Non-linear optimisation and learning

Authors: Mengwu Guo, Tjeerd Jan Heeringa and Ruben Hoeksma[1]

Department of Applied Mathematics
Faculty EEMCS
University of Twente

November 17, 2023

[1]Chapter 2 and 3 and Appendices A and B are based on the lecture notes for the course *Mathematical Optimization* by Ulrich Faigle, Walter Kern and Georg Still.

Contents

CHAPTER 1

Introduction

These are lecture notes for the University of Twente B-AM course Non-linear Optimisation and Learning. Part 1 treats non-linear unconstrained optimisation and Part 2 treats the basics of neural networks. The parts can be studied in that order or mixed (which is how the course is taught). The appendix summarises several mostly known concepts from (linear) algebra. If any of the concepts from the appendix are unfamiliar to the reader, they are advised to read those carefully. In particular, if the reader is unfamiliar with convex sets and/or positive (semi)definite matrices, first read Appendix B and Definition A.2, respectively.

These lecture notes are still under construction. If you notice any typo's or mistakes or if you have any comments, please send them to r.p.hoeksma@utwente.nl.

1.1. Notation

Throughout these lecture notes, we denote vectors by boldface letters (e.g., $\mathbf{x}$), while real numbers are denoted by regular font letters (e.g., x). $\nabla f(\mathbf{x})$ denotes the gradient of the function f at the point $\mathbf{x}$ as a row vector, while $\nabla^2 f(\mathbf{x})$ denotes the Hessian of f at $\mathbf{x}$.

Part 1

Non-linear optimisation

From a very simplified and general point of view, optimisation is the field of mathematics that deals with finding minima and maxima of functions. Whoever reads this is likely familiar with optimisation of functions of few variables (most likely, single variable functions) and with a limited number of critical points (those points that are candidates to be an optimum). In this part, we learn about techniques for non-linear unconstrained optimisation. That is, these techniques allow for general functions, but are (mostly) limited to have an open domain.

Chapter 2 treats convex functions and some of their properties. These give a fundamental understanding of general functions around their optimal points and are an important subset of functions in their own right. Chapter 3 is an overview of different iterative methods for function optimisation, starting with, the quintessential, gradient descent method and Newton's method.

CHAPTER 2

Convex Functions

This chapter introduces the concept of convex functions. In the next chapter, we will see that convex functions are fundamental to understand performance of optimisation techniques near optimal points. To read this chapter, you should know about convex sets (see Appendix B).

With a function $f : \mathcal{F} \to \mathbb{R}$, defined on some subset $\mathcal{F} \subseteq \mathbb{R}^n$, one usually associates a "graphical representation" via its *graph* , *i.e.*, the set

$$\text{graph } f = \left\{ \begin{pmatrix} \mathbf{x} \\ f(\mathbf{x}) \end{pmatrix} \in \mathbb{R}^{n+1} \,\middle|\, \mathbf{x} \in \mathcal{F} \right\} .$$

A slight generalisation yields the concept of the *epigraph* of f, given by

$$\text{epi } f = \left\{ \begin{pmatrix} \mathbf{x} \\ z \end{pmatrix} \in \mathbb{R}^{n+1} \,\middle|\, \mathbf{x} \in \mathcal{F}, z \in \mathbb{R}, z \geq f(\mathbf{x}) \right\} .$$

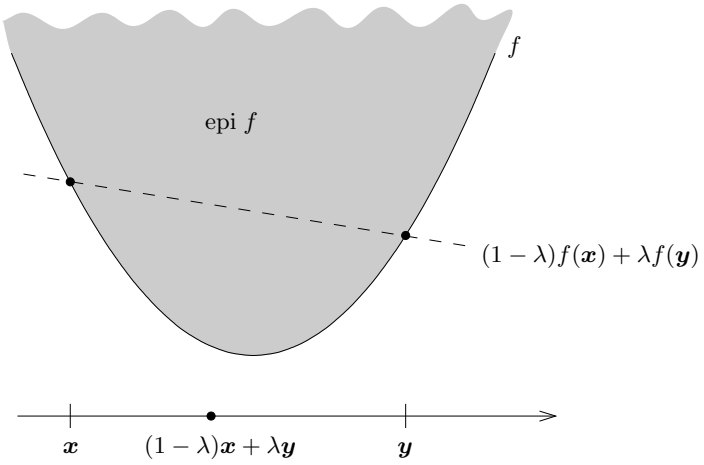


FIGURE 2.1. *Convex function with epigraph.*

Studying functions in terms of their epigraphs we are led to define a *convex function* to be a function with a convex epigraph (see Figure 2.1). Equivalently, we say that the function $f : \mathcal{F} \to \mathbb{R}$ is *convex* if for all $\mathbf{x}, \mathbf{y} \in \mathcal{F}$ and $0 \leq \lambda \leq 1$,

$$f[\mathbf{x} + \lambda(\mathbf{y} - \mathbf{x})] \leq f(\mathbf{x}) + \lambda(f(\mathbf{y}) - f(\mathbf{x})) ,$$

which we can also write as

$$f[(1-\lambda)\mathbf{x} + \lambda\mathbf{y}] \leq (1-\lambda)f(\mathbf{x}) + \lambda f(\mathbf{y}) .$$

The function f is called *strictly convex* if this inequality holds strictly for all $0 < \lambda < 1$. A function f is called *concave* if $(-f)$ is convex.

Observation 2.1. *The definition of convexity of f implies that each convex combination of points $\mathbf{x}, \mathbf{y}$ in the domain of f yields a point in the domain of f, i.e., $\mathcal{F} \subseteq \mathbb{R}^n$ is a convex set (see Appendix B.4).*

Convex (or concave) functions generalise, among others, linear and affine functions and thus form a particularly rich class of functions.

For an arbitrary set $C \subset \mathbb{R}^n$ the set

$$\operatorname{conv}(C) = \{\mathbf{x} = \sum_{i=1}^{k} \lambda_i \mathbf{x}_i \mid \mathbf{x}_i \in C, \lambda_i \geq 0, \sum_{i=1}^{k} \lambda_i = 1\}$$

is called the convex hull of C.

Before investigating the structure of general convex functions in further detail, we observe that a large part of the analysis can be reduced to the study of the one-dimensional case. To see this, let $f : \mathcal{F} \to \mathbb{R}$ be an arbitrary function that is defined on the convex set $\mathcal{F} \subseteq \mathbb{R}^n$. Then f is convex if and only if for all $\mathbf{x}, \mathbf{y} \in \mathcal{F}$, the restriction of f to the line segment $[\mathbf{x}, \mathbf{y}] \subseteq \mathcal{F}$ is a convex function. This fact is an immediate consequence of the definition of a convex function. We nevertheless state it explicitly because we will make repeated use of it later.

Lemma 2.2. *The real-valued function $f(\mathbf{x})$ is convex on the set $\mathcal{F}$ if and only if for every $\mathbf{x}_0 \in \mathcal{F}$ and $\mathbf{h} \in \mathbb{R}^n$*

$$p_{\mathbf{h}}(t) = f(\mathbf{x}_0 + t\mathbf{h})$$

is a convex function of t on the interval $I = \mathcal{F}_{\mathbf{h}}(\mathbf{x}_0) = \{t \in \mathbb{R} \mid \mathbf{x}_0 + t\mathbf{h} \in \mathcal{F}\}$.

Convex Optimisation. In linear programming, it is only a matter of taste whether one bases the optimisation model on "maximisation" or on "minimisation". Both models are equivalent. This is *not* the case in convex optimisation, where the objective functions f and $(-f)$ generally are not *both* convex. We point out some essential features already here.

As a generalisation of the optimality property of linear programs we note for *convex maximisation* problems:

Theorem 2.3. *Assume $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \mathbb{R}^n$ and $f : \operatorname{conv}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\} \to \mathbb{R}$ is convex. Then*

$$\max\{f(\mathbf{x}) \mid \mathbf{x} \in \operatorname{conv}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}\} = \max\{f(\mathbf{v}_i) \mid i = 1, \ldots, k\}\ .$$

Proof. If the scalars $\lambda_1, \ldots, \lambda_k \geq 0$ satisfy $\lambda_1 + \ldots + \lambda_k = 1$, we find

$$f(\lambda_1\mathbf{v}_1 + \ldots + \lambda_k\mathbf{v}_k) \ \leq\ \lambda_1 f(\mathbf{v}_1) + \ldots + \lambda_k f(\mathbf{v}_k) \ \leq\ \max_{1 \leq i \leq k} f(\mathbf{v}_i)\ .$$

□

As the one-dimensional example of $f(x) = x^2$ with $v_1 = -1$ and $v_2 = +1$ shows, the analogue of Theorem 2.3 is not true for *convex minimisation* problems. However,

one important property of linear programs is shared by all convex minimisation problems: a local minimiser is also a global minimiser. Generally, we call a point $\overline{\mathbf{x}} \in \mathcal{F}$ a *local minimiser* of f on $\mathcal{F} \subseteq \mathbb{R}^n$ if there is some $\varepsilon > 0$ are such that

$$f(\overline{\mathbf{x}}) \leq f(\mathbf{x}) \qquad \text{holds for all } \mathbf{x} \in \mathcal{F} \text{ with } \|\mathbf{x} - \bar{\mathbf{x}}\| < \varepsilon \ .$$

In case this is true for all $\varepsilon > 0$, $\overline{\mathbf{x}}$ is a *global minimiser*.

Theorem 2.4. *Let $f : \mathcal{F} \to \mathbb{R}$ be a convex function, $\mathcal{F} \subseteq \mathbb{R}^n$. Then the point $\bar{\mathbf{x}} \in \mathcal{F}$ is a global minimiser of f provided $\bar{\mathbf{x}}$ is a local minimiser.*

Proof. Let $\mathbf{w} \in \mathcal{F}$ and choose $\lambda > 0$ small enough for $\mathbf{x} = (1-\lambda)\bar{\mathbf{x}} + \lambda\mathbf{w}$ to satisfy $\|\mathbf{x} - \bar{\mathbf{x}}\| \leq \varepsilon$. Then the convexity of f yields

$$f(\bar{\mathbf{x}}) \leq f(\mathbf{x}) \leq (1-\lambda)f(\bar{\mathbf{x}}) + \lambda f(\mathbf{w})$$

and therefore $\lambda(f(\mathbf{w}) - f(\bar{\mathbf{x}})) \geq 0$, which implies $f(\mathbf{w}) \geq f(\bar{\mathbf{x}})$. □

As is common in non-linear optimisation, we base our standard model on *minimisation*. Convex maximisation problems thus belong to the model of *concave* minimisation.

Convex Functions Intro - Exercises.

Exercise 2.0.1.

(a) Show that every affine function $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} + b$, where $\mathbf{c} \in \mathbb{R}^n$ and $b \in \mathbb{R}$, is convex.
(b) Let $f_1, f_2 : \mathcal{F} \to \mathbb{R}$ be convex functions and $\gamma_1, \gamma_2 \geq 0$ scalars. Show that $f = \gamma_1 f_1 + \gamma_2 f_2$ yields a convex function.
(c) Let $g : \mathbb{R}^n \to I$, $I \subset \mathbb{R}$ be convex and $f : I \to \mathbb{R}$ be convex and non-decreasing. Show that the composition $f \circ g$ [1] of the functions f and g is convex.
(d) Show that $f(\mathbf{x}) = \|\mathbf{x}\|$ (any norm on $\mathbb{R}^n$) defines a convex function $f : \mathbb{R}^n \to \mathbb{R}$.
(e) Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Show that $g(\mathbf{x}) = f(\mathbf{x}_0 + \mathbf{x}) - f(\mathbf{x}_0)$ is convex for all $\mathbf{x}_0 \in \mathbb{R}^n$.

Exercise 2.0.2. Show that $f : C \to \mathbb{R}$ is convex if and only if for $\mathbf{x}_1, \ldots, \mathbf{x}_k \in C$ and scalars $\lambda_1, \ldots, \lambda_k \geq 0$ with $\lambda_1 + \ldots + \lambda_k = 1$ we have:

$$f(\lambda_1\mathbf{x}_1 + \ldots + \lambda_k\mathbf{x}_k) \leq \lambda_1 f(\mathbf{x}_1) + \ldots + \lambda_k f(\mathbf{x}_k) \ .$$

Exercise 2.0.3. Let $f : \mathcal{F} \to \mathbb{R}$ be convex. Show that the set of minimisers of f is a convex set. Moreover, if f is strictly convex, the minimiser (if there exists one) is unique.

Exercise 2.0.4. Given a set of convex functions F, prove that

$$g(\mathbf{x}) = \max_{f \in F} \{f(\mathbf{x})\}$$

is convex.

[1] $f \circ g(\mathbf{x}) = f(g(\mathbf{x}))$.

2.1. Convex Functions in One Variable

Motivated by Lemma 2.2, let us study convex functions in one variable. The convex subsets of $\mathbb{R}$ are the intervals (possibly including one or both of the endpoints). The key to our analysis of convex functions in one variable is the following technical but fundamental observation.

Lemma 2.5. *Let $f : (a, b) \to \mathbb{R}$ be a convex function on the open interval $(a, b) \subseteq \mathbb{R}$ and $x_0 \in (a, b)$. Define for all $t \in (a - x_0, 0) \cup (0, b - x_0)$,*

$$\varphi(t) = \frac{f(x_0 + t) - f(x_0)}{t} .$$

Then $\varphi(t)$ is monotonically increasing in t. Moreover, the following one-sided limits exist:

$$f'_-(x_0) := \lim_{t \uparrow 0} \frac{f(x_0 + t) - f(x_0)}{t} \leq \lim_{t \downarrow 0} \frac{f(x_0 + t) - f(x_0)}{t} =: f'_+(x_0) .$$

Proof. Assume $x_0 = 0$ and $f(x_0) = 0$ for ease of notation, *i.e.*, $\varphi(t) = f(t)/t$. (Otherwise, replace $f(x)$ by the convex function $g(x) = f(x + x_0) - f(x_0)$ on the interval $(a - x_0, b - x_0)$).

For $0 < \varepsilon \leq 1$, convexity of f now yields for every $t > 0$,

$$f(\varepsilon t) = f(0 + \varepsilon(t - 0)) \leq f(0) + \varepsilon(f(t) - f(0)) = \varepsilon f(t) .$$

Dividing by εt, we obtain $\varphi(\varepsilon t) \leq \varphi(t)$, which means that $\varphi(t)$ decreases monotonically as $t \downarrow 0$. Applying the same argument to $t < 0$, we see that $\varphi(t)$ increases monotonically as $t \uparrow 0$.

Since f is convex, the convex combination $0 = -t/2 + t/2$ implies for all $t > 0$,

$$0 = f(0) \leq \frac{1}{2} f(-t) + \frac{1}{2} f(t)$$

and hence $\varphi(-t) \leq \varphi(t)$. By the completeness property[2] of $\mathbb{R}$, both limits $f'_-(x_0) = \lim_{t \downarrow 0} \varphi(-t)$ and $f'_+(x_0) = \lim_{t \downarrow 0} \varphi(t)$ exist and satisfy the inequality $f'_-(x_0) \leq f'_+(x_0)$. □

Theorem 2.6. *Let $f : (a, b) \to \mathbb{R}$ be a convex function on the open interval (a, b). Then f is continuous.*

Proof. Consider any $x_0 \in (a, b)$. We claim $\lim_{x \to x_0} f(x) = f(x_0)$. Indeed, from Lemma 2.5 we learn that both $f'_-(x_0)$ and $f'_+(x_0)$ exist. Hence we must have in particular

$$\lim_{t \uparrow 0} f(x_0 + t) = f(x_0) = \lim_{t \downarrow 0} f(x_0 + t) .$$

□

The example of $f(x) = |x|$ with $f'_-(0) = -1$ and $f'_+(0) = 1$ shows that a convex function need not be differentiable and strict inequality $f'_-(x) < f'_+(x)$ may hold for some x in the domain of f.

[2] See any analysis text book.

On the other hand, if the convex function $f : (a,b) \to \mathbb{R}$ is differentiable at $x \in (a,b)$, then $f'(x) = f'_+(x) = f'_-(x)$ holds and Lemma 2.5 implies for every $y \in (a,b)$,

$$f(y) \geq f(x) + f'(x)(y-x)\ . \tag{2.1}$$

This observation motivates us to call a number $d_x \in \mathbb{R}$ a *subderivative* of f at x if it satisfies for all $y \in (a,b)$,

$$f(y) \geq f(x) + d_x(y-x)\ .$$

We denote by $\partial f(x)$ the collection of all subderivatives of f at x and say that the set $\partial f(x)$ is the *subdifferential* of f at x.

The existence of subderivatives is characteristic for convex functions on open domains.

Theorem 2.7. *The function $f : (a,b) \to \mathbb{R}$ is convex on the open interval (a,b) if and only if $\partial f(x) \neq \emptyset$ for all $x \in (a,b)$.*

Proof. If f is convex, then $\partial f(x) = [f'_-(x), f'_+(x)] \neq \emptyset$ (*cf.* Ex. 2.1.1). Conversely, assume that f has subderivatives.

To see that f is convex, let $x, y \in (a,b)$. Given any $0 \leq \lambda \leq 1$, set $z = x + \lambda(y-x)$ and choose a subderivative $d_z \in \partial f(z)$ in order to obtain the inequalities

$$\begin{aligned} f(x) &\geq f(z) + d_z(x-z) \\ f(y) &\geq f(z) + d_z(y-z). \end{aligned}$$

Multiplying the first by $1-\lambda$, the second by λ and adding both inequalities, we arrive at the desired inequality

$$f(x) + \lambda(f(y) - f(x)) \geq f(z) + d_z \cdot 0 = f[x + \lambda(y-x)]\ .$$

□

Differentiable Convex Functions. Let us assume that $f : (a,b) \to \mathbb{R}$ is differentiable. In this case we conclude from (2.1) and Theorem 2.7 that f is convex if and only if

$$f(y) \geq f(x) + f'(x)(y-x)\ . \tag{2.2}$$

holds for all $x, y \in (a,b)$.

The next result provides a characterisation of convexity for twice differentiable functions (see Corollary 2.9) that is often easier to check in practice.

Theorem 2.8. *The differentiable function f is convex on the open interval (a,b) if and only if the derivative $f'(x)$ is monotonically increasing on (a,b).*

Proof. Assume that f is convex and consider $a < x < y < b$. We claim that $f'(x) \leq f'(y)$ holds. Indeed, the convexity of f yields

$$f(y) \geq f(x) + f'(x)(y-x) \quad \text{and} \quad f(x) \geq f(y) + f'(y)(x-y)$$

which implies

$$f'(y)(y-x) \geq f(y) - f(x) \geq f'(x)(y-x) \ \text{ and hence } \ f'(y) \geq f'(x)\ .$$

Conversely, assume that f' increases monotonically and consider without loss of generality $a < x < y < b$ (the case $a < y < x < b$ is treated the same way). In order to exhibit the convexity of f, we need to show

$$f(y) \geq f(x) + f'(x)(y - x) \; .$$

Since f is differentiable on (a, b), the Mean Value Theorem[3] assures us of the existence of some number $x < \xi < y$ with the property

$$f(y) - f(x) = f'(\xi)(y - x) \; .$$

Because $f'(x) \leq f'(\xi)$ holds, we obtain $f(y) - f(x) \geq f'(x)(y - x)$. □

Corollary 2.9. *The twice differentiable function $f : (a, b) \to \mathbb{R}$ is convex on the open interval (a, b) if and only if $f''(x) \geq 0$ for all $x \in (a, b)$.*

Proof. Assume that f is convex. By Theorem 2.8, f' is monotonically increasing. So for $a < x < x + t < b$, we find $f'(x + t) - f'(x) \geq 0$, which implies

$$0 \leq \lim_{t \downarrow 0} \frac{f'(x + t) - f'(x)}{t} = f''(x) \; .$$

Conversely if $f'' \geq 0$ on (a, b) and $a < x < x+t < b$, then the Mean Value Theorem yields

$$f'(x + t) - f'(x) = f''(\xi)t \geq 0 \quad \text{for some } x \leq \xi \leq x + t \; .$$

So f' is monotonically increasing. □

Convex Functions in One Variable - Exercises.

Exercise 2.1.1.

(a) Let $f : (a, b) \to \mathbb{R}$ be convex. Show for all $x \in (a, b)$:

$$\partial f(x) = \{d \in \mathbb{R} \mid f'_-(x) \leq d \leq f'_+(x)\} \; .$$

(b) Determine the subdifferentials for the function $f(x) = |x^2 - 1|$ at the points $x_0 = 0$, $x_1 = 1$, and $x_2 = 4$.

Exercise 2.1.2. Show that $f(x) = e^x$ is convex on $\mathbb{R}$ and conclude that $1 + x \leq e^x$ holds for all $x \in \mathbb{R}$.

2.2. Convex Functions in n Variables

We now investigate convex functions $f : U \to \mathbb{R}$ that are defined on an open convex subset $U \subseteq \mathbb{R}^n$. Generalising the notion of a subderivative, we say that the vector $\mathbf{d} \in \mathbb{R}^n$ is a *subgradient* of f at the point $\mathbf{x} \in U$ if for all $\mathbf{y} \in U$,

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \mathbf{d}^T(\mathbf{y} - \mathbf{x}) \; . \tag{2.3}$$

By the *subdifferential* $\partial f(\mathbf{x})$ we understand the set of all subgradients of f at the point $\mathbf{x}$.

[3] See any analysis text book.

Remark. *The subdifferential may equally well be defined* locally *at* $\mathbf{x} \in U$*:*

Suppose $\mathbf{d} \in \mathbb{R}^n$ *is such that (2.3) holds for all* $\mathbf{y} \in U_\varepsilon(\mathbf{x})$ *for some* $\varepsilon > 0$*. In other words,* $\mathbf{x} \in U$ *is a local minimiser of the convex function* $f(\mathbf{y}) - \mathbf{d}^T\mathbf{y}$*. Hence* $\mathbf{x}$ *is a global minimiser (*cf. *Theorem 2.4),* i.e.*, (2.3) holds for all* $\mathbf{y} \in U$*. So the subdifferential* $\partial f(\mathbf{x})$ *of a convex function* f *does not depend on the open set* U *(containing* $\mathbf{x}$*).*

The notion of a subdifferential provides us with a quite straightforward general characterisation of (global) minima:

$$\mathbf{x} \in U \text{ solves } \min_{\mathbf{y}\in U} f(\mathbf{y}) \quad \Longleftrightarrow \quad \mathbf{0} \in \partial f(\mathbf{x}) \; . \tag{2.4}$$

In order to connect these general definitions with their one-dimensional analogue, note that for every $\mathbf{x} \in U$ and vector $\mathbf{h} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$, the parameter set

$$U_{\mathbf{h}} = \{t \in \mathbb{R} \,|\, \mathbf{x} + t\mathbf{h} \in U\}$$

forms an open interval in $\mathbb{R}$. Moreover, if $\mathbf{d} \in \partial f(\mathbf{x} + t_0\mathbf{h})$, the number $d = \mathbf{d}^T\mathbf{h}$ is a subderivative at $t = t_0$ for the one-dimensional function

$$p_{\mathbf{h}}(t) = f(\mathbf{x} + t\mathbf{h}) \; .$$

Indeed, $\mathbf{d} \in \partial f(\mathbf{x} + t_0\mathbf{h})$ implies for all $t \in U_{\mathbf{h}}$,

$$\begin{array}{rcl} p_{\mathbf{h}}(t) & = & f(\mathbf{x} + t_0\mathbf{h} + (t - t_0)\mathbf{h}) \;\geq\; f(\mathbf{x} + t_0\mathbf{h}) \;+\; \mathbf{d}^T(t - t_0)\mathbf{h} \\ & & \qquad\qquad = \quad p_{\mathbf{h}}(t_0) \;+\; d \cdot (t - t_0). \end{array}$$

Hence, if f admits a subgradient at every point of U, all the one-dimensional functions $p_{\mathbf{h}}(t)$ have non-empty subdifferentials and thus are convex (*cf.* Theorem 2.7). In view of Lemma 2.2, this argument shows

Proposition 2.10. *Let* $f : U \to \mathbb{R}$ *be such that* $\partial f(\mathbf{x}) \neq \emptyset$ *for all* $\mathbf{x} \in U$*. Then* f *is convex.*

Recall that in the one-dimensional situation subdifferentials are completely determined by the one-sided derivatives (*cf.* Ex. 2.1.1), which often permits the explicit computation of subderivatives. Unfortunately, no such procedure is known for the general n-dimensional case and the merit of Proposition 2.10 is to a large part theoretical. The situation is better in the case of differentiable functions, where the gradient turns out to be the unique candidate for a "subgradient".

Theorem 2.11. *The differentiable function* $f : U \to \mathbb{R}$ *is convex if and only if for all* $\mathbf{x}, \mathbf{y} \in U$*,*

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})(\mathbf{y} - \mathbf{x}) \; .$$

Proof. Since f is differentiable, $\nabla f(\mathbf{x})$ exists for every $\mathbf{x} \in U$. Assume that f satisfies the inequality, *i.e.*, each gradient of f is also a subgradient. Then Proposition 2.10 exhibits f to be convex.

