x_i|\mu_j, \Sigma_j)}\,,$$

and

$$s_l := \sum_{i=1}^{s} \gamma(x_i, M, \Sigma)_l\,,$$

and made the assumption that Σ_l is invertible. Setting the partial derivative with respect to a covariance matrix Σ_l to zero and bringing Σ_l onto one side of the equation then yields

$$\Sigma_l = \frac{1}{s_l} \sum_{i=1}^{s} \gamma(x_i, M, \Sigma)_l\, (x_i - \mu_l)(x_i - \mu_l)^\top\,. \tag{3.7}$$

Note that we require the identity $\frac{\partial}{\partial \Sigma_j} \det(\Sigma_j)^{-\frac{1}{2}} = -\frac{1}{2} \det(\Sigma_j)^{-\frac{1}{2}} \Sigma_j^{-1}$ to derive (3.7). The optimality condition for the variable ρ is a bit trickier, as we have to also ensure the constraints

Algorithm 5 Gaußian-mixture model clustering.

```
Specify: the number of clusters $k$.
Initialise: $M^0 \in \mathbb{R}^{n \times k}, \Sigma^0 \in \mathbb{R}^{n \times kn}, \rho^0 \in \mathbb{R}^k$
Iterate:
1: for $p = 0, \ldots, N-1$ do
2:     for $l = 1, \ldots, k$ do
3:         $s_l^{p+1} = \sum_{i=1}^s \gamma(x_i, M^p, \Sigma^p)_l$ ,
4:         $\mu_l^{p+1} = \frac{1}{s_l^{p+1}} \sum_{i=1}^s \gamma(x_i, M^p, \Sigma^p)_l \, x_i$ ,
5:         $\Sigma_l^{p+1} = \frac{1}{s_l^{p+1}} \sum_{i=1}^s \gamma(x_i, M^p, \Sigma^p)_l \left(x_i - \mu_l^{p+1}\right)\left(x_i - \mu_l^{p+1}\right)^\top$ ,
6:         $\rho_l^{p+1} = \frac{s_l^{p+1}}{s}$
7:     end for
8: end for
return $M^N$, $\Sigma^N$, $\rho^N$.
```

$\rho_l \geq 0$ and $\sum_{j=1}^k \rho_k = 1$. One way to do so is by introducing a so-called *Lagrange multiplier* λ, and formulate the Lagrange function

$$-\log(p(X|M, \Sigma, \rho)) + \lambda \left(1 - \sum_{j=1}^k \rho_j\right) .$$

Computing the partial derivative of this function with respect to ρ_l and setting it to zero then yields

$$0 = \sum_{i=1}^s \gamma(x_i, M, \Sigma)_l + \lambda \, .$$

Multiplying both sides by ρ and summing from one to k then leads to the relation $\lambda = -s$, so that we compute

$$\rho_l = \frac{s_l}{s} \, . \tag{3.8}$$

Equations (3.6) to (3.8) from an optimality system that characterises the first-order optimality condition of minimising NLL with respect to M, Σ and ρ. The question that remains is: how do we utilise this optimality system in order to obtain an algorithm to compute the parameters numerically? Many options are available, but the most common one in the literature is the *Expectation Maximisation (EM)* algorithm. The idea is to solve the optimality system (3.6) to (3.8) via the fixed-point iteration in Algorithm 5.

3.1.3 Spectral clustering

When we transitioned from semi-supervised to unsupervised learning, we have discovered a first example of *spectral clustering* in form of Problem (2.51). The eigenvalues of a matrix are often referred to as *spectrum*, hence the name spectral clustering. In this section we want to focus on more generic eigenvalue problems of the form

$$\min_{S(u)=0} \frac{F(u)}{G(u)} \, . \tag{3.9}$$

Algorithm 6 Nonlinear inverse power iteration.

```
Specify: parameter $\tau > 0$, index $K$
Initialise: $u^0$
Iterate:
1: for $k = 0, \ldots, K-1$ do
2:     $\lambda^k = F(u^k)/G(u^k)$
3:     $u^{k+1} = (I + \tau\,\partial F)^{-1}(u^k + \tau\lambda^k \nabla G(u^k))$
4: end for
return $u^K, \lambda^k$.
```

Note that (2.51) is a special case of (3.9) for the choices $F(u) = \|M_w u\|$, $G(u) = \|u\|_p$ and $S(u) = \chi_{[0,1]^s}(u)$. But other, more interesting eigenvalue problems can be formulated as well; the following problem is particularly interesting for graph-based spectral clustering:

$$\min_{\text{median}(u)=0} \frac{\|M_w u\|_1}{\|u\|_1} = \min_u \frac{\|M_w u\|_1}{\|u - \text{median}(u)\|_1} = \min_u \frac{\sum_{i=1}^{\#\text{ edges}} \left| \sum_{j=1}^{\#\text{ vertices}} w_{ij} u_j \right|}{\|u - \text{median}(u)\|_1} ; \tag{3.10}$$

Here, median refers to the *median* of a vector, which is defined as

$$\text{median}(u) := \arg\min_{s\in\mathbb{R}} \sum_{j=1}^{\#vertices} |s - u_j| \, .$$

Problem (3.10) is also known as the *ratio cut* problem, which is equivalent to the Cheeger cut problem. The motivation behind (3.10) is to create labels u that are piecewise constant. We will discover in the computational coursework that this leads to superior classification results compared to model (2.51). A key question that remains is the following: how do we computationally solve (3.10), or more generally (3.9)? We start with Problem (3.9) and the assumption that both F and G are continuously differentiable. We can then compute the gradient of $F(u)/G(u)$ and set it to zero, which reads

$$0 = \nabla F(\hat{u}) - \frac{F(\hat{u})}{G(\hat{u})} \nabla G(\hat{u}) \, . \tag{3.11}$$

We aim to approximate a solution of (3.11) via a generalised inverse power iteration of the form

$$\begin{aligned} u^{k+1} &= \arg\min_u \left\{ F(u) - \lambda^k \langle \nabla G(u^k), u - u^k \rangle + \frac{1}{2\tau} \|u - u^k\|^2 \right\} , \\ &= \arg\min_u \left\{ \frac{1}{2} \left\| u - \left(u^k + \tau\lambda^k \nabla G(u^k) \right) \right\|^2 + \tau F(u) \right\} , \\ &= (I + \tau\partial F)^{-1} \left(u^k + \tau\lambda^k \nabla G(u^k) \right) , \end{aligned}$$

for $\lambda^k := F(u^k)/G(u^k)$ and a positive step-size parameter τ. What is nice about this generalised inverse power iteration is that it is simple (assuming that computing the proximal map is simple) and that we easily observe a decrease of the objective $F(u)/G(u)$.

Lemma 3.2. *Suppose the functions F and G are continuous and convex, and G is also differentiable. Then the iterates of Algorithm satisfy*

$$\lambda^{k+1} \leq \lambda^k \, .$$

Here, $\lambda^k = F(u^k)/G(u^k)$ denotes the Rayleigh quotient of the current iterate, for all $k \in \mathbb{N}$.

Proof. We observe

$$\begin{aligned}
&F(u^{k+1}) - \lambda^k G(u^{k+1})\\
&\leq F(u^{k+1}) - \lambda^k G(u^{k+1}) + \frac{1}{2\tau}\|u^{k+1} - u^k\|^2\\
&\leq F(u^{k+1}) - \lambda^k \left(G(u^{k+1}) - D_G(u^{k+1}, u^k)\right) + \frac{1}{2\tau}\|u^{k+1} - u^k\|^2\\
&= F(u^{k+1}) - \lambda^k \left(G(u^k) + \langle \nabla G(u^k), u^{k+1} - u^k\rangle\right) + \frac{1}{2\tau}\|u^{k+1} - u^k\|^2\\
&\leq F(u^k) - \lambda^k \left(G(u^k) + \langle \nabla G(u^k), u^k - u^k\rangle\right) + \frac{1}{2\tau}\|u^k - u^k\|^2\\
&= F(u^k) - \lambda^k G(u^k)\,.
\end{aligned}$$

Here, the second inequality follows from the non-negativity of the squared Euclidean norm and the parameter τ. The third inequality follows from the definition of the Bregman distance and the convexity of G. The last inequality follows from the fact that u^{k+1} minimises $E^k(u) := F(u) - \lambda^k\langle \nabla G(u^k), u - u^k\rangle + \frac{1}{2\tau}\|u - u^k\|^2$, which means that the objective value has to be smaller for u^{k+1} than for u^k. Based on the assumption that $G(u^{k+1}) \neq 0$ and the definition of the Rayleigh quotient $\lambda^k = F(u^k)/G(u^k)$ for all $k = 0, 1, \ldots$, we conclude the proof by verifying

$$\begin{aligned}
F(u^{k+1}) - \lambda^k G(u^{k+1}) &\leq F(u^k) - \lambda^k G(u^k)\\
&= F(u^k) - F(u^k) = 0\\
\implies \qquad \frac{F(u^{k+1}) - \lambda^k G(u^{k+1})}{G(u^{k+1})} &\leq 0\\
\implies \qquad \frac{F(u^{k+1})}{G(u^{k+1})} - \lambda^k &\leq 0\\
\implies \qquad \lambda^{k+1} &\leq \lambda^k\,.
\end{aligned}$$

□

The beauty of Lemma 3.2 is that a monotonic decrease of the generalised Rayleigh quotient $F(u)/G(u)$ over the course of the generalised inverse power iteration is guaranteed. We do, however, also have enforce normalisation constraints, as the the algorithm could otherwise make u^k smaller and smaller at every iteration. Another more important issue, however, is that in order to realise Problem (3.10), the assumption that G is differentiable is too restrictive. On top of that, solving an optimisation problem of the form

$$u^{k+1} = \arg\min_u \left\{ \frac{1}{2}\left\| u - \left(u^k + \tau\lambda^k \nabla G(u^k)\right)\right\|^2 + \tau\|M_w u\|_1 \right\}$$

has no closed-form solution, which is why we have to approximate this optimisation problem with another iterative method. We want to address these two issues in the following two sections.

Subdifferential calculus

Note that Corollary 2.1 gives us an alternative characterisation of differentiable, convex functions, via the inequality

$$E(u) - E(v) - \langle \nabla E(v), u - v\rangle \geq 0\,,$$

for all $u, v \in \mathbb{R}^n$. We can use this inequality to characterise so-called *subgradients* for functions that are convex but not necessarily differentiable.

Definition 3.1 (Subdifferential)**.** *Let $E : \mathcal{C} \subset \mathbb{R}^n \to \mathbb{R}$ be a convex and continuous function. Then it's* subdifferential *∂E is defined as the set*

$$\partial E(v) := \{g \in \mathbb{R}^n \,|\, E(w) - E(v) \geq \langle g, w - v \rangle \,, \forall w \in \mathbb{R}^n\} \,.$$

The elements $g \in \partial E(v)$ are called subgradients*.*

Example 3.1 (Subdifferential of absolute value function)**.** Let $E : \mathbb{R} \to \mathbb{R}_{\geq 0}$ be the absolute value function $E(w) := |w|$. Then the subdifferential ∂E is

$$\partial E(w) = \text{sign}(w) := \begin{cases} \{1\} & w > 0 \\ [-1, 1] & w = 0 \\ \{-1\} & w < 0 \end{cases} .$$

The mathematical verification of this subdifferential identity is left as an exercise.

Example 3.2 (Subdifferential of the median function)**.** Suppose E is the function $E(s) = \sum_{j=1}^{\#\text{vertices}} |s - u_j|$, such that $\text{median}(u) = \arg\min_{s \in \mathbb{R}} E(s)$. Since E is convex, an equivalent characterisation is $0 \in \partial E(s)$. The subdifferential ∂E of E can be characterised as

$$0 \in \sum_{j=1}^{\#\text{vertices}} \text{sign}(\hat{s} - u_j) \,.$$

The right-hand-side can only equal zero if the number of positive entries and the number of negative entries of $\hat{s} - u$ is identical.

Based on the subdifferential notion we can characterise minimisers of convex functions as follows.