Conversely, assume that there exist points $\mathbf{x}, \mathbf{y} \in U$ such that

$$f(\mathbf{y}) < f(\mathbf{x}) + \nabla f(\mathbf{x})(\mathbf{y} - \mathbf{x})$$

and consider the function $p_{\mathbf{h}}(t) = f(\mathbf{x} + t\mathbf{h})$, where $\mathbf{h} = \mathbf{y} - \mathbf{x}$. Then $p_{\mathbf{h}}$ is differentiable with $p'_{\mathbf{h}}(0) = \nabla f(\mathbf{x})\mathbf{h}$ and hence

$$p_{\mathbf{h}}(1) = f(\mathbf{y}) < f(\mathbf{x}) + \nabla f(\mathbf{x})\mathbf{h} = p_{\mathbf{h}}(0) + p'_{\mathbf{h}}(0)(1-0) \; .$$

By (2.1), $p_{\mathbf{h}}$ and hence also f cannot be convex. □

Corollary 2.12. *Let $f : U \to \mathbb{R}$ be twice differentiable. Then f is convex if and only if the Hessian matrix $\nabla^2 f(\mathbf{x})$ is positive semidefinite[4] for all $\mathbf{x} \in U$.*

Proof. By Lemma 2.2, f is convex if and only if for any $\mathbf{x}_0 \in U$ and direction $\mathbf{h} \in \mathbb{R}^n$ the function $p_{\mathbf{h}}(t) = f(\mathbf{x}_0 + t\mathbf{h})$ is convex. Since $p''_{\mathbf{h}}(t) = \mathbf{h}^T \nabla^2 f(\mathbf{x}_0 + t\mathbf{h})\mathbf{h}$, the claim follows from Corollary 2.9. □

Returning to general functions $f : U \to \mathbb{R}$, we want to establish the equivalent of Theorem 2.7 and show that the existence of subgradients is not only sufficient (Proposition 2.10) but also necessary for convexity. We will derive this result by proving the existence of supporting hyperplanes for the epigraph of f. First, we establish the analogue of Theorem 2.6.

Theorem 2.13. *Let $f : U \to \mathbb{R}$ be convex on the open set $U \subseteq \mathbb{R}^n$. Then f is continuous on U.*

Proof. We want to show the continuity of f at $\bar{\mathbf{x}} \in U$ and assume $f(\bar{\mathbf{x}}) = 0$ for simplicity. (Otherwise, consider the convex function $\bar{f}(\mathbf{x}) = f(\mathbf{x}) - f(\bar{\mathbf{x}})$.) We claim $f(\mathbf{y}) \to 0$ if $\mathbf{y} \to \bar{\mathbf{x}}$.

Because U is open, we can find some $\varepsilon > 0$ such that

$$U_\varepsilon(\bar{\mathbf{x}}) = \{\mathbf{y} \in \mathbb{R}^n \,|\, \|\mathbf{y} - \bar{\mathbf{x}}\| < \varepsilon\} \subseteq U \; .$$

If $\alpha > 0$ is now chosen so that $n\alpha^2 < \varepsilon^2$ is satisfied, then the polytope

$$P = \{\mathbf{y} \in \mathbb{R}^n \,|\, \bar{x}_j - \alpha \le y_j \le \bar{x}_j + \alpha, \; j = 1, \ldots, n\}$$

is contained in $U_\varepsilon(\overline{\mathbf{x}})$ and hence in U. From Theorem 2.3, we know that

$$M = \max_{\mathbf{y} \in P} f(\mathbf{y})$$

exists and is finite. Moreover, in view of $f(\bar{\mathbf{x}}) = 0$, Lemma 2.5 guarantees, for every $-\alpha \le t \le \alpha$ and every $\mathbf{u} \in \mathbb{R}^n$ with $\|\mathbf{u}\| = 1$, the monotone relation

$$\frac{-M}{\alpha} \le \frac{f(\bar{\mathbf{x}} - \alpha\mathbf{u})}{-\alpha} \le \frac{f(\bar{\mathbf{x}} + t\mathbf{u})}{t} \le \frac{f(\bar{\mathbf{x}} + \alpha\mathbf{u})}{\alpha} \le \frac{M}{\alpha} \; ,$$

which implies for $\mathbf{y} = \bar{\mathbf{x}} + t\mathbf{u}$,

$$|f(\mathbf{y})| \le \alpha^{-1} M |t| = \alpha^{-1} M \|\mathbf{y} - \bar{\mathbf{x}}\|$$

and hence $f(\mathbf{y}) \to 0$ as $\mathbf{y} \to \bar{\mathbf{x}}$. □

Theorem 2.14. *The function $f : U \to \mathbb{R}$ is convex on the open set $U \subseteq \mathbb{R}^n$ if and only if $\partial f(\mathbf{x}) \neq \emptyset$ for all $\mathbf{x} \in U$.*

[4] See Definition A.2

Proof. By Proposition 2.10, the condition is sufficient for the convexity of f. We will prove that f admits a subgradient at every $\mathbf{x} \in U$ if f is convex.

Let $\varepsilon > 0$ be such that the closed set $B = \{\mathbf{y} \in \mathbb{R}^n \,|\, \|\mathbf{y} - \mathbf{x}\| \leq \varepsilon\}$ is completely contained in U. Because f is continuous (Theorem 2.13), it is straightforward to verify that the epigraph of the restriction of f to B

$$\hat{B} = \{(\mathbf{y}, z) \,|\, \mathbf{y} \in B, z \geq f(\mathbf{y})\} \subseteq \text{epi } f$$

is closed (and, of course, also convex). Moreover, $(\mathbf{x}, f(\mathbf{x}))$ is a boundary point of $\hat{B}$. By Theorem B.5, there exists a vector $\mathbf{d} \in \mathbb{R}^n$ together with a number $d_{n+1} \in \mathbb{R}$, $(\mathbf{d}, d_{n+1}) \neq (\mathbf{0}, 0)$, such that for all $\mathbf{y} \in B$, and $z \geq f(\mathbf{y})$,

$$\mathbf{d}^T\mathbf{y} + d_{n+1}z \leq \mathbf{d}^T\mathbf{x} + d_{n+1}f(\mathbf{x}) \,.$$

Considering $z \to \infty$, we see that $d_{n+1} \leq 0$ must hold. Suppose $d_{n+1} = 0$ were true. Then we would have $\mathbf{d}^T(\mathbf{y} - \mathbf{x}) \leq \mathbf{0}$ for all $\mathbf{y} \in B$, which is impossible as $\mathbf{d} \neq \mathbf{0}$.

We thus know that $d_{n+1} < 0$ is valid, and we might as well assume $d_{n+1} = -1$ (otherwise we divide the inequality above by the number $|d_{n+1}| \neq 0$). So we obtain for all $\mathbf{y} \in B$ and the choice $z = f(\mathbf{y})$,

$$f(\mathbf{y}) - \mathbf{d}^T\mathbf{y} \;\geq\; f(\mathbf{x}) - \mathbf{d}^T\mathbf{x} \,.$$

Hence $\mathbf{d}$ is a subgradient of f in $\mathbf{x}$. □

Convex Functions in n Variables - Exercises.

Exercise 2.2.1. Let $f : U \to \mathbb{R}$ be differentiable and convex on the open set U. Show: $\partial f(\mathbf{x}) = \{\nabla f(\mathbf{x})^T\}$ for every $\mathbf{x} \in U$.

Exercise 2.2.2. Let $\mathbf{Q} \in \mathbb{R}^{n \times n}$ be a symmetric matrix. Show that $\mathbf{Q}$ is positive semidefinite $\iff f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x}$ is convex.

Exercise 2.2.3. Let $f : U \to \mathbb{R}$ be twice differentiable. Show: f is strictly convex if the Hessian matrix $\nabla^2 f(\mathbf{x})$ is positive definite for all $\mathbf{x} \in U$.

Exercise 2.2.4. Let $f : U \to \mathbb{R}$ be convex and $\mathbf{x} \in U$ (U open). Show: The subdifferential $\partial f(\mathbf{x})$ is a (non-empty) convex set.

CHAPTER 3

Iterative methods for unconstrained optimisation

In a very general setting, unconstrained optimisation deals with the problem of minimising a function $f : \mathcal{F} \to \mathbb{R}$ over an open set $\mathcal{F} \subseteq \mathbb{R}^n$:

$$\min_{\mathbf{x} \in \mathcal{F}} f(\mathbf{x}) \,. \tag{3.1}$$

A point $\overline{\mathbf{x}} \in \mathcal{F}$ is said to be a *global minimiser* of f over $\mathcal{F} \subseteq \mathbb{R}^n$ if

$$f(\overline{\mathbf{x}}) \leq f(\mathbf{x}) \qquad \text{for all } \mathbf{x} \in \mathcal{F} \,.$$

We call $\overline{\mathbf{x}} \in \mathcal{F}$ a *local minimiser* of f in $\mathcal{F}$ if there is an $\varepsilon > 0$ such that

$$f(\overline{\mathbf{x}}) \leq f(\mathbf{x}) \qquad \text{for all } \mathbf{x} \in \mathcal{F} \text{ satisfying } \|\mathbf{x} - \overline{\mathbf{x}}\| < \varepsilon \,.$$

One major difficulty in nonlinear optimisation arises from the fact that (3.1) typically has local minimisers that are not global ones. See Exercise 3.2.1 for an example of a function that has many non-global local minimisers.

Exhibiting global minimisers for high-dimensional problems of type (3.1) can be extremely difficult both in theory and practice. In nonlinear optimisation we therefore content ourselves usually with finding a local minimiser. There are a few exceptions from that general rule: For example, convex problems (see Theorem 2.4) or Lipschitz problems.

Generally speaking, unconstrained optimisation is a *local* theory. Since we assume $\mathcal{F}$ to be open, each point $\overline{\mathbf{x}} \in \mathcal{F}$ admits an $\varepsilon > 0$ such that a whole ε-neighbourhood of $\overline{\mathbf{x}}$ is contained in $\mathcal{F}$. Therefore, the constraint $\mathbf{x} \in \mathcal{F}$ is locally redundant ("not active"). Hence, there results no loss in generality when we concentrate on the case $\mathcal{F} = \mathbb{R}^n$. We emphasise that all local results of this chapter remain valid if we replace $\mathbb{R}^n$ by some open set $\mathcal{F} \subseteq \mathbb{R}^n$.

3.1. Convergence

Typically, an iterative method for solving (3.1) will generate a sequence of points $\mathbf{x}^{(k)}$ that (hopefully) converge to a local minimiser $\overline{\mathbf{x}}$ of f. We also are interested in the *rate of convergence* of the numerical method. Assume $\mathbf{x}^{(k)} \to \overline{\mathbf{x}}$ for $k \to \infty$. Then the sequence $(\mathbf{x}^{(k)})$ is said to be *linearly convergent* if some constant $0 \leq C < 1$ and some $K \in \mathbb{N}$ exist with the property

$$\|\mathbf{x}^{(k+1)} - \overline{\mathbf{x}}\| \leq C \|\mathbf{x}^{(k)} - \overline{\mathbf{x}}\| \qquad \text{for all } k \geq K.$$

The number C is called *convergence factor*. We speak of *superlinear convergence* if

$$\lim_{k\to\infty} \frac{\|\mathbf{x}^{(k+1)} - \overline{\mathbf{x}}\|}{\|\mathbf{x}^{(k)} - \overline{\mathbf{x}}\|} = 0 \, .$$

The sequence $(\mathbf{x}^{(k)})$ is *quadratically convergent* if a constant $C \geq 0$ exists so that

$$\|\mathbf{x}^{(k+1)} - \overline{\mathbf{x}}\| \leq C\|\mathbf{x}^{(k)} - \overline{\mathbf{x}}\|^2 \qquad \text{for all } k \in \mathbb{N}.$$

Convergence - Exercises.

Exercise 3.1.1. Let $\overline{\mathbf{x}}$ be a point and consider some process that starts with $\mathbf{x}^{(0)}$ and then generates $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$ iteratively so that

$$\|\mathbf{x}^{(k+1)} - \overline{\mathbf{x}}\| \leq C \, \|\mathbf{x}^{(k)} - \overline{\mathbf{x}}\|^2 \quad \text{for all } \; k \in \mathbb{N}.$$

Show that $\mathbf{x}^{(k)} \to \overline{\mathbf{x}}$ holds, provided $\mathbf{x}^{(0)}$ is chosen sufficiently close to $\overline{\mathbf{x}}$. Make "sufficiently close" explicit.

3.2. Optimality Conditions

In this section, we derive optimality conditions for a local minimiser of the unconstrained problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} \; f(\mathbf{x}) \, . \tag{3.2}$$

We assume that f is a C^1-function (or, even stronger, a C^2-function) on $\mathbb{R}^n$. So the gradient $\nabla f(\mathbf{x})$ of f exists and is a continuous function of $\mathbf{x}$.

A fundamental idea for solving this problem in theory and in practice is to consider so-called *descent directions* $\mathbf{d} \in \mathbb{R}^n$ for f in $\mathbf{x}^{(0)}$, satisfying (by definition)

$$\nabla f(\mathbf{x}^{(0)}) \, \mathbf{d} < 0 \, .$$

The differentiability of f implies in this case for all sufficiently small $\eta > 0$,

$$f(\mathbf{x}^{(0)} + \eta\mathbf{d}) = f(\mathbf{x}^{(0)}) + \eta\nabla f(\mathbf{x}^{(0)})\mathbf{d} + o(t) < f(\mathbf{x}^{(0)}) \tag{3.3}$$

i.e., the function f decreases strictly in the direction $\mathbf{d}$ at $\mathbf{x}^{(0)}$.

As a practical procedure, we might therefore try to minimise the function on the ray $\mathbf{x}^{(0)} + \eta\mathbf{d}$, $\eta \geq 0$, along the descent direction $\mathbf{d}$ and compute a solution $\eta^{(0)}$ for the one-dimensional problem

$$\min_{\eta\geq 0} \; f(\mathbf{x}^{(0)} + \eta\mathbf{d}) \, .$$

Setting $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \eta^{(0)}\mathbf{d}$, we then have the improvement $f(\mathbf{x}^{(1)}) < f(\mathbf{x}^{(0)})$. Such a step forms the basis of all so-called *descent methods* (*cf.* Section 3.4) .

Analysing f on a ray $\overline{\mathbf{x}} + \eta\mathbf{d}$, $\eta > 0$, we are led to optimality conditions for (3.2).

Lemma 3.1 (Necessary optimality conditions)**.** *Let f be a C^2-function on $\mathbb{R}^n$. Then each local minimiser $\overline{\mathbf{x}} \in \mathbb{R}^n$ of (3.2) satisfies:*

(a) (First order condition) $\quad \nabla f(\overline{\mathbf{x}}) = \mathbf{0}^T$

(b) (Second order condition) $\quad \mathbf{d}^T\nabla^2 f(\overline{\mathbf{x}})\mathbf{d} \geq 0 \quad$ for all $\mathbf{d} \in \mathbb{R}^n$.

Proof. (a) Suppose $\nabla f(\overline{\mathbf{x}}) \neq \mathbf{0}^T$. Then $\overline{\mathbf{x}}$ admits the descent direction $\mathbf{d} = -[\nabla f(\overline{\mathbf{x}})]^T$. So $\overline{\mathbf{x}}$ cannot be a local minimiser.

(b) Suppose $\mathbf{d} \in \mathbb{R}^n$ yields the inequality $\mathbf{d}^T \nabla^2 f(\overline{\mathbf{x}})\mathbf{d} < 0$. By (a), we may assume $\nabla f(\overline{\mathbf{x}}) = \mathbf{0}^T$. Taylor's formula (*cf.* p. 80) then implies

$$\begin{aligned} f(\overline{\mathbf{x}} + \eta\mathbf{d}) &= f(\overline{\mathbf{x}}) + \eta \nabla f(\overline{\mathbf{x}})\mathbf{d} + \tfrac{1}{2}\eta^2 \mathbf{d}^T \nabla^2 f(\overline{\mathbf{x}})\mathbf{d} + o(\eta^2) \\ &= f(\overline{\mathbf{x}}) + \tfrac{1}{2}\eta^2 \mathbf{d}^T \nabla^2 f(\overline{\mathbf{x}})\mathbf{d} + o(\eta^2) \\ &< f(\overline{\mathbf{x}}) \end{aligned}$$

for $\eta > 0$ small enough. This again contradicts the local minimality of $\overline{\mathbf{x}}$. □

In view of Lemma 3.1, a point $\overline{\mathbf{x}}$ satisfying the *critical equation* $\nabla f(\overline{\mathbf{x}}) = \mathbf{0}^T$ is often called a *critical point* of f. Yet, we must emphasise that the conditions (a) and (b) above are only necessary but generally not sufficient for local optimality (consider $f(x) = x^3$ at $\overline{x} = 0$, for instance). Moreover, as the example of the functions $f_1(x) = x^4$ and $f_2(x) = -f_1(x)$ at $\overline{x} = 0$ shows, (b) may not even be able to distinguish between a minimum and a maximum.

By strengthening condition (b) to positive definiteness of the Hessian matrix $\nabla^2 f(\overline{\mathbf{x}})$, however, we obtain a sufficient condition for local minimality. We say that $\overline{\mathbf{x}}$ is a *strict local minimiser* if there is an $\varepsilon > 0$ such that

$$f(\overline{\mathbf{x}}) < f(\mathbf{x}) \qquad \text{for all } \mathbf{x} \neq \overline{\mathbf{x}},\ \|\mathbf{x} - \overline{\mathbf{x}}\| < \varepsilon\ .$$

Lemma 3.2 (Sufficient optimality condition)**.** *Let f be a C^2-function on $\mathbb{R}^n$ and $\overline{\mathbf{x}} \in \mathbb{R}^n$ such that $\nabla f(\overline{\mathbf{x}}) = \mathbf{0}^T$ holds. Then $\overline{\mathbf{x}}$ is a strict local minimiser of f, provided $\overline{\mathbf{x}}$ satisfies*

$$\mathbf{d}^T \nabla^2 f(\overline{\mathbf{x}})\mathbf{d} > 0 \qquad \textit{for all } \mathbf{d} \in \mathbb{R}^n \setminus \{\mathbf{0}\}.$$

Proof. By assumption, $\nabla^2 f(\overline{\mathbf{x}}) \succ \mathbf{0}$. The continuity of $\mathbf{x} \to \nabla^2 f(\mathbf{x})$ therefore implies that for sufficiently small $\varepsilon > 0$, $\nabla^2 f(\mathbf{x}) \succ \mathbf{0}$ for all $\mathbf{x}$, $\|\mathbf{x} - \overline{\mathbf{x}}\| < \varepsilon$. Hence Taylor's formula yields (in view of $\nabla f(\overline{\mathbf{x}}) = \mathbf{0}^T$) for $\|\mathbf{d}\| = 1$ and $\mathbf{x} = \overline{\mathbf{x}} + \eta\mathbf{d}$ with $0 < |t| \leq \varepsilon$ (some $0 < \theta < 1$):

$$f(\mathbf{x}) - f(\overline{\mathbf{x}}) \ = \ \frac{1}{2}\eta^2 \mathbf{d}^T \nabla^2 f(\overline{\mathbf{x}} + \theta\eta\mathbf{d})\mathbf{d} > 0\ .$$

□

Based on these optimality conditions, the general algorithmic approach for computing a local minimiser of (3.2) is as follows:

- Find a critical point, *i.e.*, a point $\overline{\mathbf{x}}$ satisfying $\nabla f(\overline{\mathbf{x}}) = \mathbf{0}^T$.
- Check whether the candidate $\overline{\mathbf{x}}$ is indeed a local minimiser.

Remark. *The positive definiteness of the Hessian $\nabla^2 f(\overline{\mathbf{x}})$ can be verified efficiently by a variation of the Gaussian elimination algorithm.*

Global Minimisers of Convex Functions. The first- and second-order conditions of Lemma 3.1 are local conditions. As pointed out before, a local minimiser usually cannot be expected to be also a global one. For convex functions, however,

local and global minimisers coincide. More precisely, Theorem 2.4 says that every local minimiser of a convex function $f : \mathcal{F} \to \mathbb{R}$ necessarily is a global minimiser.

Moreover, recall from Chapter 2 that $\overline{\mathbf{x}} \in \mathcal{F}$ is a minimiser of f if and only if $\mathbf{0} \in \partial f(\overline{\mathbf{x}})$ (*cf.* (2.4)). So when f is differentiable, Ex. 2.2.1 implies

$$\overline{\mathbf{x}} \in \mathcal{F} \text{ is a (global) minimiser} \quad \Longleftrightarrow \quad \nabla f(\overline{\mathbf{x}}) = \mathbf{0}^T.$$

So for differentiable convex functions the necessary optimality condition $\nabla f(\overline{\mathbf{x}}) = \mathbf{0}^T$ is also sufficient. So, no second order information is needed. (Observe, however, that for convex C^2-functions the second order necessary optimality condition $\nabla^2 f(\overline{\mathbf{x}}) \succeq \mathbf{0}$ is automatically fulfilled (see Corollary. 2.12). In case of strict convexity, even the sufficient condition $\nabla^2 f(\overline{\mathbf{x}}) \succ \mathbf{0}$ is satisfied.

Optimality Conditions - Exercises.

Exercise 3.2.1. Show: The function $f : \mathbb{R} \to \mathbb{R}$, given by $f(x) = 2x^4 + x^4 \sin \frac{1}{x}$ for $x \neq 0$ and $f(0) = 0$, has the global minimiser $\overline{x} = 0$. Moreover, there are two sequences (x_k), (y_k) of local minimisers $x_k > 0$ and $y_k < 0$ that converge to $\overline{x}$.

Exercise 3.2.2. Find the critical points of the function $f(\mathbf{x}) = 2x_1^4 + 2x_1x_2 + 2x_1 + (1 + x_2)^2$ and determine all local minimisers.

Exercise 3.2.3. Assuming that the conditions of Lemma 3.2 hold, show that $\overline{\mathbf{x}}$ is an isolated local minimiser, *i.e.*, there exists a neighbourhood U of $\overline{\mathbf{x}}$ such that $\overline{\mathbf{x}}$ is the only local minimiser of f in U. (So an example as in Ex. 3.2.1 is impossible for C^2-functions.)

Exercise 3.2.4. Assume that $\mathbf{A}$ is a positive definite matrix. Show: $\overline{\mathbf{x}} = -\mathbf{A}^{-1}\mathbf{b}$ is the unique minimiser of the quadratic function $q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x}$.

Exercise 3.2.5. Show: Given $\mu > 0$ and $\mathbf{p} = (p_1, \ldots, p_n)^T \in \mathbb{R}^n$ with $\mathbf{p} > \mathbf{0}$, the function

$$f(\mathbf{x}) = \mathbf{p}^T\mathbf{x} - \mu \sum_{i=1}^{n} \ln x_i$$

is (strictly) convex on $\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x} > \mathbf{0}\}$ and the point $\overline{\mathbf{x}} = (\mu/p_1, \ldots, \mu/p_n)^T$ is the unique minimiser (*cf.* Ex. 2.2.3).

3.3. Descent Methods

We now investigate descent methods for finding local minimisers of the unconstrained problem

$$\min_{\mathbf{x} \in \mathbb{R}^n} \; f(\mathbf{x}) \, .$$

We again assume that f is a C^1- or even a C^2-function on $\mathbb{R}^n$. Moreover, for brevity of notation, we will set in the sequel

$$g(\mathbf{x}) = [\nabla f(\mathbf{x})]^T, \quad \mathbf{g}^{(k)} = g(\mathbf{x}^{(k)}).$$

Algorithm 1 Conceptual Descent Method

Init: Choose a starting point $\mathbf{x}^{(0)} \in \mathbb{R}^n$ and $\varepsilon > 0$.
Set $k = 0$
while $\|\mathbf{g}^{(k)}\| \geq \varepsilon$ **do**
  Choose a descent direction $\mathbf{d}^{(k)}$ at $\mathbf{x}^{(k)}$.
  Determine a solution $\eta^{(k)}$ for the problem:
  (*) $\min_{t \geq 0} f(\mathbf{x}^{(k)} + t\mathbf{d}^{(k)})$.
  Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \eta^{(k)}\mathbf{d}^{(k)}$.
  Set $k = k + 1$.

Assume that $\mathbf{x}^{(k)}$ admits the descent direction $\mathbf{d}^{(k)}$ (*i.e.*, $\nabla f(\mathbf{x}^{(k)})\mathbf{d}^{(k)} < 0$ holds). From Section 3.2 we then know $f(\mathbf{x}^{(k)} + \eta\mathbf{d}^{(k)}) < f(\mathbf{x}^{(k)})$ for all sufficiently small $\eta > 0$, which immediately suggests the following minimisation method:

This method reduces the minimisation of f in n variables x_i to the repeated so-called *line minimisation* problem (*) in one variable η. A sequence of points $\mathbf{x}^{(k)}$ is generated such that $f(\mathbf{x}^{(k+1)}) < f(\mathbf{x}^{(k)})$ is fulfilled and one may hope that the $\mathbf{x}^{(k)}$ converge to a local minimiser or at least to a critical point $\overline{\mathbf{x}}$ of f.

Let us look at the line-minimisation step (*) of the conceptual descent method in more detail. By Lemma 3.1 , a (local) minimiser $\eta^{(k)} > 0$ for (*) must satisfy

$$0 = \frac{d}{dt} f(\mathbf{x}^{(k)} + \eta\mathbf{d}^{(k)})|_{t=\eta^{(k)}} = \nabla f(\mathbf{x}_k + \eta^{(k)}\mathbf{d}^{(k)})\mathbf{d}^{(k)} \ .$$

We collect this fundamental fact into a lemma.