Lemma 3.3. *Suppose $E : \mathcal{C} \to \mathbb{R}$ is a continuous and convex function that is bounded from below. Then an argument $\hat{w} \in \mathcal{C}$ satisfies $0 \in \partial E(\hat{w})$ if and only if $\hat{w}$ is a global minimiser.*

Proof. The condition $0 \in \partial E(\hat{w})$ is equivalent to

$$E(w) - E(\hat{w}) \geq \langle 0, w - \hat{w} \rangle = 0$$

for all $w \in \mathcal{C}$, which is equivalent to $\hat{w}$ being a global minimiser. □

Hence, we have a notion of optimality for convex, subdifferentiable functions. The question that remains is: how does this help us with applying the inverse power iteration to functions G that are not differentiable? The simple idea is that we modify the inverse power iteration as follows:

$$u^{k+1} = (I + \tau \partial F)^{-1} \left(u^k + \tau \lambda^k g^k \right) \,, \tag{3.12a}$$

$$g^{k+1} \in \partial G(u^{k+1}) \,, \tag{3.12b}$$

for $k \in \mathbb{N}$. Putting the difficulty of choosing an appropriate subgradient aside for a moment, a remarkable result is that we can transfer the result of Lemma 3.2 directly to Algorithm (3.12).

Lemma 3.4. *Suppose the functions F and G are continuous and convex, and G has a non-empty subdifferential ∂G. Then the iterates of Algorithm* (3.12) *satisfy*

$$\lambda^{k+1} \leq \lambda^k \, .$$

Here, $\lambda^k = F(u^k)/G(u^k)$ denotes the Rayleigh quotient of the current iterate u^k, for all $k \in \mathbb{N}$.

Proof. The proof is identical to the one of Lemma 3.2, since all inequalities still hold if the gradient of G is replaced with any suitable subgradient of G. □

Coming back to Problem (3.10), we can solve this problem via (3.12), which then reads

$$\begin{aligned}
\tilde{u}^{k+1} &= \arg\min_u \left\{ \frac{1}{2} \left\| u - \left(v^k + \tau \lambda^k g^k \right) \right\|^2 + \tau \|M_w u\|_1 \right\} \, , \\
u^{k+1} &= \frac{\tilde{u}^{k+1}}{\|\tilde{u}^{k+1}\|_1} \, , \\
v^{k+1} &= u^{k+1} - \operatorname{median}(u^{k+1}) \, , \\
g_j^{k+1} &= \begin{cases} \operatorname{sign}(v_j^{k+1}) & v_j^{k+1} \neq 0 \\ -\frac{|v_+^{k+1}| - |v_-^{k+1}|}{|v_0^{k+1}|} & v_j^{k+1} = 0 \end{cases} \, , \qquad \forall j \in \{1, \ldots, \#\text{vertices}\} \, , \\
\lambda^{k+1} &= \frac{\|M_w u^{k+1}\|_1}{\|v^{k+1}\|_1} \, .
\end{aligned}$$

Note that we added a normalisation step that doesn't affect the convergence properties of the algorithm but that ensures that the generalised eigenvector doesn't become arbitrarily small. We further choose each subgradient g^{k+1} so that they satisfy $\sum_{i=1}^{\#\text{vertices}} g_j^{k+1} = 1$ for all k, in order to guarantee $\lambda^k \langle u^k, g^k \rangle = \lambda^k \langle v^k, g^k \rangle$ for all k. What remains to be shown is how we can solve the subproblems

$$\tilde{u}^{k+1} = \arg\min_u \left\{ \frac{1}{2} \left\| u - \left(v^k + \tau \lambda^k g^k \right) \right\|^2 + \tau \|M_w u\|_1 \right\} \tag{3.13}$$

computationally.

Solving the inner problem

We conclude by discussing the solution of the subproblems (3.13). In order to do so, we use the same duality trick that we employed in the support vector machine section. Note that we can reformulate the underlying minimisation problem of (3.13) as the min-max problem

$$\min_u \max_p \frac{1}{2} \left\| u - \left(v^k + \tau \lambda^k g^k \right) \right\|^2 + \tau \langle M_w u, p \rangle - \chi_{\|\cdot\|_\infty \leq 1}(p) \, .$$

Since the objective is convex in u and concave in p, we can interchange min and max. Minimising with respect to u first, i.e.

$$\min_u \frac{1}{2} \left\| u - \left(v^k + \tau \lambda^k g^k \right) \right\|^2 + \tau \langle M_w u, p \rangle \, ,$$

leads to a differentiable problem with solution

$$u = v^k + \tau \lambda^k g^k - \tau M_w^\top p \, . \tag{3.14}$$

Inserting this solution into the original problem leaves us with the dual problem

$$\begin{aligned}
&\max_p \frac{\tau^2}{2}\left\|M_w^\top p\right\|^2 + \tau\left\langle M_w\left(v^k + \tau\lambda^k g^k - \tau M_w^\top p\right), p\right\rangle - \chi_{\|\cdot\|_\infty\leq 1}(p)\,,\\
=\ &\max_p -\frac{\tau^2}{2}\left\|M_w^\top p\right\|^2 + \tau\left\langle M_w\left(v^k + \tau\lambda^k g^k\right), p\right\rangle - \chi_{\|\cdot\|_\infty\leq 1}(p)\,,\\
=\ &\min_p \frac{\tau^2}{2}\left\|M_w^\top p\right\|^2 - \tau\left\langle M_w\left(v^k + \tau\lambda^k g^k\right), p\right\rangle + \chi_{\|\cdot\|_\infty\leq 1}(p)\,,\\
=\ &\min_{\|p\|_\infty\leq 1} \frac{1}{2}\left\|\tau M_w^\top p - \left(v^k + \tau\lambda^k g^k\right)\right\|^2\,.
\end{aligned}$$

This dual problem we can for instance solve via proximal gradient descent as described in Section 2.7.3 or coordinate descent as described in Section 2.9.3. We then recover the solution of the primal problem via (3.14).

3.2 Matrix factorisation

Unsupervised machine learning strategies often rely on data dimensionality reduction techniques in order to reveal information hidden in the data. A very basic class of data dimensionality reduction techniques is known as *matrix factorisation*. The idea behind matrix factorisation is that data is given in form of a matrix X, and we assume that the true dimensionality of the matrix (for example in form of the rank of the matrix) is much lower than the actual dimension of the matrix X. This assumption can be formulated as

$$X = WZ^\top\,, \tag{3.15}$$

for matrices $X \in \mathbb{R}^{s\times n}$, $W \in \mathbb{R}^{s\times k}$ and $Z \in \mathbb{R}^{n\times k}$. If k is smaller than s and n, the rank of X is k instead of s or n. In practice, this means that instead of storing sn values of X, we only need to store $k(n+s)$ values, which often is much smaller if k is chosen to be small. Imagine for example that we have a data matrix $X \in \mathbb{R}^{60000\times 784}$, where every row is the vector representation of one image of the MNIST training dataset, and assume that all images of hand-written digits can (approximately) be seen as linear combinations of only 100 different images, i.e. $k = 100$. Then one could store all images with only $100 \times (784 + 60000) = 6.078.400$ entries, as opposed to the 47.040.000 entries of the original dataset. These are around 13 % of the original data entries.

Matrix factorisation has uses in many unsupervised machine learning applications, such as data compression and recommender systems. There are many different ways of factorising matrices and many different matrix factorisation techniques in order to compute specific factorisations. In the following section, we want to discuss two extremely popular and widely used approaches known as singular value decomposition and principal component analysis.

3.2.1 Singular value decomposition (SVD) and principal component analysis (PCA)

Singular value decomposition, or *SVD* in short, is a factorisation that generalises the concept of eigendecompositions of square matrices. It can be shown that every real matrix $X \in \mathbb{R}^{s\times n}$ can be factorised into three matrices $U \in \mathbb{R}^{s\times s}$, $\Sigma \in \mathbb{R}^{s\times n}$ and $V \in \mathbb{R}^{n\times n}$ via

$$X = U\Sigma V^\top\,. \tag{3.16}$$

Here both U and V are orthogonal matrices, i.e. $U^\top U = I_{s\times s}$, $UU^\top = I_{s\times s}$, $V^\top V = I_{n\times n}$ and $VV^\top = I_{n\times n}$, whose columns are called (left- and right-) *singular vectors* of X. The matrix Σ is a diagonal matrix of the form

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \dots & \sigma_n \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} \qquad \text{or} \qquad \Sigma = \begin{pmatrix} \sigma_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & \ddots & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & \sigma_s & 0 & \dots & 0 \end{pmatrix} ,$$

depending on whether $s > n$ (first case) or $n > s$ (second case). If $n = s$, the matrix Σ is the square diagonal matrix

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \dots & \sigma_n \end{pmatrix} .$$

The entries $\{\sigma_j\}_{j=1}^{\min(s,n)}$ are called the *singular values* of X, and are all non-negative, i.e. $\sigma_j \geq 0$ for all $j \in \{1, \dots, \min(s,n)\}$. By convention, the singular values are ordered in descending order, i.e. $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{\min(s,n)} \geq 0$.

It is easy to see that (3.16) is a special case of (3.15), for example via $W = U\Sigma$ and $Z = V$. For now, we either have $k = n$ or $k = s$, but we will soon address the choice of $k < \min(s,n)$. Note that the SVD of a matrix X allows us to easily compute the Frobenius norm of that matrix X, as the Frobenius norm is equivalent to the Euclidean norm of the vector of singular values, which we see from

$$\|X\|_{\text{Fro}}^2 = \langle X, X\rangle = \langle U\Sigma V^\top, U\Sigma V^\top\rangle = \langle \Sigma V^\top, \underbrace{U^\top U}_{=I_{s\times s}} \Sigma V^\top\rangle = \langle \Sigma \underbrace{V^\top V}_{=I_{n\times n}}, \Sigma\rangle = \|\Sigma\|_{\text{Fro}}^2 = \sum_{j=1}^{\min(s,n)} \sigma_j^2 .$$

Here $\langle X, Y\rangle = \sum_{i=1}^s \sum_{j=1}^n x_{ij} y_{ij}$ denotes the matrix inner product for two matrices $X, Y \in \mathbb{R}^{s\times n}$, and the Frobenius norm of a matrix is defined as $\|X\|_{\text{Fro}} = \sqrt{\sum_{i=1}^s \sum_{j=1}^n |x_{ij}|^2}$.

Getting back to the idea of lower-dimensional approximations of X, we can easily define such an approximation with the help of the SVD of X. Suppose we define a new matrix $U_k \in \mathbb{R}^{s\times k}$ as the first k columns of U. Then we observe

$$U_k U_k^\top X = U_k U_k^\top U\Sigma V^\top = U_k \begin{pmatrix} I_{k\times k} & 0_{k\times(s-k)} \end{pmatrix} \Sigma V^\top = U\Sigma_k V^\top ,$$

where $\Sigma_k \in \mathbb{R}^{s\times n}$ is defined as

$$\Sigma_k = \begin{pmatrix} \sigma_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & \ddots & 0 & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_k & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{pmatrix} ,$$

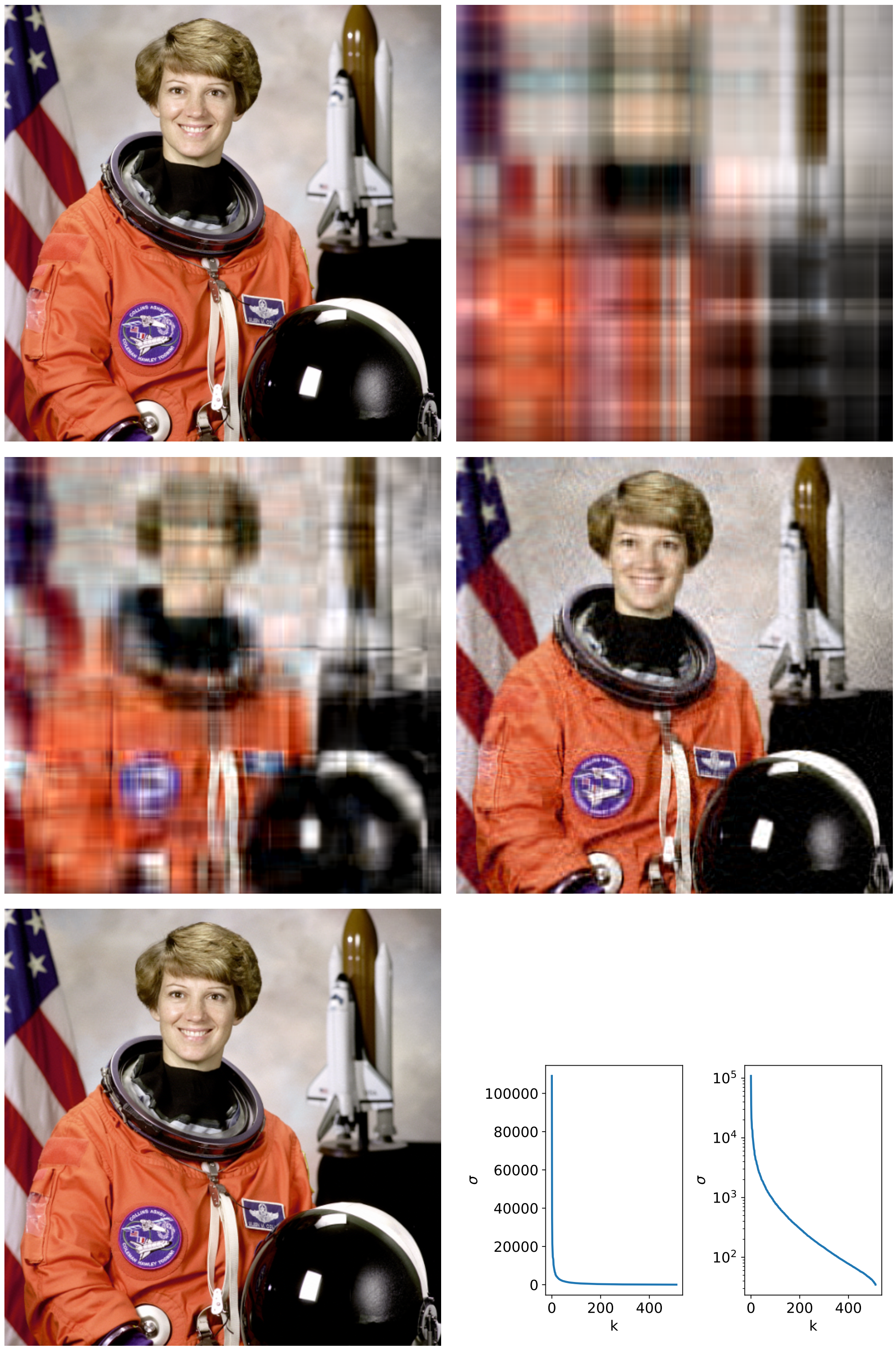


Figure 3.1: An image of astronaut Eileen Collins from the NASA Great Images database and several low-rank approximations for $k = 3$, $k = 10$, $k = 50$ and $k = 256$. In the bottom right corner we also see two plots of the singular values (one with normal and one withh logarithmic scaling).

where we have only visualised Σ_k when Σ is square for the sake of notational simplicity. Consequently, the matrix $X_k := U\Sigma_k V^\top = U_k U_k^\top X$ is a rank-k-approximation of X, and a special case of (3.15) with $W = U_k$ and $Z = X^\top U_k$. In fact, it is not only a rank-k-approximation but the best rank-k-approximation in the sense of the Frobenius norm.

Theorem 3.1 (Best rank-k-approximation / Eckart–Young–Mirsky theorem)**.** *For any matrix $X \in \mathbb{R}^{s\times n}$ and any matrix $\hat{X} \in \mathbb{R}^{s\times n}$ with rank$(\hat{X}) = k$, we have*

$$\|X - \hat{X}\|_{Fro}^2 \geq \|X - X_k\|_{Fro}^2 = \|X - U_k U_k^\top X\|_{Fro}^2 = \sum_{j\geq k+1}^{\min(s,n)} \sigma_j^2 \, .$$

Hence, X_k is the best rank-k-approximation in the sense of the Frobenius norm.

Proof. For a sketch of a proof you can for example follow the explanations at this link here, or look at this original publication by Eckart and Young. □

Please find a visual SVD example in Figure 3.1. Here we have taken an image of astronaut Eileen Collins (from the data module of the Python library scikit-image) and converted it into a matrix $X \in \mathbb{R}^{512\times 1536}$. We have computed several lower rank-approximations of this matrix and visualised them in Figure 3.1, together with a plot of the singular values.

For the remainder of this section we want to embrace a probabilistic view-point and assume that we have s samples $x_j \in \mathbb{R}^n$, $j \in \{1, \ldots, s\}$, that are all drawn i.i.d. from some distribution, and store them as columns in a matrix $X \in \mathbb{R}^{n\times s}$ (note that s and n are interchanged in comparison to our usual convention). We then compute the empirical mean $\overline{x}$ and the covariance matrix K for those samples as

$$\overline{x} := \frac{1}{s}\sum_{j=1}^{s} x_j \qquad \text{and} \qquad K := \frac{1}{s}\sum_{j=1}^{s} (x_j - \overline{x})\,(x_j - \overline{x})^\top \, ,$$

Now assume that we have already subtracted the empirical mean from the data, i.e. $\overline{x} = \underbrace{(0 \quad \ldots \quad 0)}_{n\,\text{entries}}{}^\top$, and a multiple of s and the covariance matrix K then reads

$$sK = \sum_{j=1}^{s} x_j x_j^\top = XX^\top = U\Sigma V^\top V \Sigma^\top U^\top = U\Sigma^2 U^\top \, ,$$

where Σ^2 denotes the matrix with squared singular vectors on the diagonal and zeros elsewhere. If we multiply sK with $U^\top$ from the left and with U from the right (or replace X in the above formula with $U^\top X$), we obtain

$$s\,U^\top K U = (U^\top X)(U^\top X)^\top = \Sigma^2 \, ,$$

which implies that applying the rotation $U^\top X$ helps to uncorrelate the different components of X. In addition, the singular vector u_1 that corresponds to the largest singular value σ_1 is also the vector amongst all singular vectors that has the largest variance. In this context, the singular vectors are referred to as *principal components*, and the decomposition is known as *principal component analysis*. Note that the only key difference compared to the SVD is the centering of the data, i.e. we subtract the mean of the columns of the data matrix.

3.2.2 Sparse principal component analysis

A potential disadvantage of principal component analysis is that the principal components are usually linear combinations of all input variables. This might make sense for some applications, but for some other applications it might make more sense if a principal component is a linear combination of only a few input variables. *Sparse principal component analysis* represents an alternative to classical principal component analysis that allows the recovery of principal components that are only a combination of few input variables. The classical way of formulating the sparse principal component problem is the optimisation problem

$$\max_{\|u\|_p=1} \|u^\top X\| \qquad \text{subject to} \qquad \|u\|_0 \leq k\,, \tag{3.17}$$

where $\|u\|_0$ denotes the number of non-zero entries of the principal component $u \in \mathbb{R}^n$, $\|u\|_p$ denotes the p-vector-norm and $k \in \mathbb{N}$ is the (maximum) number of non-zero entries. Because of the cardinality constraint, Problem (3.17) is NP-hard and difficult to solve. A remedy is to replace the cardinality constrain with a one-norm constraint. There are many ways to formulate such relaxed formulations, but we opt for a nonlinear eigenvalue formulation of the form

$$\min_{\|u\|_p=1} \frac{\alpha\|u\|_1}{\|X^\top u\|} = \min_{\|u\|_p=1} \frac{\alpha\|u\|_1}{\sqrt{u^\top K u}}\,, \tag{3.18}$$

where $K = XX^\top$ denotes the covariance matrix of X and α is a positive scalar. Similar to the ratio cut problem in Section 3.1.3, we can iteratively approximate solutions of (3.18) via the following instance of the nonlinear inverse power iteration:

$$w^{k+1} = \arg\min_{w\in\mathbb{R}^n} \left\{ \alpha\|w\|_1 - \lambda^k \frac{\langle Ku^k, w\rangle}{\|X^\top w^k\|} + \frac{1}{2\tau}\|w - u^k\|^2 \right\}, \tag{3.19a}$$

$$u^{k+1} = \frac{w^{k+1}}{\|w^{k+1}\|_p}\,, \tag{3.19b}$$

$$\lambda^{k+1} = \frac{\alpha\|u^{k+1}\|_1}{\|X^\top u^{k+1}\|}\,. \tag{3.19c}$$

Note that (3.19a) is simply a proximal mapping with respect to the one-norm as defined in Section 2.7.3, i.e.

$$w^{k+1} = (I + \tau\alpha\|\cdot\|_1)^{-1}\left(u^k + \tau\lambda^k \frac{Ku^k}{\|X^\top u^k\|}\right).$$

In the next section we explore another alternative to principal component analysis that is robust towards isolated, sparse outliers in the data.

3.2.3 Robust principal component analysis

Singular value decomposition and principal component analysis are powerful tools for finding low-rank representations of data matrices. However, many data matrices are only approximately low-rank, for example as a result of measurement errors, artefacts or other nonlinear influences on the data acquisition process. In this section we therefore want to discuss a PCA-variant that is robust towards outliers in the data. Before we introduce the method, we want to reformulate the problem of finding a low-rank approximation first.

The problem of finding a low-rank matrix approximation $\hat{L} \in \mathbb{R}^{n\times s}$ to a (centred) matrix $X \in \mathbb{R}^{n\times s}$ (of potentially full rank) can be formulated as the optimisation problem

$$\hat{L} = \operatorname*{arg\,min}_{L\in\mathbb{R}^{n\times s}} \left\{ \frac{1}{2}\|L - X\|_{\text{Fro}}^2 + \alpha\|L\|_* \right\} . \tag{3.20}$$

Here, $\|\cdot\|_*$ denotes the so-called *nuclear-norm*, which is the one-norm of the vector of singular values of X, which is simply the sum of the singular values, i.e.

$$\|L\|_* = \sum_{j=1}^{\min(n,s)} \sigma_j ,$$

where $\{\sigma_j\}_{j=1}^{\min(n,s)}$ denote the singular values of L, and $\alpha > 0$ is a regularisation parameter. A tiny exercise for the curious reader: why is the one norm in this case just the sum? What is the difference of the nuclear norm compared to the Frobenius norm?

Problem (3.20) will find a matrix $\hat{L}$ close to X, with lower rank, depending on the choice of α. If X, however, is only low-rank up to some sparsely distributed but significant outliers, model (3.20) is of no good use for the approximation of this low-rank matrix. We therefore replace the squared Frobenius-norm in (3.20) with the matrix one-norm, i.e. $\|\cdot\|_1 := \sum_{j=1}^n \sum_{i=1}^s |\cdot_{ij}|$, and obtain

$$\hat{L} = \operatorname*{arg\,min}_{L\in\mathbb{R}^{n\times s}} \left\{ \|L - X\|_1 + \alpha\|L\|_* \right\} . \tag{3.21}$$

If we substitute $S = X - L$ in (3.21), we have

$$(\hat{L}, \hat{S}) = \operatorname*{arg\,min}_{L\in\mathbb{R}^{n\times s}, S\in\mathbb{R}^{n\times s}} \left\{ \|S\|_1 + \alpha\|L\|_* \quad \text{subject to} \quad X = L + S \right\} . \tag{3.22}$$

Problem (3.22) is known as *robust principal component analysis* and has first been proposed in Emmanuel J. Candès, Xiaodong Li, Yi Ma, and John Wright. "Robust principal component analysis?." Journal of the ACM (JACM) 58, no. 3 (2011): 1-37. In the following we want to address how to solve problem (3.22) computationally (and efficiently).