Lemma 3.3. *In the line-minimisation step (*) of the descent method, we have*

$$\nabla f(\mathbf{x}^{(k+1)})\mathbf{d}^{(k)} = (\mathbf{g}^{(k+1)})^T\mathbf{d}^{(k)} = 0 \ , \tag{3.4}$$

i.e., the subsequent gradient $\nabla f(\mathbf{x}^{(k+1)})$ is orthogonal to the search direction $\mathbf{d}^{(k)}$.

□

Descent Methods - Exercises.

Exercise 3.3.1. Applying the conceptual descent method to $q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}$ (with $\mathbf{A} \succ \mathbf{0}$), show that the unique minimiser of the line minimisation problem (*) is given by

$$t_k = -\frac{\mathbf{g}_k^T\mathbf{d}_k}{\mathbf{d}_k^T\mathbf{A}\mathbf{d}_k} \, .$$

3.4. Gradient Descent (Steepest Descent)

If $(\mathbf{g}^{(k)})^T = \nabla f(\mathbf{x}^{(k)}) \neq \mathbf{0}^T$, a natural choice for a search direction in (*) is the opposite direction of the gradient, the vector

$$\mathbf{d}^{(k)} = -\mathbf{g}^{(k)} = -g(\mathbf{x}^{(k)}) \ .$$

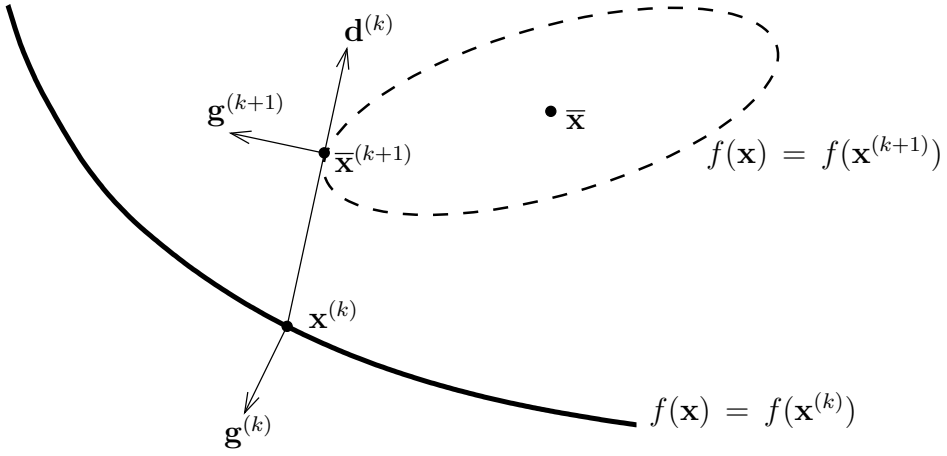


FIGURE 3.1. *Illustration of relation (3.4)*

We know[1] that this direction indicates the maximal marginal decrease for f at $\mathbf{x}^{(k)}$. Therefore, we call $\mathbf{d}^{(k)} = -\mathbf{g}^{(k)}$ the *steepest descent direction* and the corresponding conceptual descent method the *Gradient Descent Method* or *Steepest Descent Method*.

Algorithm 2 Gradient Descent Method

Init: Choose a starting point $\mathbf{x}^{(0)} \in \mathbb{R}^n$ and $\varepsilon > 0$.
Set $k = 0$
while $\|\mathbf{g}^{(k)}\| \geq \varepsilon$ **do**
  Set $\mathbf{d}^{(k)} = -\mathbf{g}^{(k)}$.
  Determine a solution $\eta^{(k)}$ for the problem:
  (*) $\qquad \min_{t \geq 0} f(\mathbf{x}^{(k)} + t\mathbf{d}^{(k)})\,.$
  Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \eta^{(k)}\mathbf{d}^{(k)}$.
  Set $k = k + 1$.

According to Lemma 3.3, the Gradient Descent Method has the property

$$(\mathbf{d}^{(k+1)})^T\mathbf{d}^{(k)} = (\mathbf{g}^{(k+1)})^T\mathbf{d}^{(k)} = 0\,,$$

i.e., successive descent directions are orthogonal. This phenomenon is known as the so-called *zigzagging* of the steepest descent method, which often results in slow convergence (see Ex. 3.4.3).

It turns out that the steepest descent method converges to a critical point – provided it converges at all.

THEOREM 3.4. *Let f be a C^1-function on $\mathbb{R}^n$ and consider the steepest descent method (i.e. $\mathbf{d}^{(k)} = -\mathbf{g}^{(k)}$). Then*

$$\mathbf{x}^{(k)} \to \overline{\mathbf{x}} \quad \text{implies} \quad \nabla f(\overline{\mathbf{x}}) = \mathbf{0}^T\,.$$

[1]See, e.g., Section A.2.3 in the appendix.

Proof. Because f has continuous partial derivatives, $\mathbf{x}^{(k)} \to \overline{\mathbf{x}}$ entails

$$\mathbf{g}^{(k)} \to g(\overline{\mathbf{x}}) \,, \qquad \text{hence} \qquad (\mathbf{g}^{(k+1)} - \mathbf{g}^{(k)}) \to \mathbf{0} \,.$$

By Lemma 3.3 we have $(\mathbf{g}^{(k+1)})^T \mathbf{g}^{(k)} = 0$. Hence, $0 \leq (\mathbf{g}^{(k)})^T \mathbf{g}^{(k)} = (\mathbf{g}^{(k)})^T (\mathbf{g}^{(k)} - \mathbf{g}^{(k+1)}) \to 0$, which implies $\mathbf{g}^{(k)} \to \mathbf{0}$ and therefore $g(\overline{\mathbf{x}}) = \mathbf{0}$. □

3.4.1. Convergence Properties. Generally, it is difficult to prove general convergence properties of the iterative methods that are used in practice. The Gradient Descent Method is no exception. However, it is one of the methods where we can prove convergence if our function f satisfies certain conditions. One such condition is the important case of f being a (convex) quadratic function of the form

$$q(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x}$$

where $\mathbf{A}$ is a positive definite (symmetric) matrix. Note that q obviously has the unique minimiser $\overline{\mathbf{x}} = \mathbf{0}$.

The motivation is the following. Quadratic functions offer not only the simplest non-linear examples where all algorithmic steps can be calculated explicitly. Computing the Taylor expansion of an arbitrary C^2-function f at a local minimiser $\overline{\mathbf{x}}$, we furthermore obtain (in view of $\nabla f(\overline{\mathbf{x}}) = \mathbf{0}$)

$$f(\overline{\mathbf{x}} + \mathbf{d}) = f(\overline{\mathbf{x}}) + \frac{1}{2} \mathbf{d}^T \nabla^2 f(\overline{\mathbf{x}}) \mathbf{d} + o\big(\|\mathbf{d}\|^2\big) \approx f(\overline{\mathbf{x}}) + \frac{1}{2} \mathbf{d}^T \nabla^2 f(\overline{\mathbf{x}}) \mathbf{d} \,,$$

which suggests that the behaviour of f near $\overline{\mathbf{x}}$ should be essentially captured by the behaviour of the quadratic function

$$q(\mathbf{d}) = \frac{1}{2} \mathbf{d}^T \nabla^2 f(\overline{\mathbf{x}}) \mathbf{d} \,.$$

Recall that $\langle \mathbf{x} | \mathbf{y} \rangle = (\mathbf{y}^T \mathbf{A} \mathbf{x})/2$ defines an inner product (*cf.* Section A.1). Geometrically speaking, the associated norm $\|\mathbf{x}\|_{\mathbf{A}} = \sqrt{(\mathbf{x}^T \mathbf{A} \mathbf{x})/2} = \sqrt{q(\mathbf{x})}$ is a distance measure for $\mathbf{x}$ relative to the minimiser $\overline{\mathbf{x}} = \mathbf{0}$ of $q(\mathbf{x})$. The next theorem describes the convergence of the steepest descent method in terms of this distance.

Theorem 3.5. *Let* $\mathbf{A}$ *be positive definite and* $q(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x}$. *Then the steepest descent method generates a sequence* $(\mathbf{x}^{(k)})$ *satisfying*

$$\sqrt{q(\mathbf{x}^{(k+1)})} \leq \left(\frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}} \right) \sqrt{q(\mathbf{x}^{(k)})} \,, \tag{3.5}$$

where $\lambda_{min} > 0$ *is the smallest and* λ_{max} *the largest eigenvalue of* $\mathbf{A}$.

Proof. In this case, the unique minimiser of the line minimisation problem (*) is given by

$$\eta^{(k)} = - \frac{(\mathbf{g}^{(k)})^T \mathbf{d}^{(k)}}{(\mathbf{d}^{(k)})^T \mathbf{A} \mathbf{d}^{(k)}} \,.$$

Therefore, the steepest descent method generates the iterates

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \eta^{(k)} \mathbf{g}^{(k)} \quad \text{with} \quad \eta^{(k)} = \frac{(\mathbf{g}^{(k)})^T \mathbf{g}^{(k)}}{(\mathbf{g}^{(k)})^T \mathbf{A} \mathbf{g}^{(k)}} \,.$$

Since $\mathbf{g}^{(k)} = \mathbf{A}\mathbf{x}^{(k)}$, we obtain

$$\begin{aligned} q(\mathbf{x}^{(k+1)}) &= \frac{1}{2}(\mathbf{x}^{(k)} - \eta^{(k)}\mathbf{g}^{(k)})^T\mathbf{A}(\mathbf{x}^{(k)} - \eta^{(k)}\mathbf{g}^{(k)}) \\ &= q(\mathbf{x}^{(k)}) - \eta^{(k)}(\mathbf{g}^{(k)})^T\mathbf{A}\mathbf{x}^{(k)} + \frac{1}{2}(\eta^{(k)})^2(\mathbf{g}^{(k)})^T\mathbf{A}\mathbf{g}^{(k)} \\ &= q(\mathbf{x}^{(k)}) - \eta^{(k)}(\mathbf{g}^{(k)})^T\mathbf{g}^{(k)} + \frac{1}{2}(\eta^{(k)})^2(\mathbf{g}^{(k)})^T\mathbf{A}\mathbf{g}^{(k)} \\ &= q(\mathbf{x}^{(k)}) - \frac{1}{2}\frac{((\mathbf{g}^{(k)})^T\mathbf{g}^{(k)})^2}{(\mathbf{g}^{(k)})^T\mathbf{A}\mathbf{g}^{(k)}}. \end{aligned}$$

On the other hand $\mathbf{g}^{(k)} = \mathbf{A}\mathbf{x}^{(k)}$ implies

$$q(\mathbf{x}^{(k)}) = \frac{1}{2}(\mathbf{x}^{(k)})^T\mathbf{A}\mathbf{A}^{-1}\mathbf{A}\mathbf{x}^{(k)} = \frac{1}{2}(\mathbf{g}^{(k)})^T\mathbf{A}^{-1}\mathbf{g}^{(k)}\,.$$

From the Kantorovich Inequality[2], we therefore conclude

$$\frac{q(\mathbf{x}^{(k)}) - q(\mathbf{x}^{(k+1)})}{q(\mathbf{x}^{(k)})} \;=\; \frac{((\mathbf{g}^{(k)})^T\mathbf{g}^{(k)})^2}{((\mathbf{g}^{(k)})^T\mathbf{A}\mathbf{g}^{(k)})((\mathbf{g}^{(k)})^T\mathbf{A}^{-1}\mathbf{g}^{(k)})} \;\geq\; \frac{4\lambda_{min}\lambda_{max}}{(\lambda_{min}+\lambda_{max})^2}$$

and hence

$$\frac{q(\mathbf{x}^{(k+1)})}{q(\mathbf{x}^{(k)})} \;\leq\; \left(1 - \frac{4\lambda_{min}\lambda_{max}}{(\lambda_{min}+\lambda_{max})^2}\right) = \left(\frac{\lambda_{max}-\lambda_{min}}{\lambda_{max}+\lambda_{min}}\right)^2 .$$

□

Theorem 3.5 guarantees fast convergence if the ratio $\kappa = \lambda_{max}/\lambda_{min}$ for $\mathbf{A}$ is small (*i.e.*, close to 1) and suggests slow convergence if κ is large (see Ex. 3.4.3 for an illustration).

Gradient Descent - Exercises.

Exercise 3.4.1. Let $f : \mathbb{R}^2 \to \mathbb{R}$ be defined by $f(x, y) = x^2 \sin^2 y + x$. Starting from $\mathbf{x}^{(0)} = (1, \frac{\pi}{2})$ apply the gradient descent method. Find the descent direction and $\mathbf{x}^{(1)}$ (by explicitly solving (*)).

Exercise 3.4.2. Let $f : \mathbb{R}^2 \to \mathbb{R}$ be defined by $f(x, y) = x^2 + y^2 + 2x$. Starting from $(x^{(0)}, y^{(0)}) = (-2, 1)$ apply the gradient descent method. Find the descent direction and $(x^{(1)}, y^{(1)})$ (by explicitly solving (*)). Is $(x^{(1)}, y^{(1)})$ a local minimiser?

Exercise 3.4.3. Let $q(x_1, x_2) = (x_1, x_2)\begin{pmatrix} 1 & 0 \\ 0 & r \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = x_1^2 + rx_2^2$ where $r > 1$. Starting steepest descent with respect to q at the point $(x_1^{(0)}, x_2^{(0)}) = (r, 1)$, show for all $k \geq 1$:

$$(x_1^{(k)}, x_2^{(k)}) = \left(\frac{r-1}{r+1}\right)^k (r, (-1)^k)\,.$$

So if $r > 1$ is large , convergence to the limit point $(0, 0)$ is slow.

[2]Kantorovich Inequality is out of the scope of this course, but the version we need here states that $1 \leq (\mathbf{x}^T\mathbf{A}\mathbf{x})(\mathbf{x}^T\mathbf{A}^{-1}\mathbf{x}) \leq \frac{(\lambda_1+\lambda_n)^2}{4\lambda_1\lambda_n}$, for all positive definite $\mathbf{A} \in \mathbb{S}^{n\times n}$ with eigenvalues $0 < \lambda_1 \leq \ldots \leq \lambda_n$, and $\mathbf{x} \in \mathbb{R}^n$ with $\|\mathbf{x}\| = 1$.

3.5. Line Search

Each iteration in the conceptual descent method described by Algorithm 1 requires the solution of a one-dimensional minimisation problem of the type

$$\min_{t\geq 0} h(t) \qquad \text{with} \quad h(t) = f(\mathbf{x}^{(k)} + t\mathbf{d}^{(k)}) \,. \tag{3.6}$$

Since $\mathbf{d}^{(k)}$ is a descent direction we have $h'(0) = \nabla f(\mathbf{x}^{(k)})\mathbf{d}^{(k)} < 0$. As usual, we cannot hope to achieve a global minimum of h and therefore concentrate on the determination of a local minimiser $\tilde{t}$ satisfying $h(\tilde{t}) < h(0)$. There are several strategies that pursue this goal.

3.5.1. Exact Line Search. We illustrate the idea by discussing a simple version. The procedure rests on the following observation.

Given a continuous function $h : \mathbb{R} \to \mathbb{R}$, and two numbers $a < b$, the interval (a, b) contains a local minimiser of h whenever there exists some $t \in (a, b)$ such that

$$h(a) \geq h(t) \quad \text{and} \quad h(b) \geq h(t) \,. \tag{3.7}$$

We start by fixing a "search horizon" $(0, T)$ and try to determine $a < t < b$ with property (3.7). This can be done by *discrete search* as follows.

We choose numbers $0 = t_0 < t_1 < \ldots < t_{s-1} < t_s = T$ and compute

$$h(t_k) = \min_{0\leq i\leq s} h(t_i) \,.$$

If $1 \leq k \leq s-1$ holds, then clearly $a = t_{k-1} < t_k < t_{k+1} = b$ is suited. If the subdivision of the horizon $(0, T)$ is sufficiently fine, then $h'(0) < 0$ will imply $h(t_k) \leq h(t_1) < h(0)$.

Bisection. Assume we have a triple $a < t := (b+a)/2 < b$ so that (3.7) holds. Consider the points $t_1 = (t + a)/2$ and $t_2 = (b + t)/2$.

If $h(t_1) \leq h(t)$, we can replace $a < t < b$ by $a < t_1 < t$ and retain property (3.7). Similarly, if $h(t_2) \leq h(t)$, we can step over to $t < t_2 < b$. If $h(t_1) \geq h(t)$ and $h(t_2) \geq h(t)$, we pass to $t_1 < t < t_2$ (see Figure 3.2).

In any case we thus find a new triple $a' < t' < b'$ that satisfies (3.7) and reduces the interval length $b - a$ to half of the previous length. We repeat the bisection procedure until we have found a small interval (a, b) determining a local minimiser $t \in (a, b)$ with accuracy $b - a < \varepsilon$.

FIGURE 3.2. *Bisection and Golden Section*

Golden Section. The *golden ratio* is defined as the number $\tau > 0$ with the property

$$\tau^2 = 1 - \tau \ , \qquad i.e., \ \tau = \frac{\sqrt{5}-1}{2} \approx 0.618 \ .$$

We say that $a < t < b$ is a *golden section* of (a, b) if

$$\text{either} \qquad \frac{t-a}{b-a} = \tau \qquad \text{or} \qquad \frac{t-a}{b-a} = 1 - \tau \ .$$

The well-known key property of golden sections is stated in Ex. 3.5.2.

Suppose now that $a < t < b$ is a golden section and satisfies (3.7) (with $t - a = \tau(b-a)$). Let $t' = b - (t-a)$ (*cf.* Figure 3.2). If $h(t') \le h(t)$, then $a < t' < t$ is a golden section satisfying (3.7). If $h(t') \ge h(t)$, then $t' < t < b$ is a golden section with property (3.7).

So we can reduce the search interval in a similar spirit as with the bisection method. Each iteration will reduce the interval to about 0.618 times its former length. Note, however, that the golden section method requires only 1 additional function evaluation per iteration!

3.5.2. Inexact Line Search. Efficient implementations of descent algorithms in higher dimensions should not spend too much computation time on subproblem (3.6) when the current iteration point $\mathbf{x}^{(k)}$ is still far from a critical point $\overline{\mathbf{x}}$ (*i.e.* $\|\nabla f(\mathbf{x}^{(k)})\|$ is still large). One might therefore often prefer *inexact line search* methods that automatically increase the accuracy of the line minimisation routine when $\mathbf{x}^{(k)}$ approaches a critical point, *i.e.*, when $\|\nabla f(\mathbf{x}^{(k)})\|$ (and thus $|h'(0)|$) becomes small. The *Goldstein test* is such a method.

Goldstein Test. Assume that h decreases locally at $t = 0$, *i.e.*, $h'(0) < 0$. We choose a slope parameter $0 < \mu < 0.5$, and consider the lines (see Figure 3.3)

$$\ell_\mu(t) = h(0) + \mu h'(0)\ t \quad \text{and} \quad \ell_{1-\mu}(t) = h(0) + (1-\mu)h'(0)\ t \ .$$

Suppose we have exhibited an approximation $\tilde{t}$ of a minimiser of h (say, by a rough discrete search as explained in the previous subsection). Intuitively, we might find $\tilde{t}$ to be not acceptable if

$$h(\tilde{t}) \ge \ell_\mu(\tilde{t}) \quad \text{or} \quad h(\tilde{t}) \le \ell_{1-\mu}(\tilde{t}) \ .$$

Hence we arrive at the following rule (*cf.* Figure 3.3): *Accept* $\tilde{t} > 0$ *as an approximate minimiser* if

$$\ell_{1-\mu}(\tilde{t}) \ < \ h(\tilde{t}) \ < \ \ell_\mu(\tilde{t}) \ . \tag{3.8}$$

For example, the quadratic function $h(t) = at + qt^2,\ q > 0, a < 0$, has the minimiser $\bar{t} = -a/(2q)$. The points that would be accepted by (3.8) form the interval (see Ex. 3.5.5)

$$(-a\mu/q \ , \ -a(1-\mu)/q) \ .$$

If $|h'(0)| = |a|$ is small, also this interval is small and the minimiser $\bar{t}$ is automatically computed with high accuracy.

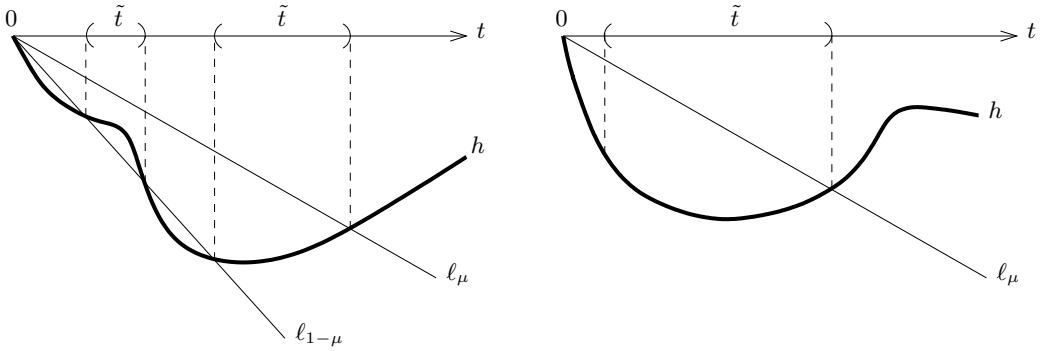


FIGURE 3.3. *Goldstein test (left) and the Goldstein-Wolfe test (right)*

Goldstein-Wolfe Test. For non-quadratic h, the condition (3.8) may exclude the (local) minimisers of h. Therefore, the Goldstein test is often replaced by the following modified rule (*cf.* Figure 3.3):

Choose parameters $0 < \mu < 0.5$ *and* $\mu < \sigma < 1$. *Accept the point* $\tilde{t} > 0$ *if*

$$|h'(\tilde{t})| \le \sigma |h'(0)| \quad \text{and} \quad h(\tilde{t}) < \ell_\mu(\tilde{t}). \tag{3.9}$$

REMARK. *Acceptable points may not always exist for the Goldstein-Wolfe test. (Take, for example, h as a linear function $h(t) = h(0) + h'(0)t$.) However, if the graph of h intersects the line $\ell_\mu(t) = h(0) + \mu h'(0)t$, one can show that acceptable points are guaranteed (cf. Ex. 3.5.6).*

Line Search - Exercises.

Exercise 3.5.1. Verify the correctness of the rule (3.7).

Exercise 3.5.2. Show: If $a < t < b$ is a golden section (satisfying $t - a = \tau(b-a)$) and $t' = b - (t-a)$, then both $a < t' < t$ and $t' < t < b$ are golden sections.

Exercise 3.5.3. Show: The reduction factor per function evaluation of the bisection method is (in the worst case) $\sqrt{2}/2 \approx 0.707$.

Exercise 3.5.4. Given $f(t) = ...$

(a) Apply the bisection line search method on the interval $[0, 1]$.
(b) Apply the golden section line search method on the interval $[0, 1]$.
(c) Apply the Goldstein test on all points in the interval $(0, 1]$ discretised in steps of 0.05.
(d) Apply the Goldstein-Wolfe test on all points in the interval $(0, 1]$ discretised in steps of 0.05.

Exercise 3.5.5. Let $h(t) = at + qt^2,\ q > 0, a < 0$. Show: The relation (3.8) defines the interval $(-a\mu/q\ ,\ -a(1-\mu)/q)$, which contains the (global) minimiser of h.

Exercise 3.5.6. Assume that μ, σ are chosen according to the Goldstein-Wolfe test and that the graph of ℓ_μ intersects the graph of h (see Figure 3.3). Let $\hat{t}$ be the smallest positive intersection point. Then $(0, \hat{t})$ contains some non-empty interval I such that (3.9) holds for all $\tilde{t} \in I$.

3.6. Newton's Method

From Lemma 3.1, we know that all local minimisers of a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ satisfy the critical equation $\nabla f(\mathbf{x}) = \mathbf{0}^T$. This equation is nonlinear in $\mathbf{x}$ if f is a non-quadratic function. In this section, we present Newton's method for (approximately) solving nonlinear critical equations. You may have encountered Newton's Method for single-variable functions before. In this section, we only treat multi-variable (both as domain and range of the function f) Newton's Method but, of course, this is a generalisation of the single-variable version.

Consider the differentiable function $F : \mathbb{R}^n \to \mathbb{R}^n$. We are looking for a solution $\overline{\mathbf{x}} \in \mathbb{R}^n$ of the equation $F(\mathbf{x}) = \mathbf{0}$. That is, any Newton's basic approach determines iteratively vectors $\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(k)}, \ldots$ as follows. If $F(\mathbf{x}^{(k)}) \neq \mathbf{0}$, we approximate F linearly *via* its linear Taylor approximation

$$F(\mathbf{x}) \approx F(\mathbf{x}^{(k)}) + \nabla F(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)})$$

and choose $\mathbf{x}^{(k+1)}$ as a solution of the system of linear equations

$$F(\mathbf{x}^{(k)}) + \nabla F(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)}) = \mathbf{0} \quad \text{or} \quad \nabla F(\mathbf{x}^{(k)})\mathbf{x} = \nabla F(\mathbf{x}^{(k)})\mathbf{x}^{(k)} - F(\mathbf{x}^{(k)}) \, .$$

We call the step from $\mathbf{x}^{(k)}$ to $\mathbf{x}^{(k+1)}$ a *Newton step*. Assuming $\nabla F(\mathbf{x}^{(k)})$ to be invertible, this step becomes the *Newton iteration*

$$\mathbf{x}^{(k+1)} \; = \; \mathbf{x}^{(k)} - [\nabla F(\mathbf{x}^{(k)})]^{-1} F(\mathbf{x}^{(k)}) \tag{3.10}$$

Remark. *If $F : \mathbb{R}^n \to \mathbb{R}^n$ is affine linear, Newton's method just solves the linear system $F(\mathbf{x}) = \mathbf{0}$ and hence converges trivially in one iteration to the desired solution.*

Convergence. In general, one can guarantee (quadratic) convergence to a solution $\overline{\mathbf{x}}$ of $F(\mathbf{x}) = \mathbf{0}$ if F is "sufficiently smooth" in a neighbourhood of $\overline{\mathbf{x}}$. We make this statement precise in the following theorem.