Before we dive into the derivation of an algorithm for the numerical solution of (3.22), we want to emphasise that (3.20) is a proximal mapping. In the following we show that this proximal map has a simple closed form solution. In order to do so, note that for $R(L) = \|L\|_*$ we can guarantee $R(Q_1\, L\, Q_2) = R(L)$ for two orthogonal matrices $Q_1 \in \mathbb{R}^{n\times n}$ and $Q_2 \in \mathbb{R}^{s\times s}$, which we see immediately from the fact that the singular values of $Q_1\, L\, Q_2$ is the same as the singular values of L (exercise). If $X = U\Sigma V^\top$ denotes the SVD of X, we can substitute $\hat{L}$ for $\tilde{L} = U^\top \hat{L} V$ with

$$\begin{aligned}\tilde{L} &= \operatorname*{arg\,min}_{L\in\mathbb{R}^{n\times s}} \left\{ \frac{1}{2}\|ULV^\top - U\Sigma V^\top\|_{\text{Fro}}^2 + \alpha\|ULV^\top\|_* \right\} , \\ &= \operatorname*{arg\,min}_{L\in\mathbb{R}^{n\times s}} \left\{ \frac{1}{2}\|L - \Sigma\|_{\text{Fro}}^2 + \alpha\|L\|_* \right\} . \qquad (3.23) \\ & \qquad (3.24)\end{aligned}$$

Since Σ is a diagonal matrix with non-negative entries, the solution of (3.23) has to be a diagonal matrix with non-negative entries, too. As a consequence, (3.23) simplifies to

$$\begin{aligned}\tilde{l} &= \operatorname*{arg\,min}_{l \in \mathbb{R}^{\min(n,s)}_{\geq 0}} \left\{ \frac{1}{2}\|l - \sigma\|^2 + \alpha \sum_{j=1}^{\min(n,s)} l_j \right\} , \\ &= \operatorname*{arg\,min}_{l \in \mathbb{R}^{\min(n,s)}_{\geq 0}} \left\{ \frac{1}{2} \sum_{j=1}^{\min(n,s)} (l_j - \sigma_j)^2 + \alpha \sum_{j=1}^{\min(n,s)} l_j \right\} . \end{aligned} \tag{3.25}$$

where $l \in \mathbb{R}^{\min(n,s)}_{\geq 0}$ is the vector of diagonal entries of $\tilde{L}$, i.e. $\tilde{L} = \operatorname{diag}(l)$, and also the vector of singular values of $\tilde{L}$. The vector $\sigma \in \mathbb{R}^{\min(n,s)}_{\geq 0}$ denotes the singular values of Σ, i.e. $\Sigma = \operatorname{diag}(\sigma)$. Problem (3.25) has a closed-form solution that we recognise from last semester: the soft-thresholding of the singular values σ! Hence, we observe

$$\tilde{l}_j = \max(\sigma_j - \alpha, 0) , \quad \forall j \in \{1, \ldots, \min(n,s)\} .$$

We can therefore compute the solution of (3.20) via

$$\hat{L} = U\tilde{L}V^\top , \qquad \text{for} \qquad \tilde{L} = \operatorname{diag}(\tilde{l}) . \tag{3.26}$$

We will usually write the solution to (3.26) in the proximal map notation as

$$\hat{L} = (I + \alpha\, \partial \| \cdot \|_*)^{-1}(X) . \tag{3.27}$$

The practical implications of this proximal map are as follows. Given a matrix X, all singular values below the threshold α will be set to zero, therefore enforcing a lower rank of $\hat{L}$ compared to X if α is larger than at least the smallest singular value of X. All other singular values are reduced by the factor α.

3.2.4 The linearised Bregman iteration

Based on the previous considerations, we now want to derive an algorithm for the numerical solution of (3.22). We base our algorithm on the following generalisation of the Bregman proximal algorithm to non-smooth functions, commonly known as Bregman iteration:

$$w^{k+1} = \operatorname*{arg\,min}_{w \in \mathbb{R}^n} \left\{ E(w) + D_J^{p^k}(w, w^k) \right\} , \tag{3.28a}$$

$$p^{k+1} = p^k - \nabla E(w^{k+1}) , \tag{3.28b}$$

for initial values w^0 and $p^0 \in \partial J(w^0)$, where ∂J denotes the subdifferential of J as defined in Definition 3.1 and $D_J^p(w,v)$ is the generalised Bregman distance

$$D_J^p(w, v) = J(w) - J(v) - \langle p, w - v \rangle ,$$

for $p \in \partial J(v)$. We can derive a linearised variant of (3.28) for the choice $J(w) = \frac{1}{\tau}\left(\frac{1}{2}\|w\|^2 + R(w)\right) - E(w)$. Method (3.28) then reads

$$\begin{aligned}
w^{k+1} &= \operatorname*{arg\,min}_{w \in \mathbb{R}^n} \left\{ E(w) + \frac{1}{2\tau}\|w - w^k\|^2 + \frac{1}{\tau} D_R^{q^k}(w, w^k) - E(w) + E(w^k) + \langle \nabla E(w^k), w - w^k \rangle \right\} , \\
&= \operatorname*{arg\,min}_{w \in \mathbb{R}^n} \left\{ \frac{1}{2\tau}\|w - w^k\|^2 + \frac{1}{\tau} D_R^{q^k}(w, w^k) + \langle \nabla E(w^k), w - w^k \rangle \right\} , \\
&= \operatorname*{arg\,min}_{w \in \mathbb{R}^n} \left\{ \frac{1}{2\tau}\|w - w^k\|^2 + \frac{1}{\tau} D_R^{q^k}(w, w^k) + \frac{1}{\tau}\langle \tau\nabla E(w^k), w - w^k \rangle + \frac{1}{2\tau}\|\tau\nabla E(w^k)\|^2 \right\} , \\
&= \operatorname*{arg\,min}_{w \in \mathbb{R}^n} \left\{ \frac{1}{2} \left\| w - \left(w^k - \tau \nabla E(w^k) \right) \right\|^2 + D_R^{q^k}(w, w^k) \right\} , \\
&= (I + \partial R)^{-1} \left(w^k + q^k - \tau \nabla E(w^k) \right) , && (3.29a) \\
q^{k+1} &= q^k - \left(w^{k+1} - w^k + \tau \nabla E(w^k) \right) , && (3.29b)
\end{aligned}$$

for subgradients $q^k \in \partial R(w^k)$ and the short-hand notation

$$(I + \partial R)^{-1}(z) := \operatorname*{arg\,min}_{x \in \mathbb{R}^n} \left\{ \frac{1}{2}\|x - z\|^2 + R(x) \right\}$$

for the proximal map as defined in Section 2.7.3. In the following, we want to focus on the special case $E(w) = \frac{1}{2}\|Aw - b\|^2$, for a matrix $A \in \mathbb{R}^{s \times n}$ and a vector $b \in \mathbb{R}^s$. For this special case, (3.29) reads

$$\begin{aligned}
w^{k+1} &= (I + \partial R)^{-1} \left(w^k + q^k - \tau A^\top (A w^k - b) \right) , && (3.30a) \\
q^{k+1} &= q^k - \left(w^{k+1} - w^k + \tau A^\top (A w^k - b) \right) . && (3.30b)
\end{aligned}$$

If we assume that $(w^k + q^k)/\tau \in \mathcal{R}(A^\top)$, we can substitute $\tau A^\top b^k = w^k + q^k - \tau A^\top (A w^k - b)$, which modifies (3.30) to

$$\begin{aligned}
w^{k+1} &= (I + \partial R)^{-1} \left(\tau A^\top b^k \right) , && (3.31a) \\
b^{k+1} &= b^k - \left(A w^{k+1} - b \right) , && (3.31b)
\end{aligned}$$

with initial value $b^0 = b$. Combining both equations of (3.31) into one yields

$$b^{k+1} = b^k - \left(A \, (I + \partial R)^{-1} \left(\tau A^\top b^k \right) - b \right) . \qquad (3.32)$$

The reason behind reformulating (3.30) to (3.32) is that (3.32) is just gradient descent, i.e. Algorithm 1, applied to a very specific energy that we characterise with the following lemma.

Lemma 3.5 (Linearised Bregman iteration as gradient descent)**.** *The linearised Bregman iteration in the form of* (3.32) *is a gradient descent method with step-size one, i.e.*

$$b^{k+1} = b^k - \nabla G_\tau(b^k) ,$$

applied to the energy

$$G_\tau(b^k) := \frac{\tau}{2}\|A^\top b^k\|^2 - \langle b^k, b \rangle - \frac{1}{\tau}\tilde{R}(\tau A^\top b^k) .$$

Here $\tilde{R}$ denotes the Moreau-Yosida regularisation of the function R, i.e.

$$\begin{aligned}\tilde{R}(z) &:= \inf_{x\in\mathbb{R}^n}\left\{R(x) + \frac{1}{2}\|x-z\|^2\right\}\,,\\ &= R\left((I+\partial R)^{-1}(z)\right) + \frac{1}{2}\left\|(I+\partial R)^{-1}(z) - z\right\|^2\,.\end{aligned}$$

Proof. The proof is relatively straight-forward if we can compute the gradient of $\tilde{R}$, since the gradient of $\frac{\tau}{2}\|A^\top b^k\|^2 - \langle b^k, b\rangle$ simply reads $\tau AA^\top b^k - b$. In order to compute the gradient $\nabla\tilde{R}$, we first rewrite $\tilde{R}$ to

$$\begin{aligned}\tilde{R}(z) &= \inf_{x\in\mathbb{R}^n}\left\{R(x) + \frac{1}{2}\|x-z\|^2\right\}\,,\\ &= \inf_{x\in\mathbb{R}^n}\left\{R(x) + \frac{1}{2}\|x\|^2 - \langle x, z\rangle + \frac{1}{2}\|z\|^2\right\}\,,\\ &= \frac{1}{2}\|z\|^2 - \sup_{x\in\mathbb{R}^n}\left\{\langle x, z\rangle - R(x) - \frac{1}{2}\|x\|^2\right\}\,,\\ &= \frac{1}{2}\|z\|^2 - \left(\frac{1}{2}\|\cdot\|^2 + R\right)^*(z)\,.\end{aligned}$$

Here $F^*(z) := \sup_{x\in\mathbb{R}^n}\langle x, z\rangle - F(x)$ denotes the *convex conjugate* or *Fenchel conjugate* of a function F. Note that by definition, we observe $\nabla F^*(z) = x^*$, where $x^* = \arg\max_{x\in\mathbb{R}^n}\{\langle x, z\rangle - F(x)\}$. In case of $F(x) = \frac{1}{2}\|x\|^2 + R(x)$, this problem reads

$$\begin{aligned}x^* &= \arg\max_{x\in\mathbb{R}^n}\left\{\langle x, z\rangle - \frac{1}{2}\|x\|^2 - R(x)\right\}\,,\\ &= \arg\max_{x\in\mathbb{R}^n}\left\{-\frac{1}{2}\|x-z\|^2 - R(x)\right\}\,,\\ &= \arg\min_{x\in\mathbb{R}^n}\left\{\frac{1}{2}\|x-z\|^2 - R(x)\right\}\,,\\ &= (I+\partial R)^{-1}(z)\,.\end{aligned}$$

For the gradient of $\tilde{R}$ we therefore observe

$$\nabla\tilde{R}(z) = z - (I+\partial R)^{-1}(z)\,.$$

As an immediate consequence of the chain rule we have $\nabla\left(\left(\frac{1}{\tau}\tilde{R}\right)\circ\left(\tau A^\top\right)\right)(b^k) = A\nabla\tilde{R}(\tau A^\top b^k)$, and therefore conclude

$$\begin{aligned}\nabla G_\tau(b^k) &= \tau AA^\top b^k - b - \tau AA^\top b^k + A(I+\partial R)^{-1}\left(\tau A^\top b^k\right)\,,\\ &= A(I+\partial R)^{-1}\left(\tau A^\top b^k\right) - b\,,\end{aligned}$$

which also concludes the proof. □

Another important result for Algorithm (3.29) applied to functions of the form $E(w) = \frac{1}{2}\|Aw - b\|^2$ is that if we converge to a solution $\hat{w}$ that satisfies $A\hat{w} = b$ after a finite number of iterations, then this solution minimises $\frac{1}{2}\|\cdot\|^2 + R$ amongst all solution of $Aw = b$.

Lemma 3.6. *We assume that $w^0 + q^0 \in \mathcal{R}(A^\top)$. Suppose that after a finite number of iterations k^*, the iterate w^{k^*} of Algorithm (3.29) for $E(w) = \frac{1}{2}\|Aw - b\|^2$ satisfies $Aw^{k^*} = b$. Then w^{k^*} satisfies*

$$\frac{1}{2}\|w^{k^*}\|^2 + R(w^{k^*}) \leq \frac{1}{2}\|w\|^2 + R(w)\,,$$

for all w that satisfy $Aw = b$.

Proof. From (3.30) we easily verify

$$w^k + q^k = w^0 + q^0 - \tau A^\top \sum_{n=0}^{k-1}(Aw^n - b) \tag{3.33}$$

via induction. Since $J(w) = \frac{1}{\tau}\left(\frac{1}{2}\|w\|^2 + R(w)\right) - E(w)$ is convex, we know that $D_J^q(w, v) \geq 0$ for all u, v with $q \in \partial R(v)$. This is in particular true for any w that satisfies $Aw = b$ and $v = w^{k^*}$. Hence, we observe

$$\begin{aligned} J(w^{k^*}) &\leq J(w) - \left\langle \frac{w^{k^*} + q^{k^*}}{\tau} - A^\top(Aw^{k^*} - b), w - w^{k^*} \right\rangle \\ &= J(w) - \frac{1}{\tau}\langle w^0 + q^0, w - w^{k^*}\rangle - \left\langle A^\top \sum_{n=0}^{k^*}(Aw^n - b), w - w^{k^*} \right\rangle\,, \end{aligned}$$

where we have used Equation (3.33). Since we made the assumption that $w^0 + q^0 \in \mathcal{R}(A^\top)$, there exists an element ξ with $w^0 + q^0 = A^\top \xi$, and we further conclude

$$\begin{aligned} J(w^{k^*}) &\leq J(w) - \frac{1}{\tau}\langle \xi, Aw - Aw^{k^*}\rangle - \left\langle \sum_{n=0}^{k^*}(Aw^n - b), Aw - Aw^{k^*} \right\rangle\,, \\ &= J(w)\,, \end{aligned}$$

since $Aw = b$ and $Aw^{k^*} = b$. Substituting $J(w) = \frac{1}{\tau}\left(\frac{1}{2}\|w\|^2 + R(w)\right) - E(w)$ yields

$$\frac{1}{2}\|w^{k^*}\|^2 + R(w^{k^*}) - \tau E(w^{k^*}) \leq \frac{1}{2}\|w\|^2 + R(w) - \tau E(w)\,.$$

Both $E(w^{k^*})$ and $E(w)$ are zero since $Aw = b$ and $Aw^{k^*} = b$; hence, we have verified the assertion. □

In the following, we want to apply Algorithm (3.32) to the robust principal component analysis problem (3.22).