Theorem 3.6 (Convergence of Newton's method)**.** *Let $F : \mathbb{R}^n \to \mathbb{R}^n$ be a C^2-function on $\mathbb{R}^n$ and $\overline{\mathbf{x}} \in \mathbb{R}^n$ a "zero" of F, i.e, $F(\overline{\mathbf{x}}) = \mathbf{0}$, so that $\det \nabla F(\overline{\mathbf{x}}) \neq 0$ holds. Then the Newton iteration (3.10) converges quadratically to $\overline{\mathbf{x}}$, provided the starting point $\mathbf{x}^{(0)}$ is chosen sufficiently close to $\overline{\mathbf{x}}$.*

Proof. Consider any component function $f_i : \mathbb{R}^n \to \mathbb{R}$ of $F = (f_1, \ldots, f_n)$. Since f_i is twice continuously differentiable and $f_i(\overline{\mathbf{x}}) = 0$, we know from Taylor's formula

$$f_i(\mathbf{x}) = \nabla f_i(\overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}}) + \frac{1}{2}(\mathbf{x} - \overline{\mathbf{x}})^T \nabla^2 f_i(\overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}}) + o(\|\mathbf{x} - \overline{\mathbf{x}}\|^2) \, .$$

Moreover, the differentiability of the gradient $\nabla f_i(\mathbf{x})$ yields

$$\nabla f_i(\mathbf{x}) = \nabla f_i(\overline{\mathbf{x}}) + (\mathbf{x} - \overline{\mathbf{x}})^T \nabla^2 f_i(\overline{\mathbf{x}}) + o(\|\mathbf{x} - \overline{\mathbf{x}}\|)$$

and thus

$$\nabla f_i(\mathbf{x})(\mathbf{x} - \overline{\mathbf{x}}) = \nabla f_i(\overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}}) + (\mathbf{x} - \overline{\mathbf{x}})^T \nabla^2 f_i(\overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}}) + o(\|\mathbf{x} - \overline{\mathbf{x}}\|^2) \, .$$

The two relations together yield

$$\nabla f_i(\mathbf{x})(\mathbf{x} - \overline{\mathbf{x}}) - f_i(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \overline{\mathbf{x}})^T \nabla^2 f_i(\overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}}) + o(\|\mathbf{x} - \overline{\mathbf{x}}\|^2)$$

and therefore, with $\beta_i = 1 + \max_{\|\mathbf{d}\|=1} |\mathbf{d}^T \nabla^2 f_i(\overline{\mathbf{x}})\mathbf{d}|$,

$$|\nabla f_i(\mathbf{x})(\mathbf{x} - \overline{\mathbf{x}}) - f_i(\mathbf{x})| \leq \beta_i \|\mathbf{x} - \overline{\mathbf{x}}\|^2$$

for sufficiently small $\|\mathbf{x} - \overline{\mathbf{x}}\|$. Taking all component functions of F into account, we thus find that there exist constants β and $\eta > 0$ such that

$$\|\nabla F(\mathbf{x})(\mathbf{x} - \overline{\mathbf{x}}) - F(\mathbf{x})\| \leq \beta \|\mathbf{x} - \overline{\mathbf{x}}\|^2 \tag{3.11}$$

holds whenever $\|\mathbf{x} - \overline{\mathbf{x}}\| \leq \eta$.

Note next that the continuity of $\nabla F(\mathbf{x})$ implies that $\mathbf{x} \to \det \nabla F(\mathbf{x})$ is continuous as well. So there exists some $\delta > 0$ such that $\det \nabla F(\mathbf{x}) \neq 0$ whenever $\|\mathbf{x} - \overline{\mathbf{x}}\| < \delta$. In the latter case, also the inverse $H(\mathbf{x}) = [\nabla F(\mathbf{x})]^{-1}$ exists. Moreover, letting $\|H(\mathbf{x})\|$ denote the Frobenius norm of $H(\mathbf{x})$ and recalling (3.11), we find for the Newton iteration:

$$\begin{aligned} \|\mathbf{x}^{(k+1)} - \overline{\mathbf{x}}\| &= \|\mathbf{x}^{(k)} - \overline{\mathbf{x}} - [\nabla F(\mathbf{x}^{(k)})]^{-1} F(\mathbf{x}^{(k)})\| \\ &= \|H(\mathbf{x}^{(k)})[\nabla F(\mathbf{x}^{(k)})(\mathbf{x}^{(k)} - \overline{\mathbf{x}}) - F(\mathbf{x}^{(k)})]\ \| \\ &\leq \|H(\mathbf{x}^{(k)})\| \cdot \beta \cdot \|\mathbf{x}^{(k)} - \overline{\mathbf{x}}\|^2 \ . \end{aligned}$$

Hence, if $\|H(\mathbf{x})\|$ is bounded, say $\|H(\mathbf{x})\| \leq \gamma$ for $\|\overline{\mathbf{x}} - \mathbf{x}\| \leq \varepsilon$, $C = \gamma\beta$ will exhibit the quadratic convergence property

$$\|\mathbf{x}^{(k+1)} - \overline{\mathbf{x}}\| \leq C \|\mathbf{x}^{(k)} - \overline{\mathbf{x}}\|^2$$

whenever $\mathbf{x}^{(0)}$ is selected so that $\|\mathbf{x}^{(0)} - \overline{\mathbf{x}}\| < \min\{1, C^{-1}, \varepsilon, \delta, \eta\}$.

Boundedness of $\|H(\mathbf{x})\|$ is implied by the continuity of $\|H(\mathbf{x})\|$. The latter can be derived as follows. The Frobenius norm is obviously a continuous function with respect to the matrix entries $h_{ij}(\mathbf{x})$ of $H(\mathbf{x})$. So it suffices to prove that each $h_{ij}(\mathbf{x})$ is a continuous function of $\mathbf{x}$. Since the jth column of $H(\mathbf{x})$ is just the solution $\mathbf{h}$ of the linear equation $\nabla F(\mathbf{x})\mathbf{h} = \mathbf{e}_j$, where $\mathbf{e}_j$ is the jth unit vector, Cramer's rule yields

$$h_{ij}(\mathbf{x}) = \frac{\det[\nabla F(\mathbf{x})]_i}{\det \nabla F(\mathbf{x})} \ ,$$

where $[\nabla F(\mathbf{x})]_i$ arises from $\nabla F(\mathbf{x})$ by substitution of $\mathbf{e}_j$ in column i, which shows that $h_{ij}(\mathbf{x})$ is continuous as claimed. □

3.6.1. Minimisation with Newton's Method. Let us return to the problem of minimising a function $f : \mathbb{R}^n \to \mathbb{R}$. With respect to the critical equation $\nabla f(\mathbf{x}) = \mathbf{0}^T$, the Newton iteration becomes

$$\mathbf{x}^{(k+1)} \ = \ \mathbf{x}^{(k)} - [\nabla^2 f(\mathbf{x}^{(k)})]^{-1} \nabla^T f(\mathbf{x}^{(k)}) \ . \tag{3.12}$$

If f is a C^3-function and $\overline{\mathbf{x}}$ is a solution of $\nabla f(\overline{\mathbf{x}}) = \mathbf{0}^T$ with $\det \nabla^2 f(\overline{\mathbf{x}}) \neq 0$, Theorem 3.6 says that the iteration (3.12) converges quadratically to $\overline{\mathbf{x}}$ when started with $\mathbf{x}^{(0)}$ close to $\overline{\mathbf{x}}$.

The Newton Descent Method. When minimising a function $f : \mathbb{R}^n \to \mathbb{R}$ we may use Newton's method for solving the critical equation $\nabla f(\mathbf{x}) = \mathbf{0}^T$. However, a critical point $\overline{\mathbf{x}}$, need not be a local minimiser of f. (It may even be a local maximiser.) Lemma 3.2 tells us that $\overline{\mathbf{x}}$ is a strict local minimiser if the Hessian

$\nabla^2 f(\overline{\mathbf{x}})$ is positive definite (and a strict local maximiser if $\nabla^2 f(\overline{\mathbf{x}})$ is negative definite).

So we may want to force the Newton method to be a descent method, *i.e.,* to generate iterates with decreasing values for f. Hence we ask in which case the *Newton direction*

$$\mathbf{d}^{(k)} = -[\nabla^2 f(\mathbf{x}^{(k)})]^{-1}\mathbf{g}^{(k)} \tag{3.13}$$

is a descent direction in the sense of Section 3.4. Let us, more generally, consider directions $\mathbf{d}^{(k)}$ of the form

$$\mathbf{d}^{(k)} = -\mathbf{H}^{(k)}\mathbf{g}^{(k)} \qquad (\mathbf{H}^{(k)} \in \mathbb{R}^{n\times n})\ .$$

Note that $(\mathbf{g}^{(k)})^T\mathbf{d}^{(k)} = -(\mathbf{g}^{(k)})^T\mathbf{H}^{(k)}\mathbf{g}^{(k)} < 0$ holds (by definition) whenever $\mathbf{H}^{(k)}$ is positive definite. So we record the general rule

$$\mathbf{H}^{(k)} \text{ positive definite} \quad \Longrightarrow \quad \mathbf{d}^{(k)} = -\mathbf{H}^{(k)}\mathbf{g}^{(k)} \text{ is a descent direction} \tag{3.14}$$

for f in $\mathbf{x}^{(k)}$. In particular, the Newton direction (3.13) is a descent direction whenever the Hessian matrix $\nabla^2 f(\mathbf{x}^{(k)})$ (and hence also the inverse matrix $[\nabla^2 f(\mathbf{x}^{(k)})]^{-1}$) is positive definite.

If $\nabla^2 f(\mathbf{x}^{(k)})$ is not positive definite, we may modify the Hessian *via*

$$\nabla^2 f(\mathbf{x}^{(k)}) + \sigma^{(k)}\mathbf{I}\ . \tag{3.15}$$

If $\sigma^{(k)}$ is large enough, this matrix is positive definite and the vector

$$\mathbf{d}^{(k)} = -(\nabla^2 f(\mathbf{x}^{(k)}) + \sigma^{(k)}\mathbf{I})^{-1}\mathbf{g}^{(k)}$$

is a descent direction. This observation, due to *Levenberg-Marquardt*, leads to the following minimisation algorithm.

Newton Descent Method (Levenberg-Marquardt Variant)

INIT: Choose $\mathbf{x}^{(0)} \in \mathbb{R}^n$ and $\varepsilon > 0$ and set $\mathbf{d}^{(0)} = -\mathbf{g}^{(0)}$;

ITER: WHILE $\|\mathbf{g}^{(k)}\| \geq \varepsilon$ DO

BEGIN

Determine a solution $\eta^{(k)}$ for the problem

(*) $\quad \min_{t\geq 0} f(\mathbf{x}^{(k)} + \eta\mathbf{d}^{(k)})$

Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \eta^{(k)}\mathbf{d}^{(k)}$

Determine $\sigma^{(k+1)}$ such that $(\nabla^2 f(\mathbf{x}^{(k+1)}) + \sigma^{(k+1)}\mathbf{I})$ is positive definite

Set $\mathbf{d}^{(k+1)} = -[\nabla^2 f(\mathbf{x}^{(k+1)}) + \sigma^{(k+1)}\mathbf{I}]^{-1}\mathbf{g}^{(k)}$.

END

Remark. *For a given $\sigma^{(k)}$, one can easily test (3.15) for positive definiteness with a diagonalisation algorithm (e.g., an adaptation of the standard Gaussian elimination algorithm achieves this). In practice, the factors $\sigma^{(k+1)}$ are obtained by successively halving or doubling $\sigma^{(k)}$ and checking the positive definiteness condition until a suitable $\sigma^{(k)}$ is found.*

Remark. *The (theoretical) advantage of Newton's method is its (local) quadratic convergence. On the other hand, it requires the evaluation of the Hessian of f in each step. In practice, the second derivatives of f (or even the first derivatives) are often not known explicitly and may be difficult to compute (even approximately).*

Newton's Method - Exercises.

Exercise 3.6.1. Let $q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x}$ a quadratic function. Assuming $\mathbf{A}$ to be positive definite, show that Newton's method finds the global minimiser $\overline{\mathbf{x}} = -\mathbf{A}^{-1}\mathbf{b}$ for q in one step (for any starting point $\mathbf{x}^{(0)}$).

Exercise 3.6.2. Let f be a C^3-function and $\overline{\mathbf{x}} \in \mathbb{R}^n$ such that $\nabla f(\overline{\mathbf{x}}) = \mathbf{0}^T$ and $\nabla^2 f(\overline{\mathbf{x}})$ is positive definite. Assume that the sequence $(\mathbf{x}^{(k)})$ generated by the Levenberg-Marquardt variant converges to $\overline{\mathbf{x}}$. Show: If $\sigma^{(k)} \to 0$, then the sequence is superlinearly convergent. (Hint. Show first: The assumptions imply that the minimisers $t^{(k)}$ of the line-minimisation step $(^*)$ in the Newton descent method have the property $t^{(k)} \to 1$.)

3.7. The Gauss-Newton Method

In this section, we present a modification of the Newton method that relies on first order derivatives only.

We discuss the procedure within the framework of the so-called *nonlinear least square problem*: Given a (nonlinear) C^2-function $r : \mathbb{R}^n \to \mathbb{R}^m$ with m component functions $r_i : \mathbb{R}^n \to \mathbb{R}$, we want to solve the minimisation problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \qquad \text{where} \quad f(\mathbf{x}) = \frac{1}{2}\|r(\mathbf{x})\|^2 = \frac{1}{2}\sum_{i=1}^{m} r_i^2(\mathbf{x}) \; . \tag{3.16}$$

If $\overline{\mathbf{x}} \in \mathbb{R}^n$ satisfies $r(\overline{\mathbf{x}}) = \mathbf{0}$, then $\overline{\mathbf{x}}$ is of course an optimal solution for (3.16). Such solutions do not always exist. For example, if we want to approximate m given data points $(\mathbf{t}_i, y_i)$, where $\mathbf{t}_i \in \mathbb{R}^k$ and $y_i \in \mathbb{R}$, $i = 1, \ldots, m$ $(m \geq n)$, by a function $g(\mathbf{x}, \mathbf{t})$, we may want to solve (3.16) with

$$r_i(\mathbf{x}) = g(\mathbf{x}, \mathbf{t}_i) - y_i, \qquad i = 1, \ldots, m \; .$$

For the derivatives of the function $f(\mathbf{x})$ in (3.16) one readily computes

$$\begin{aligned} \nabla f(\mathbf{x}) &= [r(\mathbf{x})]^T \nabla r(\mathbf{x}) \\ \nabla^2 f(\mathbf{x}) &= [\nabla r(\mathbf{x})]^T \nabla r(\mathbf{x}) + \sum_{i=1}^{m} r_i(\mathbf{x}) \nabla^2 r_i(\mathbf{x}). \end{aligned}$$

The idea behind the Gauss-Newton method is to replace the Hessian $\nabla^2 f(\mathbf{x})$ in the Newton method by the approximation

$$\mathbf{A}(\mathbf{x}) = [\nabla r(\mathbf{x})]^T \nabla r(\mathbf{x}) \; ,$$

which only uses the first order derivatives of the function r. In view of the factors $r_i(\mathbf{x})$ in the second order term of $\nabla^2 f(\mathbf{x})$, the approximation can be expected to be good if the residues $r_i(\mathbf{x})$ are small.

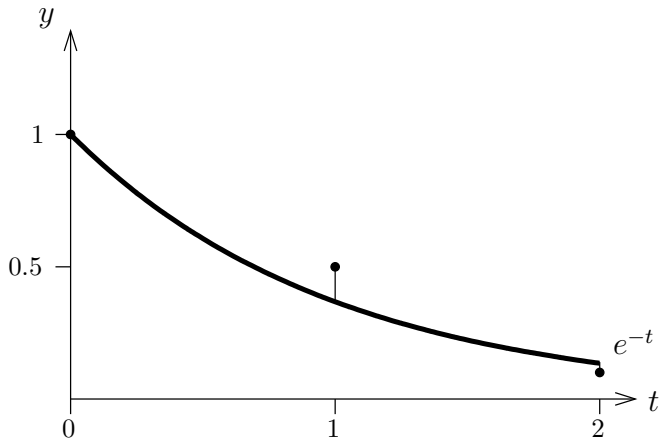


FIGURE 3.4. *Illustration of data approximation*

With $\mathbf{H}_k = [\mathbf{A}(\mathbf{x}_k)]^{-1}$, the iteration step of the *Gauss-Newton Method* for solving the critical equation $\nabla f(\mathbf{x}) = \ r(\mathbf{x})^T \nabla r(\mathbf{x}) = \mathbf{0}^T$ thus becomes

$$\boxed{\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k, \quad \text{where } \mathbf{d}_k = -\mathbf{H}_k[\nabla f(\mathbf{x}_k)]^T} \tag{3.17}$$

The corresponding search method is defined by

$$\boxed{\mathbf{x}_{k+1} = \mathbf{x}_k + t_k \mathbf{d}_k, \quad t_k \text{ a solution of } \ \min_{t \geq 0} f(\mathbf{x}_k + t\mathbf{d}_k).}$$

REMARK. Since the Gauss-Newton method uses approximations of the Hessian of f, we cannot hope for quadratic convergence. However, if the residue $\|r(\overline{\mathbf{x}})\|$ at a local minimiser $\overline{\mathbf{x}}$ is small, the method can be shown to be linearly convergent with a small convergence factor (see e.g. **[24]**).

The Gauss-Newton Method - Exercises.

Exercise 3.7.1. The data $(t_1, y_1) = (0, 1), (t_2, y_2) = (1, 0.5), (t_3, y_3) = (2, 0.1)$ are to be fitted by a function of type $g(x_1, x_2, t) = x_1\, e^{-x_2 t}$ in the sense of (3.16). Perform one step of the Gauss-Newton iteration (3.17) beginning with $(x_1^{(0)}, x_2^{(0)}) = (1, 1)$ (*cf.* Figure 3.4).

Exercise 3.7.2. Show: The matrix $\mathbf{A}(\mathbf{x}_k)$ in the Gauss-Newton method is positive semidefinite. $\mathbf{A}(\mathbf{x}_k)$ is positive definite if and only the matrix $\nabla r(\mathbf{x}_k)$ has rank n.

3.8. Conjugate Directions

As in Section 3.3, we assume that we are given a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ and want to locate a point $\mathbf{x}$ with the property $\nabla f(\mathbf{x}) = \mathbf{0}^T$ according to the conceptual descent method. The method presented in this section is a slight modification of the gradient descent method but it has the advantage that it finds the minimiser of a positive definite quadratic function after at most n steps.

Quadratic Functions. Assume we want to minimise the quadratic function

$$q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x} \ ,$$

where $\mathbf{A}$ is a positive definite matrix and $\mathbf{b} \in \mathbb{R}^n$. Can we choose descent directions such that the descent method computes the unique minimiser $\overline{\mathbf{x}} = -\mathbf{A}^{-1}\mathbf{b}$ in finitely many steps?

If we employ the gradient descent method with line minimisation, a sequence of points is generated according to $\mathbf{x}_{k+1} = \mathbf{x}_k + t_k\mathbf{d}_k$ with $t_k \neq 0$ (since $\mathbf{d}_k$ is a descent direction). Lemma 3.3 guarantees the orthogonality property

$$\nabla q(\mathbf{x}_{k+1})\mathbf{d}_k = \mathbf{g}_{k+1}^T\mathbf{d}_k = 0 \ .$$

Interestingly, a slight strengthening of the condition $\mathbf{g}_{k+1}^T\mathbf{d}_k = 0$ already yields the finite termination property: Assume, the directions $\mathbf{d}_i$ generated satisfy $\mathbf{g}_j^T\mathbf{d}_i = 0$ for all $i = 0, \ldots, j-1$. Then, after n steps we obtain

$$\nabla q(\mathbf{x}_n)\mathbf{d}_i = 0 \ , \qquad \text{for } i = 0, \ldots, n-1 \ .$$

If moreover the vectors $\mathbf{d}_0, \ldots, \mathbf{d}_{n-1}$ are linearly independent, the gradient $\nabla q(\mathbf{x}_n)$ is orthogonal to a basis of $\mathbb{R}^n$, implying $\nabla q(\mathbf{x}_n) = \mathbf{0}^T$.

The next lemma establishes an equivalent condition for the desired property (in case the descent method is applied to the quadratic function $q(\mathbf{x})$).

Lemma 3.7. *The following two statements are equivalent:*

(i) $\mathbf{g}_{j+1}^T\mathbf{d}_i = 0 \quad$ for all $\ 0 \leq i < j \leq k$;
(ii) $\mathbf{d}_j^T\mathbf{A}\mathbf{d}_i = 0 \quad$ for all $\ 0 \leq i < j \leq k$.

Proof. Observing $\nabla q(\mathbf{x}) = \mathbf{x}^T\mathbf{A} + \mathbf{b}^T$, we have

$$\mathbf{g}_{j+1}^T = \nabla q(\mathbf{x}_{j+1}) = (\mathbf{x}_j + t_j\mathbf{d}_j)^T\mathbf{A} + \mathbf{b}^T = \mathbf{g}_j^T + t_j\mathbf{d}_j^T\mathbf{A} \tag{3.18}$$

and hence

$$\mathbf{g}_{j+1}^T\mathbf{d}_i = \mathbf{g}_j^T\mathbf{d}_i + t_j\mathbf{d}_j^T\mathbf{A}\mathbf{d}_i \ .$$

From this identity, the claim follows easily by induction on k. (Recall $t_j \neq 0$). □

Non-zero vectors $\mathbf{d}_0, \ldots, \mathbf{d}_k$ satisfying property (ii) of Lemma 3.7 are called $\mathbf{A}$-*conjugate* (or $\mathbf{A}$-*orthogonal*). The property means that the vectors are mutually orthogonal with respect to the inner product $\langle \mathbf{y}|\mathbf{x}\rangle = \mathbf{y}^T\mathbf{A}\mathbf{x}$ that we had already found useful in the analysis leading to Theorem 3.4. It is straightforward to verify that $\mathbf{A}$-conjugate descent directions are linearly independent.

Our problem now is to find an "easy" method to generate descent directions that are $\mathbf{A}$-conjugate to the directions we already have. To do so, we try the following direct approach (which implicitly takes all of the preceding directions into account but explicitly uses only the last of these directions):

$$\mathbf{d}_k = -\mathbf{g}_k + \alpha_k\mathbf{d}_{k-1} \quad \text{with some appropriate } \alpha_k \ . \tag{3.19}$$

Remark. *Letting the new search direction depend on the previous directions in a smart way often improves the robustness of search algorithms (avoids zigzagging).*

How should we choose α_k for this approach to work? The $\mathbf{A}$-conjugacy requirement says

$$\mathbf{d}_k^T \mathbf{A}\mathbf{d}_{k-1} = 0 \quad \text{and hence} \quad 0 = -\mathbf{g}_k^T \mathbf{A}\mathbf{d}_{k-1} + \alpha_k \mathbf{d}_{k-1}^T \mathbf{A}\mathbf{d}_{k-1} \, .$$

So we must take

$$\alpha_k = \frac{\mathbf{g}_k^T \mathbf{A}\mathbf{d}_{k-1}}{\mathbf{d}_{k-1}^T \mathbf{A}\mathbf{d}_{k-1}} \tag{3.20}$$

It turns out that this choice of α_k actually attains our goal.

Theorem 3.8. *Assume that the conceptual descent method is applied to the quadratic function $q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x}$ ($\mathbf{A} \succ \mathbf{0}$) with directions $\mathbf{d}_k$ of the form (3.19). Then the vectors $\mathbf{d}_k$ are $\mathbf{A}$-conjugate descent directions if and only if α_k is chosen according to (3.20).*

In particular, with the choice (3.20), the algorithm stops after at most n steps with the unique minimiser $\overline{\mathbf{x}} = -\mathbf{A}^{-1}\mathbf{b}$ of q.

Proof. By the foregoing discussion, it suffices to show that (3.20) is sufficient for $\mathbf{A}$-conjugacy. Therefore, proceeding by induction on k, assume that $\mathbf{d}_0, \ldots, \mathbf{d}_{k-1}$ are $\mathbf{A}$-conjugate descent directions.

Note first that $\mathbf{d}_k$ is a descent direction. Indeed, we know $\mathbf{g}_k^T \mathbf{d}_{k-1} = 0$ from Lemma 3.3 and find

$$\mathbf{g}_k^T \mathbf{d}_k = -\mathbf{g}_k^T \mathbf{g}_k + \alpha_k \mathbf{g}_k^T \mathbf{d}_{k-1} = -\mathbf{g}_k^T \mathbf{g}_k < 0 \, ,$$

provided $\mathbf{g}_k \neq \mathbf{0}$. Our choice of α_k guarantees $\mathbf{d}_k^T \mathbf{A}\mathbf{d}_{k-1} = 0$. It therefore remains to verify the orthogonality relation $\mathbf{d}_k^T \mathbf{A}\mathbf{d}_i = 0$ for $i < k-1$.

In view of (3.19) we obtain from our induction hypothesis

$$\mathbf{d}_k^T \mathbf{A}\mathbf{d}_i = -\mathbf{g}_k^T \mathbf{A}\mathbf{d}_i + \alpha_k \mathbf{d}_{k-1}^T \mathbf{A}\mathbf{d}_i = -\mathbf{g}_k^T \mathbf{A}\mathbf{d}_i \, .$$

Combining (3.18) and (3.19), we conclude

$$\mathbf{A}\mathbf{d}_i = (t_i)^{-1}(\mathbf{g}_{i+1} - \mathbf{g}_i) = (t_i)^{-1}(-\mathbf{d}_{i+1} + \alpha_{i+1}\mathbf{d}_i + \mathbf{d}_i - \alpha_i \mathbf{d}_{i-1}) \, .$$

Since $i < k-1$ the induction hypothesis and property (i) of Lemma 3.7 finally establishes the desired orthogonality:

$$\mathbf{d}_k^T \mathbf{A}\mathbf{d}_i = -\mathbf{g}_k^T \mathbf{A}\mathbf{d}_i = -(t_i)^{-1}\mathbf{g}_k^T(-\mathbf{d}_{i+1} + \alpha_{i+1}\mathbf{d}_i + \mathbf{d}_i - \alpha_i \mathbf{d}_{i-1}) = 0 \, .$$

□

The conjugate gradient version of our conceptual descent method now proceeds as follows.

Conjugate Gradient Method

INIT: Choose $\mathbf{x}_0 \in \mathbb{R}^n$ and $\varepsilon > 0$ and set $\mathbf{d}_0 = -\mathbf{g}_0$;
ITER: WHILE $\|\mathbf{g}_k\| \geq \varepsilon$ DO
BEGIN
Determine a solution t_k for the problem
(*) $\min_{t \geq 0} f(\mathbf{x}_k + t\mathbf{d}_k)$
Set $\mathbf{x}_{k+1} = \mathbf{x}_k + t_k \mathbf{d}_k$.
Set $\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \alpha_{k+1}\mathbf{d}_k$.
END

Note that we have not specified the choice of α_{k+1} in the schematic outline. If $f(\mathbf{x}) = q(\mathbf{x})$ is a quadratic function (as in Theorem 3.8), we should of course take α_{k+1} according to (3.20). In this case it is straightforward to derive (for example, *via* relation (3.18)) the equivalent expressions

$$\alpha_{k+1} = \frac{\mathbf{g}_{k+1}^T \mathbf{A}\mathbf{d}_k}{\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k} = \frac{\mathbf{g}_{k+1}^T(\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{d}_k^T(\mathbf{g}_{k+1} - \mathbf{g}_k)} = \frac{\mathbf{g}_{k+1}^T(\mathbf{g}_{k+1} - \mathbf{g}_k)}{\|\mathbf{g}_k\|^2} = \frac{\|\mathbf{g}_{k+1}\|^2}{\|\mathbf{g}_k\|^2} \tag{3.21}$$

Non-quadratic Functions. Our formulation of the conjugate gradient method does not explicitly rely on the matrix $\mathbf{A}$ and hence may be applied to arbitrary (differentiable) functions f. In doing so, one must decide on a choice for α_{k+1} that does not depend on $\mathbf{A}$.

Several suggestions have been made that are all based on the analogy with the quadratic case (see Ex. 3.21). In the non-quadratic case, however, these choices are no longer equivalent:

$$\textit{Hestenes-Stiefel (1952):} \qquad \alpha_{k+1} = \frac{\mathbf{g}_{k+1}^T(\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{d}_k^T(\mathbf{g}_{k+1} - \mathbf{g}_k)}$$

$$\textit{Fletcher-Reeves (1964):} \qquad \alpha_{k+1} = \frac{\|\mathbf{g}_{k+1}\|^2}{\|\mathbf{g}_k\|^2}$$

$$\textit{Polak-Ribiere (1969):} \qquad \alpha_{k+1} = \frac{\mathbf{g}_{k+1}^T(\mathbf{g}_{k+1} - \mathbf{g}_k)}{\|\mathbf{g}_k\|^2}$$

In the non-quadratic case we generally do **not** have the guarantee that the conjugate gradient methods stops after at most n iterations with a critical point.

Remark. *In practical implementations of the conjugate gradient method, often a* restart *after each cycle of n steps is carried out by setting $\alpha_n = 0$. This modification has stabilization effects but also allows to prove the $n-$step quadratic convergence property (*cf. **[3]**):*

$$\|\mathbf{x}_{n+k} - \overline{\mathbf{x}}\| \leq c \, \|\mathbf{x}_k - \overline{\mathbf{x}}\|^2 \, .$$

Conjugate Directions - Exercises.

Exercise 3.8.1. Under the assumptions of Theorem 3.8, show that the iteration point $\mathbf{x}_{k+1}$ is the (global) minimiser of the quadratic function q on the affine subspace

$$S_k = \{\mathbf{x}_0 + \gamma_0\mathbf{d}_0 + \ldots + \gamma_k\mathbf{d}_k \mid \gamma_0, \ldots, \gamma_k \in \mathbb{R}\} .$$

Exercise 3.8.2. Verify the equations (3.21) for the case of the quadratic function $q(\mathbf{x})$.

Exercise 3.8.3. Consider the function $f : \mathbb{R}^n \to \mathbb{R}$ given by

$$f(\mathbf{x}) = \|\mathbf{x}\|_2^2 + 42x_1 .$$

(a) Given conjugate directions $\mathbf{d}_0, \ldots, \mathbf{d}_k$ and the point $\mathbf{x}_k$, show that the direction of the Conjugate Gradient Method for f in iteration k is equal to

$$\mathbf{d}_k = -\mathbf{g}_k .$$

(b) Prove that the gradient descent method with exact line minimisation finds the minimiser of f in at most n steps independent of the starting point $\mathbf{x}_0$.

3.9. Quasi-Newton Methods

Motivated by the discussion of general descent and Newton methods, we investigate in this section descent methods where the search directions take the general form

$$\mathbf{d}_k = -\mathbf{H}_k\mathbf{g}_k , \quad \text{where } \mathbf{H}_k \in \mathbb{R}^{n\times n} \text{ and } \mathbf{g}_k = [\nabla f(\mathbf{x}_k)]^T . \tag{3.22}$$

$\mathbf{H}_k$ should be a "good" approximation of the inverse Hessian $[\nabla^2 f(\mathbf{x}_k)]^{-1}$. The computationally simplest choice here is $\mathbf{H}_k = \mathbf{I}$, which exhibits steepest descent as a "Newton-type" method.

There are several desirable properties for $\mathbf{H}_k$ to have. For example,

(I) $\mathbf{H}_k$ is positive definite (so that $\mathbf{d}_k$ is a descent direction).
(II) The iteration step $\mathbf{H}_k \to \mathbf{H}_{k+1}$ involves a relatively simple update matrix $\mathbf{E}_k = \mathbf{H}_{k+1} - \mathbf{H}_k$.
(III) The descent method reduces to a conjugate gradient method (see Section 3.8) in the case where f is a quadratic function.

Our fourth and final requirement (*cf.* below) is also motivated by the quadratic case. Let $\mathbf{A}$ be a positive definite symmetric matrix and consider the quadratic function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x} .$$

Then $\mathbf{g}_k = \mathbf{A}\mathbf{x}_k + \mathbf{b}$ and $\nabla^2 f(\mathbf{x}) = \mathbf{A}$. So Newton's method in its pure form chooses $\mathbf{H}_k = \mathbf{H}_{k+1} = \mathbf{A}^{-1}$ and we obtain the minimiser $\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}_k\mathbf{g}_k = -\mathbf{A}^{-1}\mathbf{b}$ in a single step. Since $\mathbf{g}_{k+1} = \mathbf{0}$ we have

$$\mathbf{x}_{k+1} - \mathbf{x}_k = \mathbf{H}_{k+1}(\mathbf{g}_{k+1} - \mathbf{g}_k) .$$

We say that the descent method is *quasi-Newton* if the iterates $\mathbf{H}_k$ satisfy the relation

(IV) $\mathbf{x}_{k+1} - \mathbf{x}_k = \mathbf{H}_{k+1}(\mathbf{g}_{k+1} - \mathbf{g}_k)$.

Assuming that we can satisfy the requirements (I)-(IV), we are led to the following minimisation algorithm

Quasi-Newton Method

INIT: Choose some $\mathbf{x}_0 \in \mathbb{R}^n$, $\mathbf{H}_0 \in \mathbb{R}^{n\times n}$ positive definite and $\varepsilon > 0$
ITER: WHILE $\|\mathbf{g}_k\| \geq \varepsilon$ DO
BEGIN
Set $\mathbf{d}_k = -\mathbf{H}_k\mathbf{g}_k$,
Determine a solution t_k for the problem
(*) $\min_{t\geq 0} f(\mathbf{x}_k + t\mathbf{d}_k)$
Set $\mathbf{x}_{k+1} = \mathbf{x}_k + t_k\mathbf{d}_k$ and update $\mathbf{H}_{k+1} = \mathbf{H}_k + \mathbf{E}_k$.
END

How can we satisfy the conditions (I)-(IV)? We could choose $\mathbf{E}_k$ to be, for example, of the form

$$\mathbf{E}_k = \alpha\mathbf{u}\mathbf{u}^T + \beta\mathbf{v}\mathbf{v}^T \qquad (\mathbf{u}, \mathbf{v} \in \mathbb{R}^n)$$

such that $\mathbf{H}_{k+1} = \mathbf{H}_k + \mathbf{E}_k$ is positive definite.

Writing $\boldsymbol{\gamma}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$ and $\boldsymbol{\delta}_k = \mathbf{x}_{k+1} - \mathbf{x}_k = t_k\mathbf{d}_k$, the quasi-Newton condition (IV) stipulates $\mathbf{H}_{k+1}\boldsymbol{\gamma}_k = \boldsymbol{\delta}_k$, *i.e.*,

$$\mathbf{H}_{k+1}\boldsymbol{\gamma}_k = \mathbf{H}_k\boldsymbol{\gamma}_k + \alpha\mathbf{u}\mathbf{u}^T\boldsymbol{\gamma}_k + \beta\mathbf{v}\mathbf{v}^T\boldsymbol{\gamma}_k = \boldsymbol{\delta}_k \ .$$

The DFP-Method. Davidon, Fletcher and Powell (1963) have recognised that the latter condition can easily be fulfilled with the choice

$$\mathbf{u} = \boldsymbol{\delta}_k \quad \text{and} \quad \mathbf{v} = \mathbf{H}_k\boldsymbol{\gamma}_k \tag{3.23}$$

and the parameters $\alpha = 1/\boldsymbol{\delta}_k^T\boldsymbol{\gamma}_k$ and $\beta = -1/\boldsymbol{\gamma}_k^T\mathbf{H}_k\boldsymbol{\gamma}_k$.

Allowing an additional mixed term, let us actually consider a slightly more general form

$$\begin{aligned} \mathbf{E}_k &= \alpha\mathbf{u}\mathbf{u}^T + \beta\mathbf{v}\mathbf{v}^T + \mu(\mathbf{u}\mathbf{v}^T + \mathbf{v}\mathbf{u}^T) \\ &= (\mathbf{u}, \mathbf{v}) \begin{pmatrix} \alpha & \mu \\ \mu & \beta \end{pmatrix} \begin{pmatrix} \mathbf{u}^T \\ \mathbf{v}^T \end{pmatrix} \end{aligned} \tag{3.24}$$

with $\mathbf{u} = \boldsymbol{\delta}_k$ and $\mathbf{v} = \mathbf{H}_k\boldsymbol{\gamma}_k$ as in the DFP-approach (3.23) – but no *a priori* assumptions on the parameters $\alpha, \beta, \mu \in \mathbb{R}$.

It turns out that (3.24) already ensures any quasi-Newton method to be a method of conjugate directions in the case of quadratic functions!

LEMMA 3.9. *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be a positive definite symmetric matrix and*

$$q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x} \ .$$

Assume that the Quasi-Newton method is applied to q so that the $\mathbf{E}_k$ *have the form (3.24) and the* $\mathbf{H}_{k+1} = \mathbf{H}_k + \mathbf{E}_k$ *satisfy condition (IV). Then the generated search directions satisfy*

$$\mathbf{d}_j^T \mathbf{A} \mathbf{d}_i = 0 \qquad \text{for all } \ 0 \le i < j \le k.$$

Proof. We suppose that the method produces the distinct iterates $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k$. Proceeding by induction on k, we show that

$$\mathbf{H}_k \boldsymbol{\gamma}_i = \boldsymbol{\delta}_i \quad \text{and} \quad \mathbf{d}_k^T \mathbf{A} \mathbf{d}_i = \mathbf{g}_k^T \mathbf{d}_i = 0 \quad \text{for all } 0 \le i \le k-1 \,. \tag{3.25}$$

The case $k = 0$ is trivial (no index i). Assume now that (3.25) holds for $k \ge 0$. We must show that (3.25) is also valid for $k+1$.

In view of $\boldsymbol{\delta}_i = \mathbf{x}_{i+1} - \mathbf{x}_i = t_i \mathbf{d}_i$, $\boldsymbol{\gamma}_k = \mathbf{g}_{k+1} - \mathbf{g}_k = \mathbf{A}\boldsymbol{\delta}_k$ and the induction hypothesis $\mathbf{d}_k^T \mathbf{A} \mathbf{d}_i = 0$, we deduce for $i \le k-1$,

$$\boldsymbol{\gamma}_k^T \mathbf{H}_k \boldsymbol{\gamma}_i = \boldsymbol{\gamma}_k^T \boldsymbol{\delta}_i = \boldsymbol{\delta}_k^T \mathbf{A} \boldsymbol{\delta}_i = t_k t_i \mathbf{d}_k^T \mathbf{A} \mathbf{d}_i = 0$$

and $\boldsymbol{\delta}_k^T \boldsymbol{\gamma}_i = \boldsymbol{\delta}_k^T \mathbf{A} \boldsymbol{\delta}_i = 0$. Hence we obtain for all $i \le k-1$

$$\mathbf{E}_k \boldsymbol{\gamma}_i = (\boldsymbol{\delta}_k, \mathbf{H}_k \boldsymbol{\gamma}_k) \begin{pmatrix} \alpha & \mu \\ \mu & \beta \end{pmatrix} \begin{pmatrix} \boldsymbol{\delta}_k^T \boldsymbol{\gamma}_i \\ \boldsymbol{\gamma}_k^T \mathbf{H}_k \boldsymbol{\gamma}_i \end{pmatrix} = \mathbf{0}$$

and consequently $\mathbf{H}_{k+1} \boldsymbol{\gamma}_i = \mathbf{H}_k \boldsymbol{\gamma}_i + \mathbf{E}_k \boldsymbol{\gamma}_i = \boldsymbol{\delta}_i$.

For $i = k$, the equality $\mathbf{H}_{k+1} \boldsymbol{\gamma}_i = \mathbf{H}_{k+1} \boldsymbol{\gamma}_k = \boldsymbol{\delta}_k$ is just the quasi-Newton property (IV), which we assume to hold at the outset.

Because $t_i \ne 0$ (by assumption), the relations $\mathbf{d}_{k+1} = -\mathbf{H}_{k+1} \mathbf{g}_{k+1}$ (*cf.* (3.22)) and $\mathbf{H}_{k+1} \boldsymbol{\gamma}_i = \boldsymbol{\delta}_i$ allow us to observe next

$$\mathbf{d}_{k+1}^T \mathbf{A} \mathbf{d}_i = \frac{1}{t_i} \mathbf{d}_{k+1}^T \mathbf{A} \boldsymbol{\delta}_i = \frac{1}{t_i} \mathbf{d}_{k+1}^T \boldsymbol{\gamma}_i = -\frac{1}{t_i} \mathbf{g}_{k+1}^T \mathbf{H}_{k+1} \boldsymbol{\gamma}_i = -\frac{1}{t_i} \mathbf{g}_{k+1}^T \boldsymbol{\delta}_i = -\mathbf{g}_{k+1}^T \mathbf{d}_i \,.$$

It thus suffices to verify $\mathbf{g}_{k+1}^T \mathbf{d}_i = 0$ for all $i \le k$. The equality $\mathbf{g}_{k+1}^T \mathbf{d}_i = 0$ holds for $i = k$ because $\mathbf{g}_{k+1}^T \mathbf{d}_k = 0$ is satisfied by any descent method with a line minimisation subroutine (*cf.* Lemma 3.3).

Writing $\mathbf{g}_{k+1} = \mathbf{A}(\mathbf{x}_k + t_k \mathbf{d}_k) + \mathbf{b} = \mathbf{g}_k + t_k \mathbf{A} \mathbf{d}_k$, we finally infer from the induction hypothesis also for $i \le k-1$ the desired relation

$$\mathbf{g}_{k+1}^T \mathbf{d}_i = \mathbf{g}_k^T \mathbf{d}_i + t_k \mathbf{d}_k^T \mathbf{A} \mathbf{d}_i = 0 \,.$$

□

Lemma 3.9 says in particular that the DFP-parameter choice according to (3.23) (and $\mu = 0$) turns the Quasi-Newton method into a minimisation procedure enjoying all of the properties (I)-(IV).

The Broyden Family. We would like to take advantage of the flexibility of the form (3.24) and determine more general parameters α, β, μ that imply all of the properties (I)-(IV) for the Quasi-Newton method.

The quasi-Newton condition $\boldsymbol{\delta}_k = \mathbf{H}_{k+1}\boldsymbol{\gamma}_k = \mathbf{H}_k\boldsymbol{\gamma}_k + \mathbf{E}_k\boldsymbol{\gamma}_k$ becomes (still with $\mathbf{u} = \boldsymbol{\delta}_k$ and $\mathbf{v} = \mathbf{H}_k\boldsymbol{\gamma}_k$):

$$\begin{aligned}\mathbf{u} &= \mathbf{v} + \alpha\mathbf{u}\mathbf{u}^T\boldsymbol{\gamma}_k + \beta\mathbf{v}\mathbf{v}^T\boldsymbol{\gamma}_k + \mu(\mathbf{u}\mathbf{v}^T + \mathbf{v}\mathbf{u}^T)\boldsymbol{\gamma}_k \\ &= \mathbf{v}(1 + \beta\mathbf{v}^T\boldsymbol{\gamma}_k + \mu\mathbf{u}^T\boldsymbol{\gamma}_k) + \mathbf{u}(\alpha\mathbf{u}^T\boldsymbol{\gamma}_k + \mu\mathbf{v}^T\boldsymbol{\gamma}_k)\ .\end{aligned}$$

To satisfy this relation, we select the parameters so that

$$\begin{array}{rcrcr} \alpha\mathbf{u}^T\boldsymbol{\gamma}_k & + & \mu\mathbf{v}^T\boldsymbol{\gamma}_k & = & 1 \\ \beta\mathbf{v}^T\boldsymbol{\gamma}_k & + & \mu\mathbf{u}^T\boldsymbol{\gamma}_k & = & -1 \end{array}$$

This linear system of two equations in the three variables α, β, μ allows us to choose one parameter arbitrarily. Setting $\mu = -\Phi/\mathbf{u}^T\boldsymbol{\gamma}_k$ for some $\Phi \in \mathbb{R}$, we find

$$\beta = \frac{\Phi - 1}{\mathbf{v}^T\boldsymbol{\gamma}_k} \qquad \text{and} \qquad \alpha = \frac{1}{\mathbf{u}^T\boldsymbol{\gamma}_k}\left(1 + \Phi \cdot \frac{\mathbf{v}^T\boldsymbol{\gamma}_k}{\mathbf{u}^T\boldsymbol{\gamma}_k}\right)$$

Note that these terms are well-defined if $\mathbf{H}_k$ is positive definite. Indeed,

$$\mathbf{u}^T\boldsymbol{\gamma}_k = \boldsymbol{\delta}_k^T(\mathbf{g}_{k+1} - \mathbf{g}_k) = t_k\mathbf{d}_k^T(\mathbf{g}_{k+1} - \mathbf{g}_k) = -t_k\mathbf{d}_k^T\mathbf{g}_k > 0 \tag{3.26}$$

because $\mathbf{g}_{k+1}^T\mathbf{d}_k = 0$ is guaranteed by the line minimisation and $\mathbf{d}_k$ is a descent direction (*i.e.*, $\mathbf{g}_k^T\mathbf{d}_k < 0$) if $\mathbf{H}_k$ is positive definite. Moreover, we then have $\mathbf{v}^T\boldsymbol{\gamma}_k = \boldsymbol{\gamma}_k^T\mathbf{H}_k\boldsymbol{\gamma}_k > 0$.

Substituting these expressions for α, β, μ into (3.24), we arrive at the explicit update formula:

$$\begin{aligned}\mathbf{H}_{k+1} &= \mathbf{H}_k + \frac{\boldsymbol{\delta}_k\boldsymbol{\delta}_k^T}{\boldsymbol{\delta}_k^T\boldsymbol{\gamma}_k} - \frac{\mathbf{H}_k\boldsymbol{\gamma}_k\boldsymbol{\gamma}_k^T\mathbf{H}_k}{\boldsymbol{\gamma}_k^T\mathbf{H}_k\boldsymbol{\gamma}_k} + \Phi\mathbf{w}\mathbf{w}^T \\ &\qquad \text{where} \quad \mathbf{w} = (\boldsymbol{\gamma}_k^T\mathbf{H}_k\boldsymbol{\gamma}_k)^{\frac{1}{2}}\left(\frac{\boldsymbol{\delta}_k}{\boldsymbol{\delta}_k^T\boldsymbol{\gamma}_k} - \frac{\mathbf{H}_k\boldsymbol{\gamma}_k}{\boldsymbol{\gamma}_k^T\mathbf{H}_k\boldsymbol{\gamma}_k}\right)\ .\end{aligned} \tag{3.27}$$

The class of formulas (3.27) (parameterised by Φ) is known as the *Broyden-family*. Note that if $\mathbf{H}_k$ is positive definite, then $\mathbf{H}_{k+1}$ is well-defined by (3.27). As we want the $\mathbf{H}_k$ to be positive definite anyway, the crucial question therefore is: When do Broyden updates preserve positive definiteness?

Lemma 3.10. *Let $\Phi \geq 0$. Then the update formula (3.27) ensures*

$$\mathbf{H}_k \textit{ positive definite} \quad \Longrightarrow \quad \mathbf{H}_{k+1} \textit{ positive definite}\ .$$

Proof. Let $\mathbf{x} \neq \mathbf{0}$ be an arbitrary vector. We claim $\mathbf{x}^T\mathbf{H}_{k+1}\mathbf{x} > 0$. Now (3.27) implies

$$\mathbf{x}^T\mathbf{H}_{k+1}\mathbf{x} = \mathbf{x}^T\mathbf{H}_k\mathbf{x} - \frac{\mathbf{x}^T\mathbf{H}_k\boldsymbol{\gamma}_k\boldsymbol{\gamma}_k^T\mathbf{H}_k\mathbf{x}}{\boldsymbol{\gamma}_k^T\mathbf{H}_k\boldsymbol{\gamma}_k} + \frac{\mathbf{x}^T\boldsymbol{\delta}_k\boldsymbol{\delta}_k^T\mathbf{x}}{\boldsymbol{\delta}_k^T\boldsymbol{\gamma}_k} + \Phi\mathbf{x}^T\mathbf{w}\mathbf{w}^T\mathbf{x}$$

Since $\mathbf{H}_k$ is positive definite by assumption, we have $\boldsymbol{\gamma}_k^T\mathbf{H}_k\boldsymbol{\gamma}_k > 0$ and $\boldsymbol{\delta}_k^T\boldsymbol{\gamma}_k > 0$ (*cf.* (3.26)). Since $\mathbf{x}^T\boldsymbol{\delta}_k\boldsymbol{\delta}_k^T\mathbf{x} = |\boldsymbol{\delta}_k^T\mathbf{x}|^2$ and $\mathbf{x}^T\mathbf{w}\mathbf{w}^T\mathbf{x} = |\mathbf{w}^T\mathbf{x}|^2$, the last

two summands above are non-negative. Introducing the inner product $\langle \mathbf{x}|\mathbf{y}\rangle = \mathbf{x}^T\mathbf{H}_k\mathbf{y}$, the first two terms become

$$\mathbf{x}^T\mathbf{H}_k\mathbf{x} - \frac{\mathbf{x}^T\mathbf{H}_k\boldsymbol{\gamma}_k\boldsymbol{\gamma}_k^T\mathbf{H}_k\mathbf{x}}{\boldsymbol{\gamma}_k^T\mathbf{H}_k\boldsymbol{\gamma}_k} = \langle \mathbf{x}|\mathbf{x}\rangle - \frac{\langle \boldsymbol{\gamma}_k|\mathbf{x}\rangle^2}{\langle \boldsymbol{\gamma}_k|\boldsymbol{\gamma}_k\rangle} \;\geq\; 0$$

by the Cauchy-Schwarz inequality $\langle \boldsymbol{\gamma}_k|\mathbf{x}\rangle^2 \leq \langle \boldsymbol{\gamma}_k|\boldsymbol{\gamma}_k\rangle\langle \mathbf{x}|\mathbf{x}\rangle$.