3.2.5 A Bregman algorithm for robust PCA

In order to solve (3.22), we choose $w = (L, S)$ and E and J as

$$E(L, S) = \frac{1}{2}\|L + S - X\|_{\text{Fro}}^2$$

and

$$J(L, S) = \frac{1}{\tau}\left(\frac{1}{2}\|L\|_{\text{Fro}}^2 + \gamma\alpha\|L\|_* + \frac{1}{2}\|S\|_{\text{Fro}}^2 + \gamma\|S\|_1\right) - E(L, S)\,,$$

Algorithm 7 Robust principal component analysis.

```
Specify: parameters $\gamma > 0$, $\alpha > 0$, index $K$
Initialise: $X^0 = X$, $\tau = 1/2$
Iterate:
1: for $k = 0, \dots, K-1$ do
2:     $L^{k+1} = (I + \gamma\,\alpha\,\partial\|\cdot\|_*)^{-1}(\tau X^k)$
3:     $S^{k+1} = (I + \gamma\,\partial\|\cdot\|_1)^{-1}(\tau X^k)$
4:     $X^{k+1} = X^k - \left(L^{k+1} + S^{k+1} - X\right)$
5: end for
return $L^K, S^K$.
```

for constants $\tau > 0$ and $\gamma > 0$. With the choices of E and J, we are exactly in the setting of (3.30) for $A = \begin{pmatrix} I & I \end{pmatrix}$ and $b = X$, and therefore can numerically solve (3.22) by iterating the updates (3.31), which for our choice of E and J read

$$L^{k+1} = (I + \gamma\,\alpha\,\partial\|\cdot\|_*)^{-1}(\tau X^k)\,, \tag{3.34a}$$

$$S^{k+1} = (I + \gamma\,\partial\|\cdot\|_1)^{-1}(\tau X^k)\,, \tag{3.34b}$$

$$X^{k+1} = X^k - \left(L^{k+1} + S^{k+1} - X\right)\,, \tag{3.34c}$$

for $X^0 := X$. Procedure (3.34) is summarised in Algorithm 7. Note that Algorithm 7 is an extremely simple procedure. It consists of computing the singular value decomposition of τX^k in every iteration, and soft-thresholding the singular values in the sense of (3.26) with threshold $\alpha\gamma$, and soft-thresholding all entries of τX^k with threshold γ. Subsequently, the matrix X^k is updated by subtracting the residual $L^{k+1} + S^{k+1} - X$ from it.

In identical fashion to the proof of Theorem 2.3 we can prove that Algorithm 7 converges at a rate of $1/k$ for $\tau \leq 1/\|A\|^2 = 1/2$, where k denotes the number of iterations. What remains to be shown is whether the limit of (3.34) converges to a solution of (3.22). This is addressed by Lemma 3.6. Thanks to Lemma 3.6 we know that if we converge to a solution of $L^{k^*} + S^{k^*} = X$, we do converge to a solution that guarantees

$$\frac{1}{2\gamma}\|S^{k^*}\|_{\text{Fro}}^2 + \|S^{k^*}\|_1 + \frac{1}{2\gamma}\|L^{k^*}\|_{\text{Fro}}^2 + \alpha\|L^{k^*}\|_* \leq \frac{1}{2\gamma}\|S\|_{\text{Fro}}^2 + \|S\|_1 + \frac{1}{2\gamma}\|L\|_{\text{Fro}}^2 + \alpha\|L\|_*\,,$$

for all L, S that satisfy $L + S = X$. This is not exactly identical to

$$\|S^{k^*}\|_1 + \alpha\|L^{k^*}\|_* \leq \|S\|_1 + \alpha\|L\|_*\,,$$

but for large γ it is a very good and tight approximation.

In the next section, we want to modify Algorithm 7 in order to guarantee a convergence rate of order $1/k^2$. Before we do so, we first study how to accelerate standard gradient descent.

3.2.6 Nesterov accelerated gradient descent

In this section we want to consider a modification of Algorithm 1, first proposed by Yurii Nesterov in Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence $\mathcal{O}(1/k^2)$. In Doklady AN USSR, volume 269, pages 543–547, 1983, with a convergence

Algorithm 8 Nesterov-accelerated gradient descent

```
Specify: Differentiable, convex function $E : \mathbb{R}^n \to \mathbb{R}$, step-size $\tau > 0$, index $K$, sequence $\{t_k\}_{k=1}^K$
Initialise: $w^0 \in \mathbb{R}^n$
Iterate:
1: for $k = 0, \ldots, K-1$ do
2:     $w^k = v^{k-1} - \tau \nabla E(v^{k-1})$
3:     $v^k = \left(1 - \frac{1}{t_{k+1}}\right) w^k + \frac{1}{t_{k+1}} u^k$
4:     $u^k = w^{k-1} + t_k \left(w^k - w^{k-1}\right)$
5: end for
return $w^K$.
```

rate of order $1/k^2$. This means we focus on the gradient descent procedure $w^{k+1} = w^k - \tau \nabla E(w^k)$ for now, but this time we consider a modification of the form

$$w^{k+1} = (1 + \beta_k) w^k - \beta_k w^{k-1} - \tau \nabla E \left((1 + \beta_k) w^k - \beta_k w^{k-1} \right) ,$$

respectively

$$w^k = v^{k-1} - \tau \nabla E(v^{k-1}) \,, \tag{3.35a}$$

$$v^k = \left(1 - \frac{1}{t_{k+1}}\right) w^k + \frac{1}{t_{k+1}} u^k \,, \tag{3.35b}$$

$$u^k = w^{k-1} + t_k \left(w^k - w^{k-1} \right) \,, \tag{3.35c}$$

for $\beta_k := \frac{t_k - 1}{t_{k+1}}$. We have summarised (3.35) in Algorithm 8. Note that the computational complexity is pretty-much the same as for gradient descent. The only change is that we replace the previous iterate w^k with a linear combination of the previous iterate and the iterate before that. Remarkably, this modification, with the right choice of sequence $\{t_k\}_{k=1}^\infty$ is sufficient to show the following improved convergence result.

Theorem 3.2 (Convergence of Algorithm 8)**.** *Suppose $E : \mathcal{C} \subset \mathbb{R}^n \to \mathbb{R}$ is a convex and $1/\tau$-smooth function with global minimiser $\hat{w}$ and the sequence $\{t_k\}_{k=1}^K$ that satisfies*

$$t_k^2 - t_k \leq t_{k-1}^2 \tag{3.36}$$

for all $k = 2, \ldots, K$ and $t_1 = 1$. Then the iterates $\{w^k\}_{k=1}^K$ of Algorithm 8 satisfy

$$t_K^2 e^K + \sum_{k=1}^{K-1} \rho_{k+1} e^k \leq \frac{d^0 - d^K}{\tau} \,,$$

with the short-hand notations

$$e^k := E(w^k) - E(\hat{w}) \,,$$

$$\rho_k := t_{k-1}^2 - t_k^2 + t_k \,,$$

$$d^k := \frac{1}{2} \| \hat{w} - u^k \|^2 \,.$$

Before we prove Theorem 3.2 we verify the following intermediate result first.

Corollary 3.1. *Suppose the same assumptions hold true as for Theorem 3.2, and define* $w^* := \arg\min_{w\in\mathbb{R}^n} \left\{E(w) + \frac{1}{2\tau}\|w - \overline{w}\|^2\right\}$ *for some* $\overline{w} \in \mathbb{R}^n$*. Then we observe*

$$E(w^*) + \frac{1}{2\tau}\|w - w^*\|^2 \leq E(w) + \frac{1}{2\tau}\|w - \overline{w}\|^2 \, ,$$

for any $w \in \mathbb{R}^n$*.*

Proof. With Lemma 2.4 and $J := \frac{1}{2\tau}\|\cdot\|^2 - E$ we can conclude

$$\begin{aligned}
&E(w^*) + D_E(w, w^*) + D_J(w, w^*) + D_J(w^*, \overline{w}) \\
&= E(w^*) + \frac{1}{2\tau}\|w - w^*\|^2 + D_J(w^*, \overline{w}) \\
&= E(w) + D_J(w, \overline{w}) \\
&= E(w) + \frac{1}{2\tau}\|w - \overline{w}\|^2 - D_E(w, \overline{w}) \\
&\leq E(w) + \frac{1}{2\tau}\|w - \overline{w}\|^2 \, ,
\end{aligned}$$

where the last inequality follows from the convexity of E and Corollary 2.1. With the same argumentation we derive $E(w^*) + \frac{1}{2\tau}\|w - w^*\|^2 \leq E(w^*) + \frac{1}{2\tau}\|w - w^*\|^2 + D_J(w^*, \overline{w})$, which concludes the proof. □

Proof of Theorem 3.2 (non-examinable). We closely follow the proof of Chambolle and Dossal in Antonin Chambolle and Charles Dossal. "On the convergence of the iterates of FISTA." (2015) and apply Corollary 3.1 with $w^* = w^{k+1}$, $\overline{w} = v^k$ and $w = \left(1 - \frac{1}{t_{k+1}}\right) w^k + \frac{1}{t_{k+1}}\hat{w}$, which yields

$$\begin{aligned}
w - w^* &= \left(1 - \frac{1}{t_{k+1}}\right) w^k + \frac{1}{t_{k+1}}\hat{w} - w^{k+1} \, , \\
&= \frac{1}{t_{k+1}}\hat{w} - \left(w^{k+1} - w^k + \frac{1}{t_{k+1}} w^k\right) \, , \\
&= \frac{1}{t_{k+1}}\left(\hat{w} - u^{k+1}\right) \, ,
\end{aligned}$$

and

$$\begin{aligned}
w - \overline{w} &= \left(1 - \frac{1}{t_{k+1}}\right) w^k + \frac{1}{t_{k+1}}\hat{w} - v^k \\
&= \left(1 - \frac{1}{t_{k+1}}\right) w^k + \frac{1}{t_{k+1}}\hat{w} - \left(1 - \frac{1}{t_{k+1}}\right) w^k - \frac{1}{t_{k+1}} u^k \\
&= \frac{1}{t_{k+1}}\left(\hat{w} - u^k\right) \, ,
\end{aligned}$$

and consequently

$$E(w^{k+1}) + \frac{1}{2\,\tau\, t_{k+1}^2}\left\|u^{k+1} - \hat{w}\right\|^2 \leq E\left(\left(1 - \frac{1}{t_{k+1}}\right) w^k + \frac{1}{t_{k+1}}\hat{w}\right) + \frac{1}{2\,\tau\, t_{k+1}^2}\left\|u^k - \hat{w}\right\|^2 \, .$$

Due to the convexity of E we can further estimate

$$E(w^{k+1}) + \frac{1}{2\,\tau\, t_{k+1}^2}\left\|u^{k+1} - \hat{w}\right\|^2 \leq \left(1 - \frac{1}{t_{k+1}}\right) E(w^k) + \frac{1}{t_{k+1}} E(\hat{w}) + \frac{1}{2\,\tau\, t_{k+1}^2}\left\|u^k - \hat{w}\right\|^2 \, ,$$

which we can rewrite to

$$E(w^{k+1}) - \left(1 - \frac{1}{t_{k+1}}\right) E(w^k) - \frac{1}{t_{k+1}} E(\hat{w}) \leq \frac{1}{2\,\tau\,t_{k+1}^2} \left(\left\| u^k - \hat{w} \right\|^2 - \left\| u^{k+1} - \hat{w} \right\|^2 \right) ,$$
$$= \frac{1}{2\,\tau\,t_{k+1}^2} \left(d^k - d^{k+1} \right) .$$