If $\mathbf{x}$ is not a scalar multiple of $\boldsymbol{\gamma}_k$, the Cauchy-Schwarz inequality is strict and $\mathbf{x}^T\mathbf{H}_{k+1}\mathbf{x} > 0$ follows. If $\mathbf{x} = \lambda\boldsymbol{\gamma}_k$ for some $\lambda \neq 0$, then

$$\frac{\mathbf{x}^T\boldsymbol{\delta}_k\boldsymbol{\delta}_k^T\mathbf{x}}{\boldsymbol{\delta}_k^T\boldsymbol{\gamma}_k} = \lambda^2\boldsymbol{\delta}_k^T\boldsymbol{\gamma}_k \;>\; 0$$

and $\mathbf{x}^T\mathbf{H}_{k+1}\mathbf{x} > 0$ follows as before. □

We summarise the main result.

Theorem 3.11. *The Quasi-Newton method with the update (3.27) and $\Phi \geq 0$ is a method of conjugate directions and preserves positiveness of the iterates $\mathbf{H}_k$.*

The parameter choice $\Phi = 0$ in (3.27) results in the DFP-method. Broyden, Fletcher, Goldfarb and Shanno (1970) suggest the choice $\Phi = 1$. The latter is called *BFGS-method*. According to Theorem 3.11, both parameter choices guarantee *bona fide* Quasi-Newton methods.

Remark. *Being a method of conjugate directions, a Quasi-Newton method identifies in particular minimisers of quadratic functions in at most n steps. It can be shown that for quadratic functions f – using exact line search in the line minimisation step (*) – Broyden family updates yield precisely the method of Fletcher-Reeves (i.e., starting from the same point $\mathbf{x}_0$, both methods generate the same sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k = \overline{\mathbf{x}}$). On non-quadratic functions, however, the methods may differ. The Quasi-Newton method with Broyden updates (3.27) and $\Phi \geq 0$ can be shown to be superlinearly convergent (see, e.g.,* [**5**] *and* [**24**]*).*

Note that, being a method of conjugate gradients, a Quasi-Newton method (with restart after n steps) has the n-step quadratic convergence property mentioned in the Remark on p. 33.

Quasi-Newton Methods - Exercises.

Exercise 3.9.1. Assuming that $\mathbf{E}_k \in \mathbb{R}^{n\times n}$ is of the form (3.24) show that $\mathbf{E}_k$ is symmetric and has rank at most 2.

3.10. The Subgradient Method.

The algorithms (and optimality criteria) for unconstrained minimisation developed in the preceding sections of this chapter assume that the objective function f is differentiable. In particular, the gradient descent method presupposes that the gradients $\nabla f(\mathbf{x}_k)$ exist (and can be computed). Without this assumption, very little structural information is available in general to help in the design of efficient minimisation procedures. However, There are other properties that, if we know the function satisfies them, can be used to obtain working optimisation methods. In

this section, we describe one such method for the minimisation of convex functions. It is an extension of the gradient descent method that relies on *sub*gradients instead of gradients, which means that the method applies also to non-differentiable convex functions.

Recall from Section 2.2 that a vector $\mathbf{g}_k \in \mathbb{R}^n$ is said to be a subgradient of the function $f : \mathbb{R}^n \to \mathbb{R}$ at $\mathbf{x}_k$ if for all $\mathbf{x} \in \mathbb{R}^n$,

$$f(\mathbf{x}) - f(\mathbf{x}_k) \;\geq\; \mathbf{g}_k^T(\mathbf{x} - \mathbf{x}_k) \; .$$

Denoting by $\partial f(\mathbf{x})$ the set of all subgradients of f at $\mathbf{x}$, Theorem 2.14 says that f is convex precisely when $\partial f(\mathbf{x}) \neq \emptyset$ for all $\mathbf{x} \in \mathbb{R}^n$. Note in particular that $\overline{\mathbf{x}}$ is a (global) minimiser of the convex function f if and only if $\mathbf{0} \in \partial f(\overline{\mathbf{x}})$ (*cf.* (2.4)).

Consider, for example, the *maximum function*

$$f(\mathbf{x}) = \max_{1 \leq j \leq m} \{\mathbf{a}_j^T \mathbf{x} + b_j\} \; ,$$

which is a piece-wise linear convex function but not everywhere differentiable as the gradient $\nabla f(\mathbf{x})$ is not defined at certain points. A straightforward modification of the steepest descent method for f would simply replace the gradient $\nabla f(\mathbf{x}_k)$ by a subgradient $\mathbf{g}_k$. (Such a subgradient is easy to compute, *cf.* Ex. 3.10.5.)

idea: Subgradient Descent (does not function generally)

INIT: Choose a starting point $\mathbf{x}_0 \in \mathbb{R}^n$ and $\varepsilon > 0$

ITER: WHILE $\mathbf{0} \notin \partial f(\mathbf{x}_k)$ DO

BEGIN

Choose a subgradient $\mathbf{g}_k \in \partial f(\mathbf{x}_k)$, set $\mathbf{d}_k = -\mathbf{g}_k / \|\mathbf{g}_k\|$

Determine a solution t_k for the problem

(*) $\quad \min_{t \geq 0} f(\mathbf{x}_k + t\mathbf{d}_k)$

Set $\mathbf{x}_{k+1} = \mathbf{x}_k + t_k \mathbf{d}_k$.

END

The problem with this approach is that $-\mathbf{g}_k \in \partial f(\mathbf{x}_k) \setminus \{\mathbf{0}\}$ may not be a descent direction (*cf.* Ex. 3.10.1). So the line minimisation (*) may compute the solution $t_k = 0$ and the procedure gets stuck at a non-optimal point $\mathbf{x}_k$.

As exemplified in Ex. 3.10.1, a subgradient $\mathbf{g}_k$ need not yield a descent direction. Subgradients of convex functions, however, always yield search directions $\mathbf{d}_k = -\mathbf{g}_k / \|\mathbf{g}_k\|$ that bring us closer to (any) minimiser $\overline{\mathbf{x}}$. Indeed, the subgradient inequality $f(\overline{\mathbf{x}}) - f(\mathbf{x}_k) \geq \mathbf{g}_k^T(\overline{\mathbf{x}} - \mathbf{x}_k)$ can equivalently be stated as

$$\mathbf{d}_k^T(\overline{\mathbf{x}} - \mathbf{x}_k) \geq \frac{f(\mathbf{x}_k) - f(\overline{\mathbf{x}})}{\|\mathbf{g}_k\|} > 0 \; , \tag{3.28}$$

i.e., the search direction $\mathbf{d}_k$ in a non-optimal point $\mathbf{x}_k$ (with $\mathbf{g}_k \neq \mathbf{0}$ and $f(\mathbf{x}_k) > f(\overline{\mathbf{x}})$) and the difference vector $\overline{\mathbf{x}} - \mathbf{x}_k$ form an acute angle. This fact can be quantified.

LEMMA 3.12. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function with a minimiser $\overline{\mathbf{x}}$ and $\mathbf{g}_k \neq \mathbf{0}$ a subgradient of f at a non-optimal point $\mathbf{x}_k$. Then for $0 < t_k \leq \mathbf{d}_k^T(\overline{\mathbf{x}} - \mathbf{x}_k)$, the point $\mathbf{x}_{k+1} = \mathbf{x}_k + t_k\mathbf{d}_k$ satisfies*

$$\|\mathbf{x}_{k+1} - \overline{\mathbf{x}}\|^2 \leq \|\mathbf{x}_k - \overline{\mathbf{x}}\|^2 - t_k\mathbf{d}_k^T(\overline{\mathbf{x}} - \mathbf{x}_k) \; .$$

Proof. The relation $\mathbf{x}_{k+1} - \overline{\mathbf{x}} = (\mathbf{x}_k - \overline{\mathbf{x}}) + t_k\mathbf{d}_k$ implies

$$\|\mathbf{x}_{k+1} - \overline{\mathbf{x}}\|^2 = \|\mathbf{x}_k - \overline{\mathbf{x}}\|^2 + 2t_k\mathbf{d}_k^T(\mathbf{x}_k - \overline{\mathbf{x}}) + t_k^2 \leq \|\mathbf{x}_k - \overline{\mathbf{x}}\|^2 - t_k\mathbf{d}_k^T(\overline{\mathbf{x}} - \mathbf{x}_k) \; .$$

□

The previous Lemma asserts that even if $\mathbf{g}_k \in \partial f(\mathbf{x}_k)$ does not define a descent direction, the iterate $\mathbf{x}_{k+1} = \mathbf{x}_k + t_k\mathbf{d}_k$ is an improvement over $\mathbf{x}_k$ if $t_k > 0$ is "small". The latter, however, means that we should control the size of t_k externally rather than computing t_k by line search.

The *subgradient method* for minimising a convex function $f : \mathbb{R}^n \to \mathbb{R}$ proceeds as follows.

Subgradient Method

INIT: Choose a starting point $\mathbf{x}_0 \in \mathbb{R}^n$ and parameters $t_k > 0$
ITER: Choose a subgradient $\mathbf{g}_k \in \partial f(\mathbf{x}_k)$. (If $\mathbf{g}_k = \mathbf{0}$, STOP)
Set $\mathbf{d}_k = -\mathbf{g}_k/\|\mathbf{g}_k\|$ and $\mathbf{x}_{k+1} = \mathbf{x}_k + t_k\mathbf{d}_k$.

THEOREM 3.13. *Assume that the convex function $f : \mathbb{R}^n \to \mathbb{R}$ has at least one minimiser $\overline{\mathbf{x}}$. Choose parameters $t_k > 0$ such that*

$$\lim_{k\to\infty} t_k = 0 \quad \textit{and} \quad \sum_{k=0}^{\infty} t_k = \infty \; .$$

Then the subgradient method generates points $\mathbf{x}_0, \mathbf{x}_1, \ldots$ such that

$$\min\{f(\mathbf{x}_0), \ldots, f(\mathbf{x}_k)\} \;\to\; f(\overline{\mathbf{x}}) \qquad (k \to \infty) \; .$$

Proof. If $\mathbf{g}_k = \mathbf{0}$ for some k, then $\mathbf{x}_k$ is a (global) minimiser and the claim is true. Hence assume the generated sequence $\mathbf{x}_0, \mathbf{x}_1, \ldots$ is infinite and $\mathbf{g}_k \neq \mathbf{0}$ for all k. To establish the claimed convergence suppose to the contrary that $f(\mathbf{x}_k) \geq \alpha > f(\overline{\mathbf{x}})$ for all k.

Since f is continuous at $\overline{\mathbf{x}}$ (*cf.* Theorem 2.13), there exists some $\rho > 0$ such that

$$f(\overline{\mathbf{x}} - \rho\mathbf{d}) \leq \alpha \quad \text{for all } \; \mathbf{d} \in \mathbb{R}^n, \; \|\mathbf{d}\| \leq 1 \; .$$

In particular, the points $\mathbf{x}_k^\rho := \overline{\mathbf{x}} - \rho\mathbf{d}_k$ satisfy $f(\mathbf{x}_k^\rho) \leq \alpha \leq f(\mathbf{x}_k)$. Hence the subgradient inequality implies

$$0 \geq f(\mathbf{x}_k^\rho) - f(\mathbf{x}_k) \geq \mathbf{g}_k^T(\mathbf{x}_k^\rho - \mathbf{x}_k) = \mathbf{g}_k^T(\overline{\mathbf{x}} - \mathbf{x}_k - \rho\mathbf{d}_k) \; .$$

Dividing by $\|\mathbf{g}_k\|$, we obtain

$$\mathbf{d}_k^T(\overline{\mathbf{x}} - \mathbf{x}_k) \geq \rho\|\mathbf{d}_k\|^2 = \rho \; .$$

From Lemma 3.12 we conclude for $t_k \le \rho$,

$$\|\mathbf{x}_{k+1} - \overline{\mathbf{x}}\|^2 \le \|\mathbf{x}_k - \overline{\mathbf{x}}\|^2 - t_k\rho \; .$$

Now $t_k \to 0$ implies $t_k \le \rho$ for $k \ge K$, say, and summation yields for every $N \ge K$,

$$\rho \sum_{k=K}^{N} t_k \;\; \le \;\; \sum_{k=K}^{N} (\|\mathbf{x}_k - \overline{\mathbf{x}}\|^2 - \|\mathbf{x}_{k+1} - \overline{\mathbf{x}}\|^2) \;\le\; \|\mathbf{x}_K - \overline{\mathbf{x}}\|^2 \; .$$

By our choice of the t_k, however, the left-hand side is unbounded as $N \to \infty$, which establishes the desired contradiction. □

The condition $\sum_{k=0}^{\infty} t_k = \infty$ leads to a very general (and rather weak) convergence result. In particular, the iteration points $\mathbf{x}_k$ need not converge. Even in case they do, the rate of convergence is extremely slow. Linear convergence, for example, is impossible: Indeed, linear convergence with some factor $q < 1$ would imply $\|\mathbf{x}_k - \overline{\mathbf{x}}\| \le q^k \, \|\mathbf{x}_0 - \overline{\mathbf{x}}\|$ and hence

$$t_k = \|\mathbf{x}_{k+1} - \mathbf{x}_k\| \le \|\mathbf{x}_{k+1} - \overline{\mathbf{x}}\| + \|\mathbf{x}_k - \overline{\mathbf{x}}\| \le 2q^k \|\mathbf{x}_0 - \overline{\mathbf{x}}\| \; ,$$

contradicting $\sum_k t_k = \infty$.

Yet, in some cases a more efficient step size control is possible. Suppose, for example, that the minimum value $\overline{f} = f(\overline{\mathbf{x}})$ of f is known in advance (we give such an example below). If $\overline{\mathbf{x}}$ is a *strict minimiser of order* 1, *i.e.*, if there exists some constant $L > 0$ such that

$$f(\mathbf{x}) - f(\overline{\mathbf{x}}) \ge L\|\mathbf{x} - \overline{\mathbf{x}}\| \tag{3.29}$$

holds for all $\mathbf{x} \in \mathbb{R}^n$, then a linearly convergent subgradient method can be designed.

THEOREM 3.14. *If $f : \mathbb{R}^n \to \mathbb{R}$ is convex and (3.29) holds for the minimiser $\overline{\mathbf{x}} \in \mathbb{R}^n$, then the subgradient method with step sizes*

$$t_k = \frac{f(\mathbf{x}_k) - f(\overline{\mathbf{x}})}{\|\mathbf{g}_k\|} \quad \left(\ge L \frac{\|\mathbf{x}_k - \overline{\mathbf{x}}\|}{\|\mathbf{g}_k\|} \right) \tag{3.30}$$

converges linearly to $\overline{\mathbf{x}}$.

Proof. Combining (3.28) with Lemma 3.12, we conclude for our choice for t_k:

$$\|\mathbf{x}_{k+1} - \overline{\mathbf{x}}\|^2 \le \|\mathbf{x}_k - \overline{\mathbf{x}}\|^2 - t_k^2 \le \|\mathbf{x}_k - \overline{\mathbf{x}}\|^2 (1 - L^2/\|\mathbf{g}_k\|^2) \; . \tag{3.31}$$

Hence, if a bound $\|\mathbf{g}_k\| \le c$ exists, linear convergence will follow with convergence factor $C = \sqrt{1 - L^2/c^2} < 1$.

From (3.31) we find in particular $\|\mathbf{x}_k - \overline{\mathbf{x}}\| \le \|\mathbf{x}_0 - \overline{\mathbf{x}}\|$. So $\|\mathbf{x}_k\|$ is bounded and hence the function values are bounded, say $|f(\mathbf{x}_k)| \le M$. Similarly, $\|\mathbf{x}_k - \mathbf{d}_k\|$ is bounded and hence $|f(\mathbf{x}_k - \mathbf{d}_k)| \le M'$. This yields

$$c \;=\; M' + M \;\ge\; f(\mathbf{x}_k - \mathbf{d}_k) - f(\mathbf{x}_k) \;\ge\; \mathbf{g}_k(-\mathbf{d}_k) = \|\mathbf{g}_k\|.$$

□

REMARK. The convergence of the subgradient method can be improved by a modification in the spirit of conjugate directions, which determines $\mathbf{d}_k$ as a combination of the

current $\mathbf{g}_k$ and the previous $\mathbf{d}_{k-1}$. We refer the interested reader to **[2]** (and the references given there), also for a discussion of the choice of the step sizes t_k in practice.

As an example of a non-smooth minimisation problem with *a priori* knowledge of the minimum value $\overline{f}$, we turn to the problem of finding a feasible point of the system

$$f_j(\mathbf{x}) \leq 0 \ , \quad j = 1, \ldots, m \ , \qquad f_j : \mathbb{R}^n \to \mathbb{R} \text{ convex} \ .$$

Obviously, $\overline{\mathbf{x}}$ is feasible if and only if $\overline{\mathbf{x}}$ is a minimiser of the convex function

$$f(\mathbf{x}) := \max\{f_1(\mathbf{x}), \ldots, f_m(\mathbf{x}), 0\} \quad \text{with } f(\overline{\mathbf{x}}) = 0 \ .$$

The Subgradient Method - Exercises.

Exercise 3.10.1. Let $f(\mathbf{x}) = \max\{x_1 + 2x_2, x_1 - 2x_2, -2x_1 - 3\}$. Show that $\overline{\mathbf{x}} = (-1, 0)$ is the unique minimiser of f.
Starting with $\mathbf{x}_0 = (1, 2)$, line-minimisation $(^*)$ leads to $\mathbf{x}_1 = (0, 0)$. Show that $\mathbf{g}_1 = (1, -2)$ is a subgradient of f in $\mathbf{x}_1$ but $-\mathbf{g}_1$ is not a descent direction in the sense

$$f(\mathbf{x}_1 - t\mathbf{g}_1) < f(\mathbf{x}_1) \quad \text{for small } t > 0.$$

Yet note that the distance between $\overline{\mathbf{x}}$ and $\mathbf{x}_2(t) = \mathbf{x}_1 - t\mathbf{g}_1$ decreases for $0 < t < 1/5$.

Exercise 3.10.2. Assuming that the set of minimisers is bounded, show under the assumptions of Theorem 3.13: There exists a subsequence of $(\mathbf{x}_k)$ converging to a minimiser $\overline{\mathbf{x}}$ of f.

Exercise 3.10.3. Show: The subgradient method with $\sum_k t_k = \infty$ cannot lead to any convergence rate

$$\|\mathbf{x}_k - \overline{\mathbf{x}}\| \leq c_0 \cdot k^{-p} \ ,$$

with $p > 1$ (unless it stops after a finite number of steps).

Exercise 3.10.4. Show that $\overline{\mathbf{x}} = (-1, 0)$ in Ex. 3.10.1 is a strict minimiser of order 1.

Exercise 3.10.5. Assume that $f_j : \mathbb{R}^n \to \mathbb{R}$, $j \in J = \{1, \ldots, m\}$ are convex C^1-functions and let

$$f(\mathbf{x}) = \max_{1 \leq j \leq m} f_j(\mathbf{x}) \quad \text{and} \quad J(\overline{\mathbf{x}}) = \{j \in J \mid f(\overline{\mathbf{x}}) = f_j(\overline{\mathbf{x}})\} \ .$$

Show: $\text{conv}\ \{\nabla f_j(\overline{\mathbf{x}}) \mid j \in J(\overline{\mathbf{x}})\} \ \subset \ \partial f(\overline{\mathbf{x}})$ for all $\overline{\mathbf{x}} \in \mathbb{R}^n$.

Part 2

Learning

CHAPTER 4

Introduction to artificial neural networks

A prime example of the nonlinear optimisation in the previous chapter is the training of artificial neural networks in machine learning. In this chapter, we discuss a few basic notions in machine learning and essential building blocks of neural networks. Neural networks come in different varieties, each working in a slightly different manner. Among all of them, however, the fundamental building blocks are the same. Hence, we limit our discussions to the classical neural network structure of multi-layer perceptrons.

4.1. Intuition behind neural networks

Let's start by looking at the following digits:

FIGURE 4.1. Handwritten digits 504192.

Most likely, we can recognise the digits 504192. This seems like an easy task, but it is actually deceptively hard to write an algorithm for such a recognition task. For example, a rule like "a two is a mirrored C combined with a horizontal line" is not sufficient (and even if it were, we'd still need to recognise a C and a horizontal line). The two in the image already does not satisfy this. Yet, a human brain can work this out. If we can find an algorithm that emulates the structure of a brain, then it may become possible to automatically recognise the digits. Many advanced processes, such as detecting manufacturing defects, self-driving cars, and cancer detection, are very similar. Therefore, solving the digit recognition problem gives us tools for tackling many bigger problems out there.

4.1.1. Perceptron. A brain is made up of many neurons. Each neuron receives a signal from other neurons, and if the cumulative signal it receives is greater than a certain threshold, it will send out a signal itself. We can model this using a *perceptron*. Such a perceptron takes several binary variables $x_1, x_2, \ldots$ and spits out a binary variable itself.

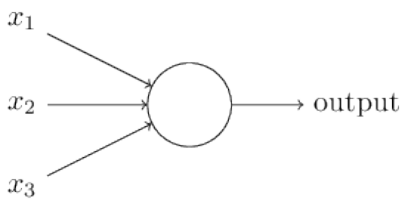


FIGURE 4.2. Example of a perceptron with 3 inputs.

The perceptron does this by weighing the inputs with weights w_i and checking whether the linear combination exceeds a certain threshold, i.e.,

$$\text{output} = \begin{cases} 0 & \sum_i \mathbf{w}_i \mathbf{x}_i < \text{threshold} \\ 1 & \sum_i \mathbf{w}_i \mathbf{x}_i \geq \text{threshold} \end{cases} \tag{4.1}$$

A nice example of how you can imagine this is the decision whether you want to go to a festival. Relevant factors for going to a festival are

(1) whether one of your friends goes,
(2) whether the festival is awesome,
(3) whether the weather is good,
(4) whether you have enough money to spare.

You want to go when either the weather is good and your friend goes or the festival is awesome, but these don't matter if you don't have enough money. This can be modelled using a perceptron by setting $w_{\text{friend}} = 2$, $w_{\text{awesome}} = 3$, $w_{\text{weather}} = 1$, $w_{\text{money}} = 10$ and threshold $= 13$.

You can change the decision-making process by changing the weight or the threshold. If you'd lower the threshold to 6, you would also go if you didn't have the money but the weather is good, the festival is awesome and one of your friends joins you. If you changed w_{friend} to $w_{\text{friend}} = 3$, the weather no longer needs to be good for you to go.

If you know when you want to go or not (so the output of the perceptron given a certain input), then you can consider finding the various weights a function fitting problem. In this case, you can always find w's to represent your decision-making, so you can find an optimal function. There are also multiple sets of weights that make sure the perceptron gives the same outputs. Thus, there is no unique optimal solution to this function fitting problem. We will use this idea of treating the process of finding the weights as a function fitting problem shortly later in these lecture notes.

4.1.2. Multi-layer Perceptron. In your brain the neurons pass information along to each other, so to more accurately model the brain we should stick several perceptrons together.

Now, the decisions from the first column of perceptrons are used in the second column of perceptrons to produce an answer. Going back to our festival example, we can now imagine that some of the inputs are the weather prediction from Buienradar, the time of year, and how cloudy it looks today. A perceptron in

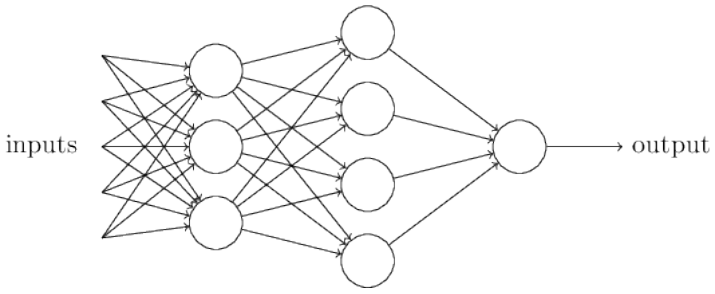


FIGURE 4.3. Example of a multi-layer perceptron.

the first column could use those to decide whether the weather is nice, which a perceptron in the second column could use to decide whether you'd want to go.

4.1.3. Fitting digits. We could use a multi-layer perceptron to decide for us what digit we see on the picture. So far, each perceptron we discussed only gave a yes or no, i.e. a binary value. We have 10 digits, so with one yes or no we cannot decide what digit we are looking at. One option is to create 10 multi-layer perceptrons which each decide on whether they think it is a particular digit. So, the first decides is this a 0, the second decides is this a 1, etc. Several digits make use of straight lines. Hence, the perceptrons for these digits have in their layers shared subdecisions. To not have to repeat the same structures in several of the perceptrons, we could also make one bigger multi-layer perceptron that gives us 10 outputs instead of 1. Once we have decided on how big a multi-layer perceptron we want to use (e.g. the one shown in fig. 4.4), we only need to find the weights. Finding those is again a function fitting problem.

4.1.4. Sigmoids. In the optimisation part of this lecture notes, you learned about gradient descent and how this can help solve optimisation problem. Function fitting problems are one kind of optimisation problem. In gradient descent, you nudge your parameters in such a way that the function gets closer to the optimal step. When doing this in our multi-layer perceptron, we run into a problem. If we slightly change a weight, the perceptron might suddenly decide the digit is no longer a 1 but a 0. It can also be that nothing changes at all. Because the multi-layer perceptron has several layers, predicting these jumps is hard to impossible. It would be better to have that a small change to a weight, changes each output a little. We can do this by smoothing out the outputs.

Using the step function (or the hard-threshold function)

$$\text{step}(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \tag{4.2}$$

we can write (4.1) as

$$\text{output} = \text{step}(\sum_i \mathbf{w}_i \mathbf{x}_i - \text{threshold}). \tag{4.3}$$

When we smooth out the step function, we get a sigmoidal function (or S-shaped function).

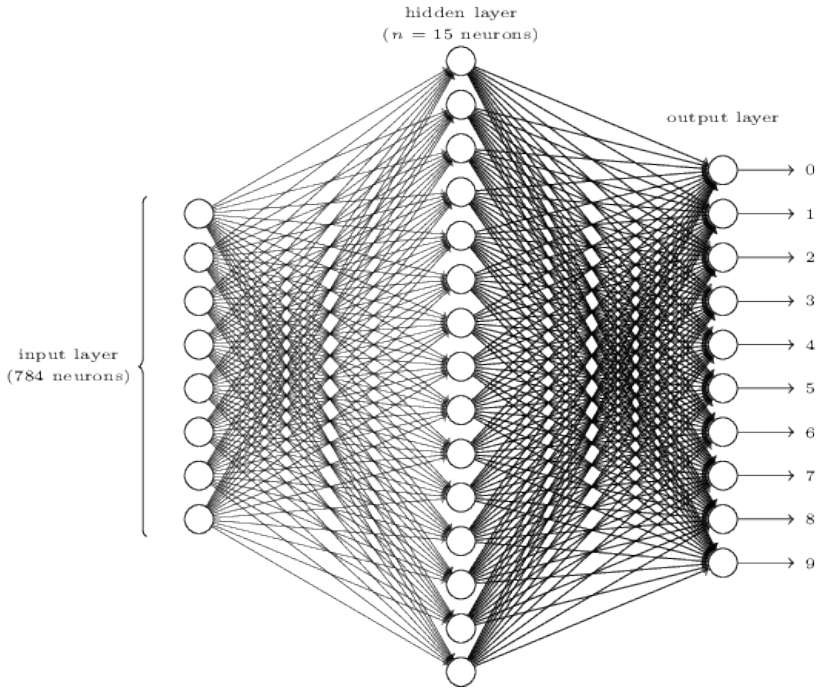


FIGURE 4.4. Example of a multi-layer perceptron with multiple outputs. Each input represents one pixel of the image. It has 784 inputs, since the digits are images of 28 by 28 pixels. Output i represents whether the network thinks it is digit i.

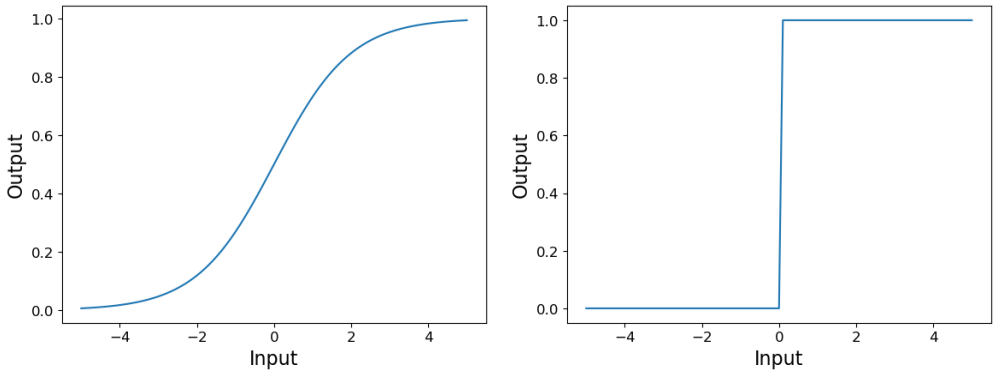


FIGURE 4.5. Left: the logistic function, an example of a sigmoid. Right: the step function.

One of these sigmoidal functions is the logistic function

$$\text{logi}(x) = \frac{e^x}{1+e^x} = \frac{1}{1+e^{-x}}. \tag{4.4}$$

We can now replace the way the perceptron computes outputs from (4.3) to

$$\text{output} = \text{logi}(\sum_i \mathbf{w}_i \mathbf{x}_i - \text{threshold}), \tag{4.5}$$

or rather

$$\text{output} = \text{logi}(\sum_i \mathbf{w}_i \mathbf{x}_i + b), \tag{4.6}$$

since the threshold is now more an offset, here represented by b (short for bias), than a threshold. With this, we will have that a small change in a weight will lead to a small change in the output of the model.

4.1.5. SoftMax. We have changed the way each neuron in our model works from a step function to a sigmoid. This means that the output can become any number between 0 and 1. This is good for tweaking the weight in the model, but the model can give us an output of $[0, 0, 0, 1, 0.8, 0.3, 1, 0, 0, 0]$. The ones in this output can indicate that the handwriting is just rubbish and the digits could have been 3 or just as well a 6. But, what to make of the 0.8 and 0.3? This is no longer a yes or a no. Rounding the output to the nearest integer gives us $[0, 0, 0, 1, 1, 0, 1, 0, 0]$. Does this mean two of the outputs are in need of tuning, or does the digit resemble three different numbers? When we show someone a digit really resembling three different numbers and ask what that digit is supposed to be, a natural response would be "Probably X, but it could also be Y". We can apply this kind of thinking to our model. We do this by mapping the output from our model onto a probability distribution. This is typical done by replacing the last logi of the model by a SoftMax function, i.e. if the model gives as output a vector x, then $\text{SoftMax}(x)$ where

$$\text{SoftMax}(\mathbf{x})_i = \frac{e^{\mathbf{x}_i}}{\sum_i e^{\mathbf{x}_i}} \tag{4.7}$$

turns x into a probability distribution. This means that the components of $\text{SoftMax}(\mathbf{x})$ add to 1, i.e. $\sum_i \text{SoftMax}(\mathbf{x})_i = 1$, and that $0 \leq \text{SoftMax}(\mathbf{x})_i \leq 1$.

This shows that we can piece together a network inspired by the human brain that can indeed solve the handwritten digit problem. The only part that we are missing at this point is an algorithm for finding the proper weights in our model. Such algorithms are based on gradient descent. To discuss those in more detail, we need to write the network in some structured form that allows us to see what for example gradients do.

Exercises.

Exercise 4.1.1. In the festival example, how should the perceptron be adapted if the model includes whether someone you really don't want to meet goes to the festival?

Exercise 4.1.2. Pi day is coming up and you'd like to bake a pie for the Pi day celebrations. You are unsure whether you want to make your favourite or the other recipe you found online. List 5 criteria (like X makes pie Y) and create a

perceptron for it. An output of 1 means making the other recipe, whilst an output of 0 means making your favourite.

Exercise 4.1.3. The annual Taipan tournament is coming up. Create a perceptron that predicts whether someone will go.

Exercise 4.1.4. It's Monday. Create a perceptron that predicts whether you will go home to (one of) your parents the coming weekend.

Exercise 4.1.5. You are sitting in the Abacus room. Someone changes the music. Create a perceptron that predicts whether you will change the music again (e.g. back to what it was).

Exercise 4.1.6. In section 4.1.2 we discussed an extension to the festival example in section 4.1.1. Write down the perceptron described and adapt the original perceptron.

Exercise 4.1.7. In the festival example, what weight and biases do you need to choose to model this using a neural network with logi as activation function, if an output of more than 0.6 means that you want to go and an output of below 0.4 means that you don't want to go?

Exercise 4.1.8. It's almost the lunch break. Create three perceptrons: one that predicts whether you play Klaverjas, one that predicts whether you play Hearts, and one that predicts whether you play Taipan.

Several of these can predict that you are going to play that card game. Create a perceptron that predicts what you are going to play based on the output of the previous perceptrons.

Exercise 4.1.9. Reconsider the problem from the previous exercise. Replace the fourth perceptron with a SoftMax. Compare for several situations the results with the multi-layer perceptron from the previous question. Which is more accurate? Why?

4.2. Neural network architecture

Consider two sets $\mathbb{X} \subseteq \mathbb{R}^{d_{in}}$ and $\mathbb{Y} \subseteq \mathbb{R}^{d_{out}}$ with $n, m \in \mathbb{N}$. $\mathbb{X}$ represents the values that we put into our model and $\mathbb{Y}$ represents the possible outcomes of our model. It will be convenient to write the change from one step to the next in vector notation instead of per neuron. If $d_{in} = 2$ and the second layer has 2 neurons, then the outputs after that layer can be written as

$$\text{logi}\left(\begin{pmatrix} \mathbf{w}_{1,1} & \mathbf{w}_{2,1} \\ \mathbf{w}_{1,2} & \mathbf{w}_{2,2} \end{pmatrix}\begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} + \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}\right) = \begin{pmatrix} \text{logi}(\sum_{i=1}^{d_{in}} \mathbf{w}_{i,1}\mathbf{x}_i + \mathbf{b}_1) \\ \text{logi}(\sum_{i=1}^{d_{in}} \mathbf{w}_{i,2}\mathbf{x}_i + \mathbf{b}_2) \end{pmatrix} \tag{4.8}$$

Hence, in general we can write the change from layer to layer as a logi with matrix multiplication and vector addition inside. Although, we also used logi in the example, but there are many sigmoid functions or other non-sigmoid function that are being used. We can use a placeholder σ for such a function, which we call an *activation function*. An arbitrary neural network $f_{\boldsymbol{\theta}} : \mathbb{X} \to \mathbb{Y}$ will have several layers. Subscripts are needed to index the vectors and matrices, so we will indicate

the layer with a superscript ℓ. Doing so allows us to write an arbitrary neural network $f_{\boldsymbol{\theta}} : \mathbb{X} \to \mathbb{Y}$ as

$$z^0(x) = \mathbf{x} \tag{4.9a}$$

$$z^\ell(x) = \sigma^\ell(\mathbf{W}^\ell z^{\ell-1}(\mathbf{x}) + \mathbf{b}^\ell) \tag{4.9b}$$

$$f_{\boldsymbol{\theta}}(x) = z^L(\mathbf{x}) \tag{4.9c}$$

with *weights* $\mathbf{W}^\ell \in \mathbb{R}^{d^\ell \times d^{\ell-1}}$ and *biases* $\mathbf{b}^\ell \in \mathbb{R}^{d^\ell}$. The dimensions $d^\ell \in \mathbb{N}$ with $d^0 = d_{in}$ and $d^L = d_{out}$ are called *widths*, and with $\ell \in \{1, ..., L\}$ we refer to which layer of the neural network we are talking about. The weights and biases combined are the *parameters* $\boldsymbol{\theta} := \{\mathbf{W}^1, \mathbf{b}^1, \mathbf{W}^2, ...\}$ of the network, so that we have a brief and convenient way of addressing both the weights and biases of each layer. The term

$$h^\ell(x) = \mathbf{W}^\ell z^{\ell-1}(\mathbf{x}) + \mathbf{b}^\ell \tag{4.10}$$

is called the *pre-activation*. z^ℓ are similarly called *post-activations*. Last,

$$a^\ell(\mathbf{q}) = \mathbf{W}^\ell \mathbf{q} + \mathbf{b}^\ell \tag{4.11}$$

is the affine step of layer ℓ.

Example 4.1. *Let* $\sigma = \text{logi}$ *and*

$$\mathbf{W}^1 = \begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix},\ \mathbf{b}^1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix},\ \mathbf{W}^2 = \begin{pmatrix} 1 & -1 \end{pmatrix},\ b^2 = \pi. \tag{4.12}$$

To compute the associated neural network f*, we start with the pre-activation function* h^1*. This is given by*

$$h^1(\mathbf{x}) = \mathbf{W}^1\mathbf{x} + \mathbf{b}^1 = \begin{pmatrix} 1 & 2 \\ -4 & 5 \end{pmatrix}\begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} + \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1 + 2\mathbf{x}_2 + 1 \\ -4\mathbf{x}_1 + 5\mathbf{x}_2 + 2 \end{pmatrix}. \tag{4.13}$$

To go from h^1 *to* z^1 *we apply the activation function* σ *component-wise, i.e.*

$$z^1(\mathbf{x}) = \text{logi}(h^1(\mathbf{x})) = \begin{pmatrix} \text{logi}(\mathbf{x}_1 + 2\mathbf{x}_2 + 1) \\ \text{logi}(-4\mathbf{x}_1 + 5\mathbf{x}_2 + 2) \end{pmatrix} \tag{4.14}$$

Now, we can move to the next layer. This is the last layer, so we can now compute f*. This is given by*

$$\begin{aligned} f_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{W}^2 z^1(\mathbf{x}) + b^2 &= \begin{pmatrix} 1 & -1 \end{pmatrix}\begin{pmatrix} \text{logi}(\mathbf{x}_1 + 2\mathbf{x}_2 + 1) \\ \text{logi}(-4\mathbf{x}_1 + 5\mathbf{x}_2 + 2) \end{pmatrix} + \pi \\ &= \text{logi}(\mathbf{x}_1 + 2\mathbf{x}_2 + 1) - \text{logi}(-4\mathbf{x}_1 + 5\mathbf{x}_2 + 2) + \pi. \end{aligned} \tag{4.15}$$

Remark. *The affine step in eq.* (4.11) *is sometimes written as*

$$a^\ell(\mathbf{q}) = \begin{pmatrix} \mathbf{W}^\ell & \mathbf{b}^\ell \end{pmatrix}\begin{pmatrix} \mathbf{q} \\ \mathbf{1} \end{pmatrix} =: \tilde{\mathbf{W}}^\ell \tilde{\mathbf{q}} \tag{4.16}$$

in proofs to simplify expressions. Using this style in the example above, we would write

$$h^1(\mathbf{x}) = \tilde{\mathbf{W}}^1\tilde{\mathbf{x}} = \begin{pmatrix} 1 & 2 & 1 \\ -4 & 5 & 2 \end{pmatrix}\begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1 + 2\mathbf{x}_2 + 1 \\ -4\mathbf{x}_1 + 5\mathbf{x}_2 + 2 \end{pmatrix} \tag{4.17}$$

instead of eq. (4.13)*. We recommend you never do this. This gives unnecessary overhead numerically, and can lead you to make false inferences about the underlying geometry in proofs.*

Neural network are used to approximate functions. We do this in two tasks: regression and classification. In regression, we fit a function to data. In classification, we assign a *class* (cat, dog, car, number 7, ...) or a probability distribution over classes to data. We measure how well a neural network does this using a loss function $\mathcal{L}$. The neural network that performs the best minimises the loss function $\mathcal{L}$. The process of finding the parameters for which this happens is called *training*, and we say the neural network *learns* the right parameters $\boldsymbol{\theta}$. Let us first discuss the bits of the neural network itself, before discussing this optimisation problem in more detail.

Exercise 4.2.1. Let $\sigma^\ell = \text{logi}$, $d_{in} = 2$, $d_{out} = 1$, and

$$\mathbf{W}^1 = \begin{pmatrix} -3 & 9 \\ 1 & 4 \end{pmatrix}, \ \mathbf{b}^1 = \begin{pmatrix} 2 \\ -1 \end{pmatrix}, \ \mathbf{W}^2 = \begin{pmatrix} -7 & -5 \\ 5 & 2 \\ 8 & 9 \end{pmatrix},$$

$$\mathbf{b}^2 = \begin{pmatrix} 3 \\ -6 \\ 8 \end{pmatrix}, \ \mathbf{W}^3 = \begin{pmatrix} 1 & -1 & 0 \end{pmatrix}, \ b^3 = -2.$$

What is $f_{\boldsymbol{\theta}}$?

Exercise 4.2.2. Let $\sigma^\ell = \text{logi}$, $d_{in} = 2$, $d_{out} = 2$, and

$$\mathbf{W}^1 = \begin{pmatrix} 1 & 4 \\ -9 & -5 \\ 3 & -1 \\ 5 & 1 \end{pmatrix}, \ \mathbf{b}^1 = \begin{pmatrix} -4 \\ 5 \\ 8 \\ 1 \end{pmatrix}, \ \mathbf{W}^2 = \begin{pmatrix} 8 & 2 & 2 & 2 \\ 3 & -2 & -1 & 2 \end{pmatrix}, \ \mathbf{b}^2 \ = \begin{pmatrix} -8 & 0 \end{pmatrix}.$$

What is $f_{\boldsymbol{\theta}}$?

Exercise 4.2.3. Recall the perceptrons from Exercise 4.1.6. Write them in vector notation.

Exercise 4.2.4. Recall the perceptrons from Exercise 4.1.8. Write them in vector notation.

Exercise 4.2.5. Recall the perceptrons from Exercise 4.1.9. Write them in vector notation.

Exercise 4.2.6. Let $\sigma(x) = ReLU(x) := \max(0, x)$. If $d_{in} = 2$ and $d_{out} = 1$, what weights and biases should we choose to get the neural network

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \max(\mathbf{x}_1, \mathbf{x}_2)? \tag{4.18}$$

Exercise 4.2.7. Let $\sigma(x) = ReLU(x) := \max(0, x)$. If $d_{in} = 1$ and $d_{out} = 1$, what is the smallest number of weights and biases to get the neural network

$$f_{\boldsymbol{\theta}}(x) = \begin{cases} x & 0 \leq x \leq 0.5 \\ 1 - x & 0.5 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.19}$$

representing a triangle wavelet?

Exercise 4.2.8. Suppose you are using a neural network to detect whether a pedestrian wants to cross the road. Is this a regression or classification task?

4.2.1. Activation function. In the definition of the neural network from (4.9), we see that in the neural network alternates an affine step and then apply an activation function, i.e. we first compute a pre-activation h^ℓ, then apply the activation function σ to that to get z^ℓ and subsequently compute the next pre-activation $h^{\ell+1}$. The choice of activation function has a large influence on what the neural network can do. If you choose it too simple, you can only approximate simple functions.

Lemma 4.2. *Let $\sigma(x) = x$, then $f_{\boldsymbol{\theta}}(\mathbf{x})$ is an affine function, i.e. there exists an $\mathbf{A}$ and a $\mathbf{B}$ such that*

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{B}. \tag{4.20}$$

If you choose it very complex, then any numerical error could give us a very different function. In practice, we stick to a select few activation functions. Since we want to use gradient based methods, we need to be able to take a derivative of the activation function and want that derivative to be non-zero. The multilayer perceptron uses step functions, which do not have a non-zero derivative. By smoothing the step function, we get the previously mentioned logi. By integrating the step function, we get

$$\mathrm{ReLU}(x) = \max(0, x). \tag{4.21}$$

The rule of thumb is to use these two inside the network, i.e. for σ^ℓ with $\ell \neq L$. When you fit a data in a regression task, it is common to use the identity $\sigma^L(x) = x$ for the final layer. When you assign classes to inputs in a classification task, it is common to use $\sigma^L = \mathrm{SoftMax}$ like we did in section 4.1.5. These are listed together with their derivatives in table 1.

Remark. *For the interested reader: There is no hard mathematical theory for why some activation functions work better than others, but reasoning and empirical evidence has shown that several properties are desirable.*

1. *σ **is easy to compute.** To evaluate the neuron network, we need to apply σ point-wise $L-1$ times. Hence, if σ is hard to compute, then $f_{\boldsymbol{\theta}}$ becomes hard too.*
2. *σ' **is easy to compute.** The algorithms that are used during the training process require us to compute the derivative of the activation function σ' many times. Hence, if σ' is hard to compute, then finding the best parameters $\boldsymbol{\theta}$ for the network $f_{\boldsymbol{\theta}}$ is hard too.*
3. *σ **is monotone.** During the training process $\mathbf{W}^\ell$ and $\mathbf{b}^\ell$ will be updated. If $z^\ell(\mathbf{x})$ need to become bigger and σ is monotone, then bigger $\mathbf{W}^\ell$ and $\mathbf{b}^\ell$ achieve this. If σ is not monotone and the algorithm overshoots how much bigger $\mathbf{W}^\ell$ and $\mathbf{b}^\ell$, there is a possibility that it might actually make $z^\ell(\mathbf{x})$ smaller.*
4. *$\sigma(0) = 0$. If $\sigma(0) = c$, then for $h^\ell(\mathbf{x}) = \mathbf{0}$ we have $z^\ell(\mathbf{x}) = c\mathbf{1}$. We can adapt $\mathbf{b}^\ell$ to compensate for $c\mathbf{1}$, so w.l.o.g. we rather have $c = 0$.*

Name	σ	σ'
Identity	x	1
Step	$H(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$	0
ReLU	$\max(0, x)$	$H(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$
Logistic	$\frac{1}{1+e^{-x}}$	$\sigma(x)(1 - \sigma(x))$
SoftMax	$\sigma(x)_i = \frac{e^{x_i}}{\sum_i e^{x_i}}$	$\frac{\partial \sigma(x)_i}{\partial x_k} = \sigma(x)_k(\delta_{ik} - \sigma(x)_i)$

TABLE 1. List of used activation function with some of their properties. SoftMax is an exception to the rule that activation functions are component-wise applied, and δ is the Dirac delta.

(5) σ ***is not a polynomial.*** *This has to do with which functions a neural network can approximate. If σ is not a polynomial, then you can approximate any continuous function up to arbitrary accuracy using a neural network when you allow the widths d^ℓ to become arbitrarily large, i.e. neural networks are dense in the continuous functions. This is called the universal approximation theory. You don't want to exclude many functions you can fit when you don't need to, so better avoid polynomial activation functions σ.*

Exercises.

Exercise 4.2.9. The SoftMax activation function is given by $(\sigma(x))_i = \frac{e^{x_i}}{\sum_{i=1}^n e^{x_i}}$. Take an $n \in \mathbb{N}$ and compute $\sigma(x)$ for several $x \in \mathbb{X}$. What function does the SoftMax approximate?

Exercise 4.2.10. SoftMax turns the output into a probability distribution over classes. Give an example of another function that turns a vector in a probability distribution.

Exercise 4.2.11. The SoftPlus activation function is given by $\sigma(x) = \log(1 + e^x)$, whereas ReLU is given by $\sigma(x) = \max(0, x)$. Sketch their graphs and compare the two activation functions.

Exercise 4.2.12. Create a neural network with logi as activation function. In classification tasks we replace the last logi with a SoftMax. What happens if we do not replace the last logi (so we keep it), but apply the SoftMax afterwards?

4.2.2. Weights and biases. Neural networks alternate computing h^ℓ, an affine step, and computing z^ℓ, an application of an activation function. In this previous section, we discussed the activation function. In this section, we will focus on the affine step. For that, recall that the pre-activation of layer ℓ was given by

$$h^\ell(\mathbf{x}) = \mathbf{W}^\ell z^{\ell-1}(\mathbf{x}) + \mathbf{b}^\ell. \tag{4.22}$$

This is a map $h^\ell : \mathbb{R}^{d^{\ell-1}} \to \mathbb{R}^{d^\ell}$. The sizes of $\mathbf{W}^\ell$ and $\mathbf{b}^\ell$ are determined by the sequence of widths $(d^\ell)_{\ell=1}^L$. It is possible to place additional constraints on the weights $\mathbf{W}^\ell$ or biases $\mathbf{b}^\ell$, like $\mathbf{W}^\ell$ is tridiagonal or skew-symmetric. In this course, we avoid doing so. The combination of the shapes of $\mathbf{W}^\ell$ and $\mathbf{b}^\ell$ combined with any potential constraints on them is called the *architecture* of the neural network. Neural networks are classified based on their architecture. Commonly used network architecture classification are depicted in fig. 4.6 through examples.

They are called *shallow* if $L = 2$ and *deep* otherwise. If each $d^\ell \gg L$, the network is *wide*. If each $d^\ell \ll L$, then the network is *narrow*. If one d^ℓ with $\ell \notin \{1, L\}$ is way smaller than the (most) other d^ℓ, then the network has a *bottleneck*. If the architecture has repeating parts, e.g. the network has $d^{k-1} = c_1$, $d^k = c_2$, $d^{k+1} = c_3$ for several values of k, then we call each of those repeating parts a *block*.

Exercises.

Exercise 4.2.13. Sketch the graph for the network $f_{\boldsymbol{\theta}}$ from Exercise 4.2.1.

Exercise 4.2.14. Sketch the graph for the network $f_{\boldsymbol{\theta}}$ from Exercise 4.2.2.

Exercise 4.2.15. Which network $f_{\boldsymbol{\theta}}$ is represented in fig. 4.7?

Exercise 4.2.16. Which network $f_{\boldsymbol{\theta}}$ is represented in fig. 4.8?

Exercise 4.2.17. Which network $f_{\boldsymbol{\theta}}$ is represented in fig. 4.9?

Exercise 4.2.18. How to classify the three networks shown in fig. 4.10?