Note that the left-hand-side is equivalent to

$$E(w^{k+1}) - E(\hat{w}) - \left(1 - \frac{1}{t_{k+1}}\right) \left(E(w^k) - E(\hat{w}) \right) ,$$
$$= e^{k+1} - \left(1 - \frac{1}{t_{k+1}}\right) e^k ;$$

hence, after multiplication with t_{k+1}^2 the estimate above reads

$$t_{k+1}^2 e^{k+1} - \left(t_{k+1}^2 - t_{k+1}\right) e^k \leq \frac{1}{2\tau} \left(d^k - d^{k+1} \right) .$$

Summing up from $k = 0, \ldots, K-1$ yields

$$t_K^2 e^K + \sum_{k=0}^{K-1} \rho_{k+1} w^k \leq \frac{d^0 - d^K}{2\tau} ,$$

which concludes the proof. □

Several practical questions arise from this result. Which sequences $\{t_k\}_{k=1}^K$ satisfy (3.36)? And do these sequences guarantee a convergence rate of $1/K^2$? For exmaple, the sequence defined by $t_1 = 1$ and

$$t_{k+1} = \sqrt{t_k^2 + \frac{1}{4}} + \frac{1}{2} ,$$

for all $k = 2, \ldots, K-1$, satisfies (3.36). More generally, for any $a \geq 2$ the sequence $\{t_k\}_{k=1}^K$ with $t_1 = 1$ and

$$t_k = \frac{k + a - 1}{a}$$

satisfies (3.36). Do these sequences guarantee a convergence rate of $1/K^2$ for Algorithm 8? The answer is yes, which is obvious for the second sequence but in general it can shown by induction that any sequence that satisfies (3.36) with $t_1 = 1$ also satisfies $t_k \geq k$. Hence, we can conclude

$$e^K = \left(E(w^K) - E(\hat{w}) \right) \leq \frac{d^0 - d^K}{\tau\, K^2} ,$$

and consequently a convergence rate of $1/K^2$. It should be noted that for general, convex E no first-order algorithms with a quicker convergence rate exist due to Nemirovsky and Yudin in Arkadiĭ Semenovich Nemirovsky and David Borisovich Yudin. "Problem complexity and method efficiency in optimization." (1979). A proof for this lower bound can be found here. However, additional conditions on E can result in faster convergence, as well as algorithms that incorporate higher-order derivatives (such as the Newton-Raphson method) can have faster convergence rates.

Algorithm 9 Accelerated robust principal component analysis.

```
Specify: parameters $\gamma > 0$, $\alpha > 0$, index $K$
Initialise: $Y^0 = X$, $\tau = 1/2$ and sequence $\{t_k\}_{k=1}^{K+1}$
Iterate:
1: for $k = 0, \ldots, K - 1$ do
2:     $L^{k+1} = (I + \gamma\,\alpha\,\partial\|\cdot\|_*)^{-1}(\tau Y^k)$
3:     $S^{k+1} = (I + \gamma\,\partial\|\cdot\|_1)^{-1}(\tau Y^k)$
4:     $X^{k+1} = Y^k - \big(L^{k+1} + S^{k+1} - X\big)$
5:     $\beta_{k+1} = (t_{k+1} - 1)/t_{k+1}$
6:     $Y^{k+1} = (1 + \beta_{k+1})X^{k+1} - \beta_{k+1}X^k$
7: end for
return $L^K, S^K$.
```

3.2.7 An accelerated Bregman algorithm for robust PCA

Thanks to Lemma 3.5 we know that Algorithm 7 is actually a gradient descent method, albeit a very strange-looking one. In the previous section we have learned how to accelerate gradient descent, and we now want to apply this acceleration strategy to our problem of robust PCA. Note that (3.34) in the form of (3.32) reads

$$X^{k+1} = X^k - \Big((I + \gamma\,\alpha\,\partial\|\cdot\|_*)^{-1}(\tau X^k) + (I + \gamma\,\partial\|\cdot\|_1)^{-1}(\tau X^k) - X\Big)\,.$$

Recall that the only difference between gradient descent and accelerated gradient descent is the replacement of X^k with a linear combination of X^k and X^{k-1} in the form of

$$\begin{aligned} X^{k+1} = (1+\beta_k)X^k - \beta_k X^{k-1} - \Big(&(I + \gamma\,\alpha\,\partial\|\cdot\|_*)^{-1}\Big(\tau((1+\beta_k)X^k - \beta_k X^{k-1})\Big) \\ &+ (I + \gamma\,\partial\|\cdot\|_1)^{-1}\Big(\tau((1+\beta_k)X^k - \beta_k X^{k-1})\Big) - X\Big)\,. \end{aligned} \tag{3.37}$$

If we define $Y^k := (1+\beta_k)X^k - \beta_k X^{k-1}$, we can rewrite (3.37) to

$$L^{k+1} = (I + \gamma\,\alpha\,\partial\|\cdot\|_*)^{-1}(\tau Y^k)\,, \tag{3.38a}$$

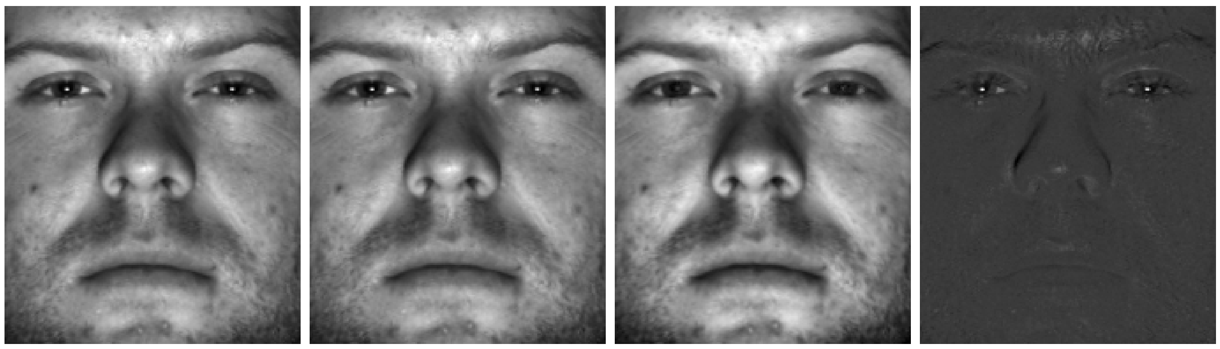
$$S^{k+1} = (I + \gamma\,\partial\|\cdot\|_1)^{-1}(\tau Y^k)\,, \tag{3.38b}$$

$$X^{k+1} = Y^k - \Big(L^{k+1} + S^{k+1} - X\Big)\,, \tag{3.38c}$$

$$Y^{k+1} = (1+\beta_{k+1})X^{k+1} - \beta_{k+1}X^k\,. \tag{3.38d}$$

Equations (3.38) are also summarised in Algorithm 9. As mentioned earlier, the complexity of the individual steps of Algorithm 9 is about the same as in Algorithm 7, but the convergence speed is of order $1/k^2$ instead of $1/k$. This is a nice demonstration of the power of mathematics, as it is hard to imaging one would come up with a modified strategy such as Algorithm 9 if one had not studied the proof for the convergence of gradient descent.

In Figure 3.2 we have visualised an example for a robust PCA computation that you will re-create as part of your coursework. Here we have taken the first human of the 28 human subjects from the Yale Faces B dataset and stored its 64 different illumination images of dimension 192×168 for the first pose in a matrix $X \in \mathbb{R}^{32256 \times 64}$. We then compute the robust PCA with Algorithm 7 and Algorithm 9 for the parameter choices $\gamma = 10$ and $\alpha = \sqrt{192}$. The first of the 64 images is visualised in Figure 3.2 (the first column of X), together with the robust PCA approximation



(a) Original **(b)** Approximation **(c)** Low-rank part **(d)** Sparse part

Figure 3.2: From left to right: the first illumination image of the first pose of the first human of the Yale B faces database, its approximation which is the sum of a low-rank and a sparse matrix, the low-rank matrix and the sparse matrix.

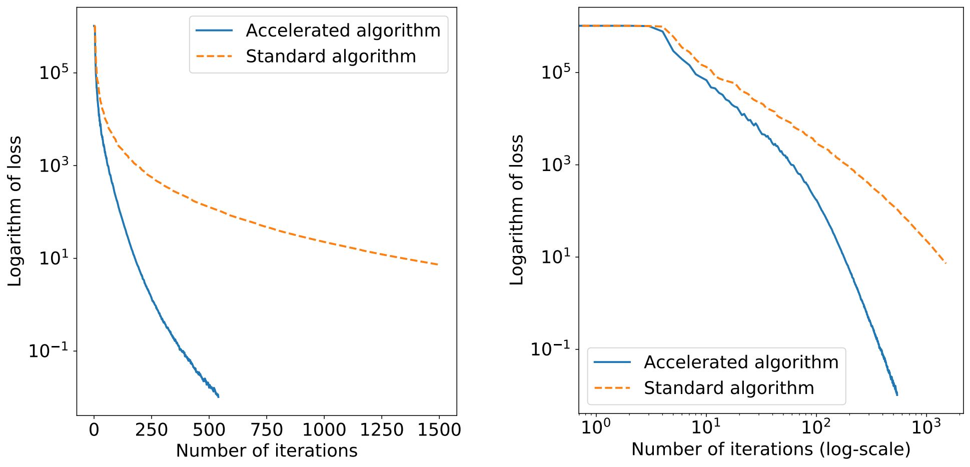


Figure 3.3: This is an empirical validation of the different convergence rates of Algorithm 7 and Algorithm 9.

(first column of $L^K + S^K$), consisting of the low-rank (first column of L^K) and the sparse part (first column of S^K).

In Figure 3.3 we see the comparison of the convergence rates of Algorithm 7 and Algorithm 9. Not that the loss value $\frac{1}{2}\|L^k + S^k - X\|_{\text{Fro}}^2$ that is achieved with Algorithm 7 after $K = 1500$ iterations is already achieved after around $K = 225$ iterations with Algorithm 9. This means we have to compute 1225 fewer SVDs to achieve the same computational accuracy and therefore save an awful lot of computational time.

3.2.8 Matrix completion

We conclude our very brief survey of matrix factorisation methods with a section on *matrix completion*. Matrix completion is a task that in the context of unsupervised machine learning finds application in *recommender systems*. Recommender systems are, as the name suggests, systems that aim at making recommendations to a customer or user. Often, this involves ranking books, movies, songs, products etc. that a user has not read, seen, heard or purchased before, based on

Algorithm 10 Matrix completion.

```
Specify: parameters $\gamma > 0$, stopping index $K$
Initialise: Set of known matrix indices $\Omega$, $z^0 = P_\Omega X$ and $\tau \leq 1$
Iterate:
1: for $k = 0, \ldots, K-1$ do
2:     $L^{k+1} = (I + \gamma\,\partial\|\cdot\|_*)^{-1}\left(\tau P_\Omega^\top z^k\right)$,
3:     $z^{k+1} = z^k - \left(P_\Omega L^{k+1} - z\right)$,
4: end for
return $L^K$.
```

ratings of other books, movies, songs, products by the same user and ratings of the seen and unseen books, movies, songs, products by other users. Completing missing ratings can be formulated as a matrix completion problem. We illustrate this with the following example. Suppose we have a several customers of food places near Queen Mary's Mile End campus and record some of their ratings (from 1 to 10) as follows[1]:

	The Curve	Verdi's	Greedy Cow	Döner Kebab	KFC
Justin	6	?	?	3	6
Kathrin	?	8	4	?	3
Martin	4	5	6	7	?
Primoz	3	?	7	?	?
Shabnam	3	5	?	9	?
Wolfram	7	?	?	?	?

The rows of this matrix represent different customers while the columns represent different food places. Every entry at row i and column j represents a rating by customer i of food place j. As most customers have only visited a subset of all restaurants, we only have an incomplete set of ratings, and hence, missing entries in the ratings matrix denoted with a question mark. The problem of matrix completion is to fill the missing entries with some meaningful entries based on some a-priori assumptions.

The first a-priori assumption that we want to make is that there is only a small number of different customer types and that every rating can be modelled as a linear combination of the ratings from all customer types. In mathematical terms this means that we assume that the matrix with all ratings has a low rank. We want to formulate this problem as the optimisation problem

$$\hat{L} = \operatorname*{arg\,min}_{L \in \mathbb{R}^{n \times s}} \left\{ \|L\|_* \qquad \text{subject to} \qquad P_\Omega L = P_\Omega X \right\} . \tag{3.39}$$

Here $\|\cdot\|_*$ denotes the nuclear norm that we have introduced before, which implicitly penalises the rank of the matrix $\hat{L}$ that we wish to recover. In order to guarantee that the entries for which $\hat{L}$ is known to match the provided ratings, we enforce the constraint $P_\Omega \hat{L} = P_\Omega X$. Here, $P_\Omega : \mathbb{R}^{n \times s} \to \mathbb{R}^r$ denotes the projection onto the r known entries of a matrix, provided by the set Ω, and $P_\Omega X$ are the known ratings at these locations. If we consider the matrix

$$X = \begin{pmatrix} -1 & 4 & ? \\ ? & -2 & 7 \end{pmatrix}$$

[1]All ratings are made up and do not represent ratings of real people of the same name

for example, we know the indices $\Omega = \{(1,1),(1,2),(2,2),(2,3)\}$ and therefore can project onto those, i.e.

$$P_\Omega X = \begin{pmatrix} -1 & 4 & -2 & 7 \end{pmatrix}^\top .$$

Note that this operator is linear and its transpose operation $P_\Omega^\top : \mathbb{R}^r \to \mathbb{R}^{n\times s}$ is

$$P_\Omega^\top \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix} = \begin{pmatrix} z_1 & z_2 & 0 \\ 0 & z_3 & z_4 \end{pmatrix} .$$

We can numerically solve (3.39) with the linearised Bregman iteration (3.29), which for (3.39) reads

$$\begin{aligned} L^{k+1} &= (I + \gamma\,\partial\|\cdot\|_*)^{-1}\left(\tau P_\Omega^\top z^k\right) , \\ z^{k+1} &= z^k - \left(P_\Omega L^{k+1} - z\right) , \end{aligned}$$

for $z^0 = z := P_\Omega X$, $\tau \leq 1$ and $\gamma > 0$, which is also summarised in Algorithm 10. It's Nesterov-accelerated counterpart reads

$$\begin{aligned} L^{k+1} &= (I + \gamma\,\partial\|\cdot\|_*)^{-1}\left(\tau P_\Omega^\top y^k\right) , \\ z^{k+1} &= y^k - \left(P_\Omega L^{k+1} - z\right) , \\ y^{k+1} &= (1 + \beta_{k+1}) z^{k+1} - \beta_{k+1} z^k , \end{aligned}$$

which is summarised in Algorithm 11.

To wrap up this section, we want to show that we can also derive a matrix completion model that is robust to sparse outliers, based on a generalisation of robust PCA that also includes the projection operator P_Ω. The model either reads

$$(\hat{L}, \hat{S}) = \operatorname*{arg\,min}_{L,S\in\mathbb{R}^{n\times s}} \left\{ \gamma\left(\|S\|_1 + \alpha\|L\|_*\right) \quad \text{subject to} \quad P_\Omega(L+S) = P_\Omega X \right\} ,$$

if we want to allow sparse outliers for all matrix entries, or

$$(\hat{L}, \hat{s}) = \operatorname*{arg\,min}_{L\in\mathbb{R}^{n\times s},\, \vec{s}\in\mathbb{R}^r} \left\{ \gamma\left(\|\vec{s}\|_1 + \alpha\|L\|_*\right) \quad \text{subject to} \quad \vec{s} + P_\Omega L = P_\Omega X \right\} ,$$

if we want to allow sparse outliers only for the known entries. Algorithms can be derived in almost identical fashion as in the robust PCA case.

3.3 Autoencoders

In Section 2.8 we have introduced the concept of (deep) neural networks in the context of supervised machine learning. In this section, we want to look at deep neural networks for unsupervised machine learning. A very popular way of using deep learning in unsupervised machine learning is via the concept of an *autoencoder*. An autoencoder $f_w : \mathbb{R}^n \to \mathbb{R}^n$ is a composition of two neural networks; one network $e_{w_e} : \mathbb{R}^n \to \mathbb{R}^r$ is called the *encoder* and the other network $d_{w_d} : \mathbb{R}^r \to \mathbb{R}^n$ is

Algorithm 11 Accelerated matrix completion.

```
Specify: parameters $\gamma > 0$, stopping index $K$
Initialise: Set of known matrix indices $\Omega$, $y^0 = P_\Omega X$, $\tau \leq 1$ and sequence $\{t_k\}_{k=1}^{K+1}$
Iterate:
1: for $k = 0, \ldots, K-1$ do
2:     $L^{k+1} = (I + \gamma\,\partial\|\cdot\|_*)^{-1}\left(\tau P_\Omega^\top y^k\right)$,
3:     $z^{k+1} = y^k - \left(P_\Omega L^{k+1} - z\right)$,
4:     $\beta_{k+1} = (t_{k+1} - 1)/t_{k+1}$
5:     $y^{k+1} = (1 + \beta_{k+1}) z^{k+1} - \beta_{k+1} z^k$,
6: end for
return $L^K$.
```

called the *decoder*. The autoencoder f_w is simply the composition of the decoder and the encoder, i.e. $f_w(x) := d_{w_d}\left(e_{w_e}(x)\right)$, for encoder weights w_e and decoder weights w_d and overall weights $w = (w_e, w_d)$. Note that we usually (but not always) assume $r \ll n$ and that we usually compose the neural networks of affine-linear transformations and point-wise nonlinear activation functions as described in Section 2.8. Given a set of unlabelled data samples $\{x_i\}_{i=1}^s$, the goal of traditional autoencoders is to approximate each sample, i.e. we aim to find parameters $w = (w_e, w_d)$ such that

$$f_w(x_i) = d_{w_d}\left(e_{w_e}(x_i)\right) \approx x_i$$

is (approximately) satisfied for all $i \in \{1, \ldots, s\}$. For $r < n$, this is a nonlinear dimensionality reduction problem. As in previous sections and chapters, we estimate the parameters by minimising an empirical risk formulation of the form

$$\min_w \frac{1}{s} \sum_{i=1}^s \ell\left(x_i, f_w(x_i)\right) ,$$

for appropriate loss functions ℓ, such as $\ell(x, y) = \frac{1}{2}\|x - y\|^2$ for example.

Example 3.3 (Linear autoencoder)**.** Suppose both encoder and decoder are linear transformations, i.e. $e_{W_e} = W_e^\top$ and $d_{W_d} = W_d$ with $W_e, W_d \in \mathbb{R}^{n\times r}$. If we choose the mean-squared error as the empirical risk that we minimise with respect to W_e and W_d, then we have to solve

$$\min_{W_e, W_d} \frac{1}{2s} \sum_{i=1}^s \|W_d W_e^\top x_i - x_i\|^2 .$$

If we store all x_i as columns in a matrix $X \in \mathbb{R}^{n\times s}$, we can rewrite the previous problem to

$$\min_{W_e, W_d} \frac{1}{2s} \|W_d W_e^\top X - X\|_{\mathrm{Fro}}^2 .$$

This problem should look very familiar; what is the optimal solution for W_e and W_d?

Denoising autoencoder

Instead of training autoencoders to map signals onto the same signals, one can instead train autoencoders that map noisy versions of the signals onto the clean versions of the signals. In the simplest case, we assume that the input signals are corrupted by additive, normal-distributed noise, which means that we train w by solving

$$\min_w \frac{1}{s} \sum_{i=1}^s \ell\left(x_i, f_w(x_i + n_i)\right) ,$$

where the n_i's are instances of a normal distributed random variable with mean zero and standard deviation σ. Similar models can be designed for different noise models, or even for more general operators that manipulate the inputs in a specific fashion.

In the following sections we want to address two key questions that we had not (or only had partially) addressed in Section 2.8: an efficient computational strategy for the empirical risk minimisation for large s and an efficient strategy to automatically compute the gradient of complicated function compositions without having to implement the backpropagation algorithm manually.

3.3.1 Stochastic gradient descent

In Section 2.8.1 we have shown how a deep (feed-forward) neural network can be trained with the backpropagation algorithm and gradient descent. In practice, gradient descent is usually replaced with a modification known as stochastic (sub-)gradient descent or variants of it. One downside when using gradient descent for applications with extremely large datasets is that the computation of the entire gradient multiple times may easily become infeasible. We basically envision a scenario where the energy E is of the form of an empirical risk (2.18), i.e.

$$E(w) := \frac{1}{s} \sum_{i=1}^s e_i(w) \, , \tag{3.40}$$

for s functions $e_i : \mathbb{R}^n \to \mathbb{R}$, with extremely large s. If we were to apply gradient descent, we would for each iteration have to compute ∇E. Instead, we could update our variables only w.r.t. a partial gradient in the sense of

$$w^{k+1} = w^k - \tau_k \nabla e_i(w^k) \, , \tag{3.41}$$

where $i \in \{1, \ldots, s\}$ are instances of a uniformly random variable and $\{\tau_k\}_{k=0}^{K-1}$ is a sequence of positive step-size parameters. Alternatively, we could think of $e_1(w), e_2(w), \ldots, e_s(w)$ as a finite number of outcomes of some random variable $\mathcal{E}(w)$ that occur with uniform probabilities $p_i = 1/s$ for $i = 1, \ldots, s$. The expected value of the corresponding gradients $\nabla e_i(w)$, i.e.

$$\mathbb{E}_i \left[\nabla e_i(w)\right] = \sum_{j=1}^s \nabla e_i(w) P(i = j) = \sum_{j=1}^s \frac{1}{s} \nabla e_j(w) = \nabla \left(\frac{1}{s} \sum_{j=1}^s e_j(w) \right) = \nabla E(w) \, ,$$

is then simply the gradient of E, and the stochastic gradient descent (SGD) performs a gradient step on one of the finite outcomes of the random variable $\nabla \mathcal{E}$. We summarise SGD in Algorithm 12. Note that SGD, despite its name, is in general not a descent method, which means that we cannot guarantee $E(w^{k+1}) \leq E(w^k)$ for subsequent iterates. We leave it as an exercise to the reader to find a counterexample. We now want to prove a convergence result in expectation, for which we first define the variance of a vector v as $\mathrm{Var}_x(v(x)) := \mathbb{E}_x\left[\|v(x)\|^2\right] - \|\mathbb{E}_x[v(x)]\|^2$ and the weighted average of the iterates as $\overline{w}^K := \left(\sum_{k=1}^K \tau_k \, w^k\right) / \left(\sum_{k=0}^{K-1} \tau_k\right)$. Then we can show the following result.

Algorithm 12 Stochastic gradient descent

```
Specify: Function $E : \mathbb{R}^n \to \mathbb{R}$ of the form (3.40), where each $e_i$ is continuously differentiable and convex, index $K$, positive step-sizes $\{\tau_k\}_{k=0}^{K-1}$,
Initialise: $w^0 \in \mathbb{R}^n$
Iterate:
1: for $k = 0, \ldots, K-1$ do
2:     Pick index $i \sim \mathcal{U}$
3:     $w^{k+1} = w^k - \tau_k \nabla e_i(w^k)$
4: end for
return $w^K$.
```

Theorem 3.3. *Let $E : \mathbb{R}^n \to \mathbb{R}$ be a convex function of the form* (3.40) *which is $1/\tau$-smooth in the sense of Definition 2.7 for $\tau > 0$. Suppose $\hat{w}$ denotes a global minimiser of E and that $Var_i(\nabla e_i(w)) \leq \sigma^2$ is satisfied for some positive constant σ and all $i \in \{1, \ldots, s\}$. Then the weighted average of the iterates $\overline{w}^K := \left(\sum_{k=1}^K \tau_k \, w^k\right) / \left(\sum_{k=0}^{K-1} \tau_k\right)$ for iterates w^k of Algorithm 12 satisfy*

$$\mathbb{E}_i \left[E(\overline{w}^K) \right] - E(\hat{w}) \leq \frac{\|w^0 - \hat{w}\|^2}{2 \sum_{k=0}^{K-1} \tau_k} + \frac{\sigma^2 \sum_{k=0}^{K-1} \tau_k^2}{2 \sum_{k=0}^{K-1} \tau_k} \, . \tag{3.42}$$

Proof (non-examinable). From the $1/\tau$-smoothness of E we can conclude $D_J(w_1, w_2) \geq 0$ for $J(w) := \frac{1}{2\tau}\|w\|^2 - E(w)$, which yields the estimates

$$E(w^{k+1}) \leq E(w^k) + \langle \nabla E(w^k), w^{k+1} - w^k \rangle + \frac{1}{2\tau} \|w^{k+1} - w^k\|^2 \, , \tag{3.43}$$

for $w_1 = w^{k+1}$ and $w_2 = w^k$. Inserting the stochastic gradient descent update (3.41) into (3.43) leads to

$$E(w^{k+1}) \leq E(w^k) - \tau_k \langle \nabla E(w^k), \nabla e_i(w^k) \rangle + \frac{\tau_k^2}{2\tau} \|\nabla e_i(w^k)\|^2 \, .$$

If we take the expected value on both sides of the inequality we observe

$$\begin{aligned}
\mathbb{E}_i \left[E(w^{k+1}) \right] &\leq E(w^k) - \tau_k \|\nabla E(w^k)\|^2 + \frac{\tau_k^2}{2\tau} \mathbb{E}_i \left[\|\nabla e_i(w^k)\|^2 \right] \\
&= E(w^k) - \tau_k \|\nabla E(w^k)\|^2 + \frac{\tau_k^2}{2\tau} \left(\left\| \mathbb{E}_i \left[\nabla e_i(w^k) \right] \right\|^2 + \mathrm{Var}_i \left[\nabla e_i(w^k) \right] \right) \, , \\
&= E(w^k) - \tau_k \|\nabla E(w^k)\|^2 + \frac{\tau_k^2}{2\tau} \left(\left\| \nabla E(w^k) \right\|^2 + \mathrm{Var}_i \left[\nabla e_i(w^k) \right] \right) \, , \\
&\leq E(w^k) - \tau_k \|\nabla E(w^k)\|^2 + \frac{\tau_k}{2} \left(\left\| \nabla E(w^k) \right\|^2 + \mathrm{Var}_i \left[\nabla e_i(w^k) \right] \right) \, , \\
&= E(w^k) - \frac{\tau_k}{2} \|\nabla E(w^k)\|^2 + \frac{\tau_k}{2} \mathrm{Var}_i \left[\nabla e_i(w^k) \right] \, , \\
&\leq E(w^k) - \frac{\tau_k}{2} \|\nabla E(w^k)\|^2 + \frac{\tau_k \, \sigma^2}{2} \, .
\end{aligned}$$

Due to the convexity of E, we can further estimate

$$E(w^k) \leq E(\hat{w}) + \langle \nabla E(w^k), \hat{w} - w^k \rangle \, . \tag{3.44}$$

Combining estimates (3.43) and (3.44) then yields

$$\begin{aligned}
\mathbb{E}_i\left[E(w^{k+1})\right] &\leq E(\hat{w}) + \langle \nabla E(w^k), \hat{w} - w^k\rangle - \frac{\tau_k}{2}\|\nabla E(w^k)\|^2 + \frac{\tau_k\,\sigma^2}{2}\,,\\
&= E(\hat{w}) + \langle \mathbb{E}_i[\nabla e_i(w^k)], \hat{w} - w^k\rangle - \frac{\tau_k}{2}\mathbb{E}_i\|\nabla e_i(w^k)\|^2 + \frac{\tau_k\,\sigma^2}{2}\,,\\
&= E(\hat{w}) + \mathbb{E}_i\left[\langle \nabla e_i(w^k), \hat{w} - w^k\rangle - \frac{\tau_k}{2}\|\nabla e_i(w^k)\|^2\right] + \frac{\tau_k\,\sigma^2}{2}\\
&= E(\hat{w}) + +\mathbb{E}_i\left[\frac{1}{\tau_k}\langle w^k - w^{k+1}, \hat{w} - w^k\rangle - \frac{1}{2\tau_k}\|w^k - w^{k+1}\|^2\right] + \frac{\tau_k\,\sigma^2}{2}\\
&= E(\hat{w}) + \frac{1}{\tau_k}\mathbb{E}_i\left[\frac{1}{2}\left(\|w^k - \hat{w}\|^2 - \|w^{k+1} - \hat{w}\|^2\right)\right] + \frac{\tau_k\,\sigma^2}{2}\,.
\end{aligned}$$

Summing up both sides of the estimate from $k = 0$ to $k = K - 1$ yields

$$\begin{aligned}
\sum_{k=0}^{K-1} \tau_k\left(\mathbb{E}_i\left[E(w^{k+1})\right] - E(\hat{w})\right) &\leq \mathbb{E}_i\left[\frac{1}{2}\left(\|w^0 - \hat{w}\|^2 - \|w^K - \hat{w}\|^2\right)\right] + \frac{\sigma^2\sum_{k=0}^{K-1}\tau_k^2}{2}\,,\\
&\leq \mathbb{E}_i\left[\frac{1}{2}\|w^0 - \hat{w}\|^2\right] + \frac{\sigma^2\sum_{k=0}^{K-1}\tau_k^2}{2}\,,\\
&= \frac{1}{2}\|w^0 - \hat{w}\|^2 + \frac{\sigma^2\sum_{k=0}^{K-1}\tau_k^2}{2}\,.
\end{aligned}$$

Due to the convexity of E and the linearity of the expectation we can estimate

$$\begin{aligned}
\sum_{k=0}^{K-1} \tau_k\left(\mathbb{E}_i\left[E(w^{k+1})\right] - E(\hat{w})\right) &= \sum_{k=0}^{K-1} \tau_k\left(\mathbb{E}_i\left[\sum_{k=0}^{K-1}\frac{\tau_k}{\sum_{k=0}^{K-1}\tau_k}E(w^{k+1})\right] - E(\hat{w})\right)\\
&\geq \sum_{k=0}^{K-1} \tau_k\left(\mathbb{E}_i\left[E\left(\frac{\sum_{k=0}^{K-1}\tau_k\,w^{k+1}}{\sum_{k=0}^{K-1}\tau_k}\right)\right] - E(\hat{w})\right)\,,\\
&= \sum_{k=0}^{K-1} \tau_k\left(\mathbb{E}_i\left[E\left(\overline{w}^K\right)\right] - E(\hat{w})\right)\,,
\end{aligned}$$

with the help of Jensen's inequality. Combining this estimate with the previous one and dividing by $\sum_{k=0}^{K-1}\tau_k$ then concludes the proof. □

Note that constant step-sizes ($\tau_k = \tau$ for all k) or fixed step-lengths (e.g. $\tau_k = \tau/\|\nabla e_i(w^k)\|$) do not necessarily guarantee convergence, as we cannot guarantee that the right-hand-side of (3.42) converges to zero for increasing k. We can, however, guarantee convergence for diminishing step-sizes that ensure

$$\lim_{k\to\infty}\tau_k = 0\,, \qquad \text{and} \qquad \sum_{k=0}^{\infty}\tau_k = \infty\,, \tag{3.45}$$

as for those diminishing step-sizes it can be shown that

$$\lim_{K\to\infty}\frac{\sum_{k=0}^{K-1}\tau_k^2}{\sum_{k=0}^{K-1}\tau_k} = 0$$

is also satisfied. Examples for sequences that satisfy (3.45) are $\tau_k = \tau/(k+1)$ or $\tau_k = \tau/\sqrt{k+1}$ for a constant $\tau > 0$. Note, however, that we won't be able to guarantee a convergence rate $\mathcal{O}(1/k)$ as for the standard gradient descent algorithm, but only $\mathcal{O}(1/\sqrt{k})$.

Note that we can derive a proximal stochastic gradient descent method in analogy to classical gradient descent by wrapping a proximal map around the update.

3.3.2 Automatic differentiation

In this section we want to briefly describe the concept of *automatic differentiation*. Instead of computing derivatives by hand or symbolically, automatic differentiation numerically evaluates derivatives of (complicated) functions automatically. The key ingredient here is the chain rule that allows us to automatically evaluate derivatives of function-compositions numerically at specific points. In terms of evaluation of the chain rule, one mostly distinguishes between *forward accumulation* and *reverse accumulation*. In forward accumulation, one traverses through the chain rule from inside to outside, while in reverse accumulation one traverses from outside to inside.

3.3.3 Dual numbers and forward accumulation

A *dual number* x is a number of the form

$$x = a + \varepsilon b \,,$$

where ε is a constant similar to the imaginary unit i in complex numbers. But where we have $i^2 = -1$, we have $\varepsilon^2 = 0$ for dual numbers. We refer to a as the *real* part and to b as the *dual* part of x. You may wonder how the concept of dual numbers is useful for automatic differentiation. We will answer this question indirectly with two examples. Suppose we have two dual numbers $x = a + \varepsilon b$ and $y = c + \varepsilon d$. Multiplying both numbers yields the product

$$xy = (a + \varepsilon b)(c + \varepsilon d) = ac + \varepsilon(ad + bc) + \underbrace{\varepsilon^2}_{=0} bd = ac + \varepsilon(ad + bc) \,.$$

The product is another dual variable with the product ac as its real part and $ad + bc$ as its dual part. Note that for $b = 1$ and $d = 0$ the dual part reduces to c, whereas for $b = 0$ and $d = 1$ it reduces to a. If we were to ignore the real parts of x and y, we'd simply obtain the partial derivatives with respect to a and b.
Lets look at the quotient of x and y as another example, i.e.

$$\frac{x}{y} = \frac{a + \varepsilon b}{c + \varepsilon d} = \frac{(a + \varepsilon b)(c - \varepsilon d)}{(c + \varepsilon d)(c - \varepsilon d)} = \frac{ac + \varepsilon(bc - ad) + \varepsilon^2 bd}{c^2 - \varepsilon^2 d^2} = \frac{ac + \varepsilon(bc - ad)}{c^2} = \frac{a}{c} + \varepsilon \frac{bc - ad}{c^2} \,.$$

Similar to the previous example, the real part of x/y is simply the quotient a/c of the real parts, whereas the dual part resembles the quotient rule $(bc - ad)/c^2$. For $b = 1$ and $d = 0$, the dual component simplifies to c, while for $b = 0$ and $d = 1$ it simplifies to $-a/c^2$. Again, these simplified dual components are equal to the partial derivatives $\frac{\partial}{\partial a} \frac{a}{c}$ and $\frac{\partial}{\partial c} \frac{a}{c}$.

We are now going to show that this is not just a coincidence, but indeed true for any differentiable function that acts on a dual number. The Taylor series of an infinitely differentiable function $f(x)$ around a point y is defined as

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(y)}{n!} (x - y)^n \,,$$

where $f^{(n)}$ denotes the n-th derivative of f. If the argument x is a dual number $x = a + \varepsilon b$, and we consider the Taylor series around the point a, we observe

$$f(x) = f(a + \varepsilon b) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a) b^n \varepsilon^n}{n!} = f(a) + \varepsilon b f'(a) \,.$$

Here, the last equality again follows from the fact that $\varepsilon^2 = 0$. Hence, applying f to a dual number yields a new dual number, where the real part is f applied to the real part and the dual part is $bf'(a)$. In the previous examples, we had $f(x) = xy$ and $f(x) = x/y$.
For function compositions $f(x) = h(g(x))$ acting on a dual variable $x = a + \varepsilon b$ we observe

$$g(x) = g(a) + \varepsilon b g'(a) \,,$$

and

$$h(g(x)) = h(g(a) + \varepsilon b g'(a)) = h(g(a)) + \varepsilon h'(g(a)) b g'(a) \,.$$

In terms of computation, when computing this composition of functions acting on dual numbers, we first evaluate g on x and then h on $g(a) + \varepsilon b g'(a)$. In terms of differentiation, we first compute the derivative $g'(a)$ and then the derivative $h'(g(a))$. This order of computing the derivatives in order to evaluate the chain rule is also known as *forward accumulation*. For a function composition $y = h(g(x))$, we can define $w_0 = x$, $w_1 = g(w_0)$ and $w_2 = h(w_1) = y$. Computing the chain rule yields

$$\frac{dy}{dx} = \frac{dw_2}{dw_0} = \frac{dw_2}{dw_1} \frac{dw_1}{dw_0} \,.$$

In forward accumulation, one evaluates the chain rule via the recursive relation

$$\frac{dw_i}{dx} = \frac{dw_i}{dw_{i-1}} \frac{dw_{i-1}}{dx} \,,$$

i.e. we first evaluate dw_1/dx, then dw_2/dw_1. This coincides with the order of computation when evaluating dual numbers, which is why forward accumulation and automatic differentiation with dual numbers are effectively the same thing.

3.3.4 Reverse accumulation

As the name suggests, reverse accumulation works in opposite order compared to forward accumulation, i.e. we evaluate the chain rule via the recursive relation

$$\frac{dy}{dw_i} = \frac{dy}{dw_{i+1}} \frac{dw_{i+1}}{dw_i} \,.$$

This means we first evaluate dy/dw_1, then dw_1/dw_0. We have already seen an example of reverse accumulation in Section 2.8.1: the backpropagation algorithm. Evaluating the forward pass and subsequent computation of the partial derivatives in reverse order is a special case of reverse accumulation.