4.3. Training

Now that we have the structured way of writing down, we can address the question of how to find the parameters of the neural network. To find the best parameters $\boldsymbol{\theta}$, so the weights $\mathbf{W}^\ell$ and biases $\mathbf{b}^\ell$, we *train* the network (or let the network *learn* them). This is a very anthropomorphized way of saying that we solve an optimisation problem of the form

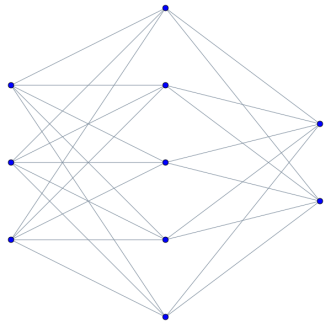
$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}), \tag{4.23}$$

where $\mathcal{L}$ is a *loss function*. The loss function assigns a score to how well your neural network performs. The most used loss function, and the loss that we will be using in this course, is the mean squared error (MSE) loss. When you have a dataset $(\mathbf{x}_n, \mathbf{y}_n)_{n=1}^N$ with $\mathbf{x}_n \in \mathbb{X}$ and $\mathbf{y}_n \in \mathbb{Y}$, then this is given by

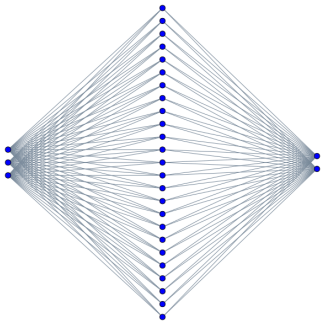
$$\mathcal{L}_{MSE}(\boldsymbol{\theta}) = \frac{1}{2} \sum_{n=1}^{N} ||\mathbf{y}_n - f_{\boldsymbol{\theta}}(x_n)||_2^2. \tag{4.24}$$

When we do a gradient step, we need to compute $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{MSE}(\boldsymbol{\theta})$. The chain rule tells us that

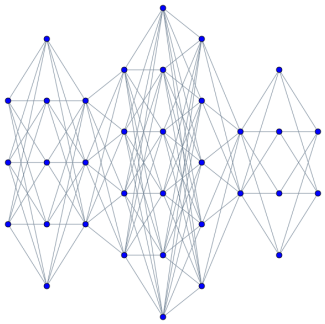
$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{MSE}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{x}_n)^\intercal (f_{\boldsymbol{\theta}}(\mathbf{x}_n) - \mathbf{y}_n). \tag{4.25}$$



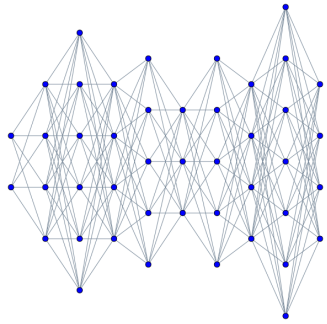
(A) Shallow



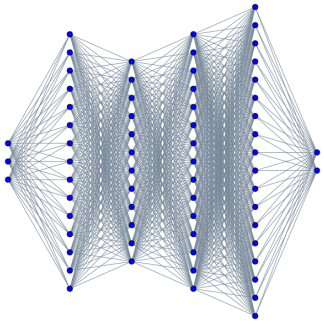
(B) Shallow and wide



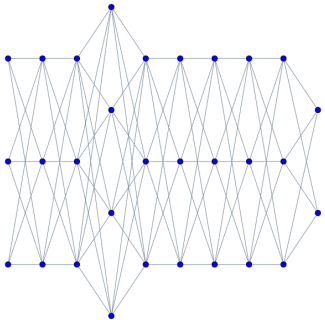
(C) Deep



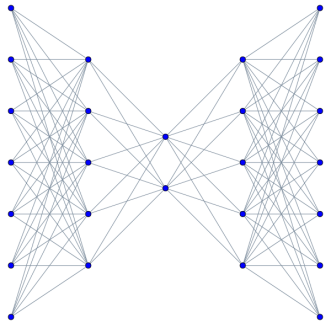
(D) Deep



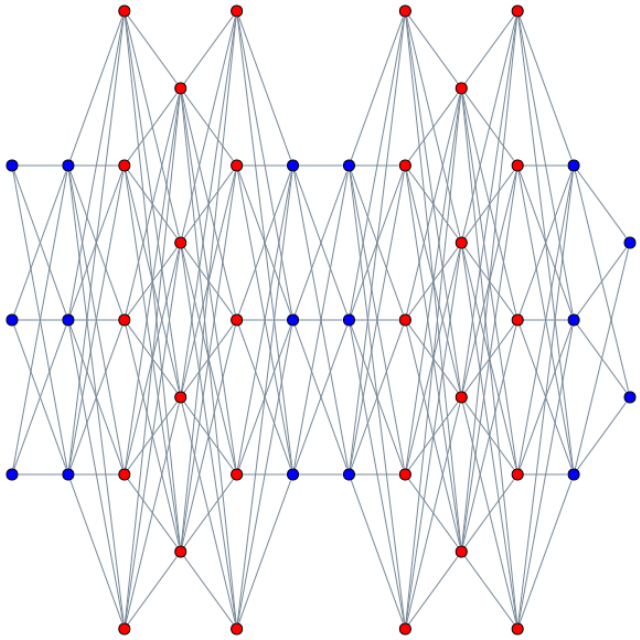
(E) Deep and Wide



(F) Deep and narrow



(G) Bottlenecked



(H) Red parts are blocks

FIGURE 4.6. Examples of networks with their architecture-based classification.

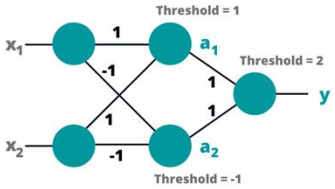


FIGURE 4.7. Caption

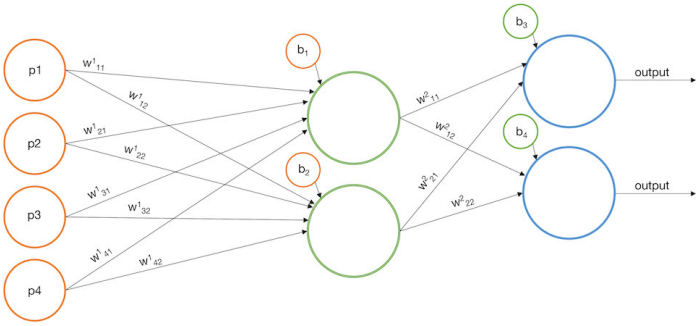


FIGURE 4.8. Caption

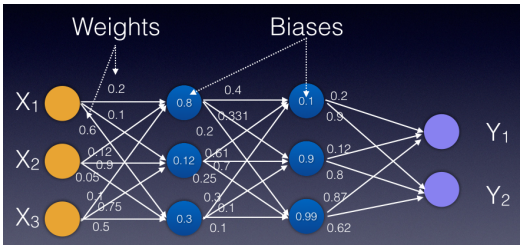


FIGURE 4.9. Caption

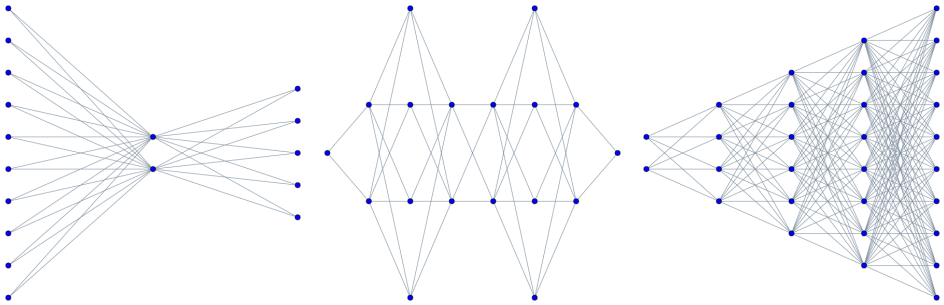


FIGURE 4.10. Three neural networks

The gradient $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}$ is not something we want to compute with finite difference. In the rest of this section, we will discuss how to solve this optimisation problem in practice.

4.3.1. Forward and backward differentiation. The first thing that we need to discuss is how to efficiently compute $\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}$. To that end, let us recall some common rules for gradients. Let f, g be two functions.

(1) **Sum rule**: $\frac{\partial}{\partial x}(f(x) + g(x)) = \frac{\partial}{\partial x} f(x) + \frac{\partial}{\partial x} g(x)$
(2) **Product rule**: $\frac{\partial}{\partial x}(f(x)g(x)) = g(x)\frac{\partial}{\partial x} f(x) + f(x)\frac{\partial}{\partial x} g(x)$
(3) **Chain rule**: $\frac{\partial}{\partial x} f(g(x)) = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$

Also recall that if $f : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$, then

$$\nabla_{\mathbf{x}} f = \frac{\partial f}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_1}{\partial \mathbf{x}_{d_{in}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{d_{out}}}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_{d_{out}}}{\partial \mathbf{x}_{d_{in}}}. \end{pmatrix} \tag{4.26}$$

Finally, recall that

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = z^L(\mathbf{x}) = (\sigma^L \circ h^\ell)(\mathbf{x}) = (\sigma^L \circ a^L \circ z^{L-1})(\mathbf{x}) = \cdots = (\sigma^L \circ a^L \ldots \sigma^1 \circ a^1)(\mathbf{x}). \tag{4.27}$$

We see that $f_{\boldsymbol{\theta}}$ is a composition of L activation functions and L affine steps. This strongly suggest that if we want to take the derivate with respect to some parameter that we want to use the chain rule many times over. Let's go over three examples to see what kind of expressions we get.

Example 4.3. *Suppose we have the function*

$$f(x) = \prod_{i=1}^{n} \sin(x) \tag{4.28}$$

with $x \in \mathbb{R}$. Clearly, $f(x) = \sin(x)^n$, and $\frac{df}{dx}(x) = n\sin(x)^{n-1}\cos(x)$.

Example 4.4. *Suppose we have the function*

$$f(x) = \left(\bigcirc_{\ell=1}^{L} \sin(\cdot) \right)(x) = \bigcirc_{\ell=1}^{L} \sin(x) \tag{4.29}$$

with $x \in \mathbb{R}$. For this function, we have

(1) If $L = 2$, then $f(x) = \sin(\sin(x))$, and $\frac{df}{dx}(x) = \cos(x)\cos(\sin(x))$.
(2) If $L = 3$, then $f(x) = \sin(\sin(\sin(x)))$, and $\frac{df}{dx}(x) = \cos(x)\cos(\sin(x))\cos(\sin(\sin(x)))$.
(3) If $L = 4$, then $\frac{df}{dx}(x) = \cos(x)\cos(\sin(x))\cos(\sin(\sin(x)))\cos(\sin(\sin(\sin(x))))$.
(4) In general, $\frac{df}{dx}(x) = \prod_{\ell=1}^{L} \cos(\bigcirc_{i=1}^{\ell-1} \sin(x))$.

Example 4.5. *Suppose we have the function*

$$f(x) = \left(\bigcirc_{\ell=1}^{L} \operatorname{logi}(w_\ell \cdot + b_\ell) \right)(x) \tag{4.30}$$

with $x \in \mathbb{R}$. For this function, we have

(1) If $L = 1$, then $f(x) = \operatorname{logi}(w_1 x + b_1)$, and $\frac{df}{dx}(x) = w_1 \frac{d\operatorname{logi}}{dx}(w_1 x + b_1)$.
(2) If $L = 2$, then $f(x) = \operatorname{logi}(w_2(\operatorname{logi}(w_1 x + b_1) + b_2)$,
and $\frac{df}{dx}(x) = w_1 \frac{d\operatorname{logi}}{dx}(\operatorname{logi}(b_1 + w_1 z))\frac{d\operatorname{logi}}{dx}(b_1 + w_1 z)$.

(3) *If* $L = 3$, *then* $\frac{df}{dx}(x) = w_1 w_2 \frac{d\,\mathrm{logi}}{dx}\left(\mathrm{logi}\left(w_1 f\left(b_2 + w_2 z\right) + b_1\right)\right)$
$\frac{d\,\mathrm{logi}}{dx}\left(w_1 \,\mathrm{logi}\left(b_2 + w_2 z\right) + b_1\right) \frac{d\,\mathrm{logi}}{dx}\left(b_2 + w_2 z\right)$.

In the first example, we have a simple product of sines. The gradient of this is easy to compute. In the second example, we have replaced the product with a composition symbol. This means that we have to compute the gradient using the chain rule. Since the thing we compose is still a sine, we can write an explicit expression for the gradient. In compact form, this may seem still okay. But, when your write it out, the gradient becomes a large expression quite fast. In the third example, we used a width 1 neural network. For this one, we already needed to break the line at $L = 3$. In practice, the width won't be just 1. This means there will be partial gradients in there too. We will also be taking the gradient with respect to the parameters and not the input of the network. Combined with the fact that L will be large, we expect to get a humongous expression for the gradient of a neural network with respect to the parameters. Even if this function could be stored in a computer, we wouldn't want to evaluate it due to the sheer computing power needed. What we need is automatic differentiation.

Remark. *For the interested reader: Machine learning has been around since at least the 50s. A form of automatic differentiation was discovered in the 60s. This was not known (or its relevance not recognised) to the machine learning community. In the early 70s, criticism of artificial intelligence grew. A few years later, funding for machine learning collapsed. The period that followed has been coined the "AI Winter". In the 80s, the modern way of automatic differentiation was discovered and, together with universal approximation theorem and some promising sounding architectures, it ended the winter. In the 90s, these new architectures had failed to live up to their promise, so funding was again hard to come by. In the 00s, automatic differentiation for GPUs became available. This allowed for training much larger networks. One of these networks was AlexNet, which beat the runner-up in the ImageNet competition of 2012 by more than 10 percentage points. This hailed the start of the current era of AI, sometimes referred to as the "AI Summer".*

What we see in the examples is that we end up with a long list of products of derivatives. There are two ways of solving this: forward and backward. These refer to the order in which we compute the gradients. We can write the gradient in example 4.5 as

$$\frac{df}{dx} = \frac{dz^L}{dh^L}\frac{dh^L}{dz^{L-1}}\frac{dz^{L-1}}{dh^{L-1}}\frac{dh^{L-1}}{dz^{L-2}}\cdots\frac{dz^3}{dh^2}\frac{dh^2}{dz^2}\frac{dz^2}{dh^1}\frac{dh^1}{dz^0}\frac{dz^0}{dx}, \tag{4.31}$$

where we suppressed the inputs to the derivatives in the notation. When we want to compute this we can place brackets like this,

$$\frac{df}{dx} = \left(\frac{dz^L}{dh^L}\left(\frac{dh^L}{dz^{L-1}}\left(\frac{dz^{L-1}}{dh^{L-1}}\left(\frac{dh^{L-1}}{dz^{L-2}}\left(\cdots\left(\frac{dz^3}{dh^2}\left(\frac{dh^2}{dz^2}\left(\frac{dz^2}{dh^1}\left(\frac{dh^1}{dz^0}\left(\frac{dz^0}{dx}\right)\cdots\right)\right.\right.\right.\right.\right.\right.\right.\right. \tag{4.32}$$

or like this,

$$\frac{df}{dx} = \left(\cdots\left(\frac{dz^L}{dh^L}\right)\frac{dh^L}{dz^{L-1}}\right)\frac{dz^{L-1}}{dh^{L-1}}\right)\frac{dh^{L-1}}{dz^{L-2}}\right)\cdots\right)\frac{dz^3}{dh^2}\right)\frac{dh^2}{dz^2}\right)\frac{dz^2}{dh^1}\right)\frac{dh^1}{dx}\right)\frac{dz^0}{dx}\right). \tag{4.33}$$

In eq. (4.32), we start at the back and move to the front, so that's called forward differentiation. In eq. (4.33), we start at the front and move to the back, so that's called backward differentiation (or reverse mode differentiation). We have shown both ways for the gradient of example 4.5 in fig. 4.11.

In eq. (4.31), only have products following from the chain rule. However, sometimes you have to compute a product rule or a quotient rule instead. We can express a function as a computational graph, where each operation like + or $(\cdot)^2$ is a node in the graph. For each of these operations, we know the symbolic form of the derivative. Hence, once we know the computational graph for a function, we can use forward or backward differentiation to compute the full gradient. How this works in practice can be seen in figs. 4.12 to 4.14. In the fig. 4.12, we construct the computational graph. In figs. 4.13 and 4.14 we compute the derivative using forward and backward differentiation respectively.

Remark. *Observe that we used a seed in both figs. 4.13 and 4.14. What we do in the automatic differentiation in actually computing directional derivative in the direction of the seed vector. So, if we have a function f which takes a vector and a seed vector p, then forward mode differentiation actually computes $(\nabla_x f)^\intercal p$. When the output is scalar, backward differentiation gives just the gradient. Since we our taking gradient of the loss function (which is scalar valued), we don't have to consider seed vectors. Hence, we will ignore them for the rest of this course.*

In the examples, we have been taking gradients with respect to the inputs. During training, we will be taking gradients with respect to the weights. These gradients will have many shared terms between them. By using backward differentiation, we can reuse the computed values for these shared terms. In many cases, this backward differentiation with respect to the parameters is not called backward differentiation but backward propagation. So, in the next section, we will refer to backward differentiation as backpropagation.

Remark. *In neural network, evaluating the network is also called the forward pass. We use backpropagation to compute the gradient of the loss. The term backpropagation comes from the fact that gradient information propagates due to backward differentiation backward, which in this case refers to the opposite direction as in the forward pass.*

Exercises.

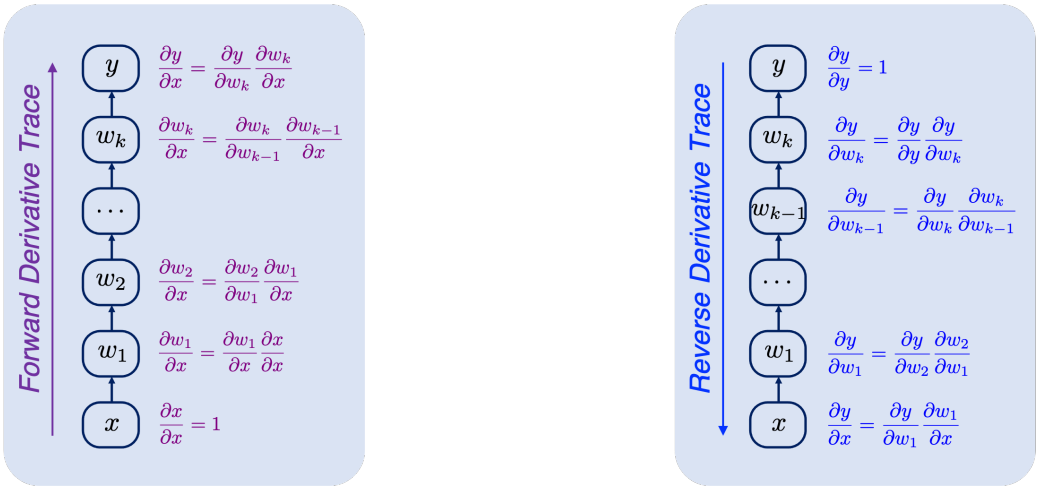
Exercise 4.3.1. Sketch the computational graph of example 4.3. Compute the gradient using forward and backward differentiation.

Exercise 4.3.2. Sketch the computational graph of example 4.4 for $L = 4$. Compute the gradient using forward and backward differentiation.

Exercise 4.3.3. Consider the function

$$f_w((x_1, x_2)) = x_2^2 + wx_2 \sin(x_1) \tag{4.34}$$

with weight $w \in \mathbb{R}$. This is a parametric version of the equation from fig. 4.12. Sketch the computational graph of eq. (4.34). Compute the gradient with respect to w using backward differentiation.



(A) Forward differentiation (B) Backward differentiation

FIGURE 4.11. Both direction of taking a derivative illustrated.

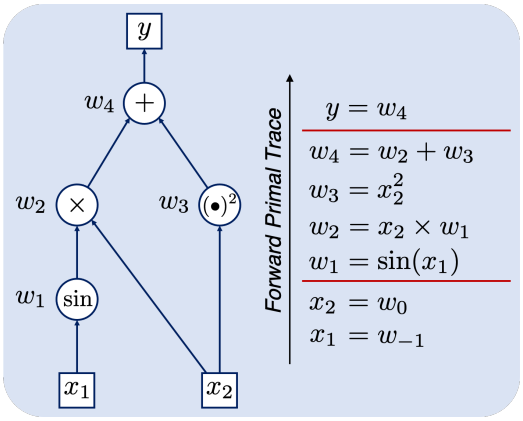


FIGURE 4.12. Evaluation of $f((x_1, x_2)) = x_2^2 + x_2 \sin(x_1)$ with corresponding computational graph.

Exercise 4.3.4. Consider again the function from eq. (4.34) and consider the point $\mathbf{x} = (1, \pi/4)$. The value it should have is $y = 2$. Pick several w and compute the gradient of

$$\mathcal{L}(w) = (y - f_w(\mathbf{x}))^2 \tag{4.35}$$

with respect to w using backward differentiation.

Exercise 4.3.5. Consider the neural network $f_{\boldsymbol{\theta}}$ from Exercise 4.2.1. Compute the gradient of

$$\mathcal{L}(\boldsymbol{\theta}) = (y - f_{\boldsymbol{\theta}}(\mathbf{x}))^2 \tag{4.36}$$

for $y = 2$ and $x = (1, \pi/4)$ with respect to $W^2_{1,1}$, $W^2_{2,2}$, b^1_1 and b^3_1 using backward differentiation.

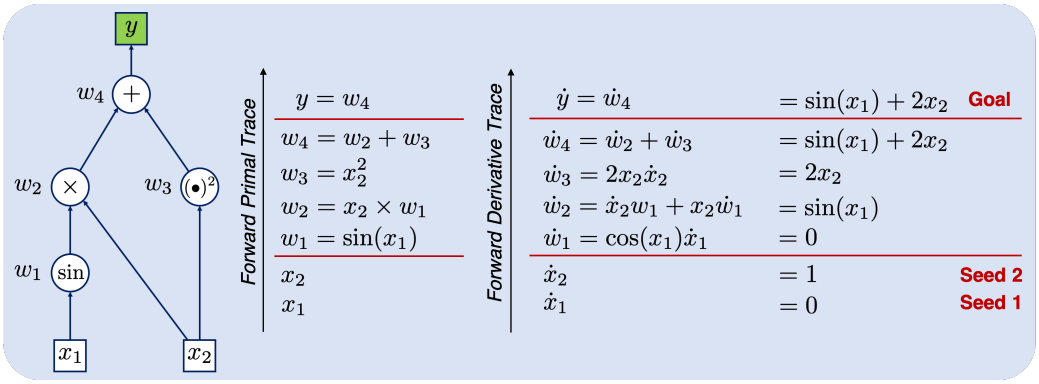


FIGURE 4.13. Forward differentiation for the function $f((x_1, x_2)) = x^2 + x_2 \sin(x_1)$.

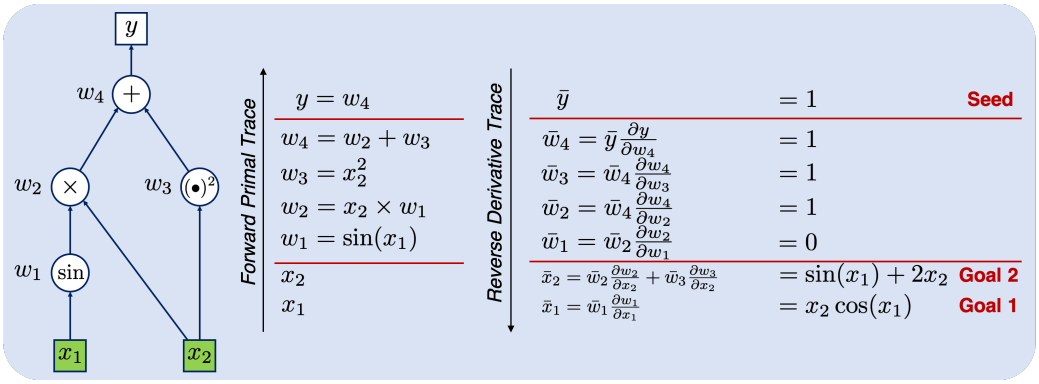


FIGURE 4.14. Backward differentiation for the function $f((x_1, x_2)) = x^2 + x_2 \sin(x_1)$.

Exercise 4.3.6. Consider the neural network $f_{\boldsymbol{\theta}}$ from Exercise 4.2.2. Compute the gradient of

$$\mathcal{L}(\boldsymbol{\theta}) = (y - f_{\boldsymbol{\theta}}(\mathbf{x}))^2 \tag{4.37}$$

for $y = 2$ and $\mathbf{x} = (1, \pi/4)$ with respect to $\mathbf{W}^1_{2,4}$, $\mathbf{W}^2_{2,2}$, $\mathbf{b}^1_3$ and $\mathbf{b}^2_2$ using backward differentiation.

4.3.2. Stochastic and batch gradient descent. Recall from section 3.3 that in order to do gradient descent we first find a descent direction and subsequently solve a minimisation problem to determine how far we should travel in that descent direction. We showed that this could be done through (in)exact line search or, if your function is nice enough, by choosing a particular sequence of step sizes t_k.

Neural networks are not nice enough for us to be able to pick a nicely converging sequence of step sizes $\eta^{(k)}$. If we do that, we most likely won't end up in any local minima. Since a neural network has many paramaters, we don't expect to take big steps sizes at all. Computing a line search to determine the best step size is

deemed not worth it. Instead, we opt for simply fixing a small step size before we start training. Since this step size determines how fast the network learns, it is in the context of machine learning called the *learning rate*.

When we take a gradient step for eq. (4.23) with the loss function eq. (4.24) with a fixed learning rate, we compute

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}_{MSE}(\boldsymbol{\theta}^{(k)}), \tag{4.38}$$

where the bracketed superscript refers to which gradient descent iteration we are in. We are using backpropagation, so we have to compute $\mathcal{L}_{MSE}(\boldsymbol{\theta}^{(k)})$ first. If N is large (as typically the case), then computing $\mathcal{L}_{MSE}(\boldsymbol{\theta}^{(k)})$ for all N is too expensive to compute. Hence, we use a sample of r points of $\mathcal{L}_{MSE}$ instead:

$$\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}) = \frac{1}{2} \sum_{n=1}^{r} ||\mathbf{y}_n - f_{\boldsymbol{\theta}}(\mathbf{x}_n)||_2^2. \tag{4.39}$$

To update the weights we thus compute

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta \nabla_{\boldsymbol{\theta}} \hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)}). \tag{4.40}$$

Each sample of r inputs is called a *batch*. The samples are down without replacement, so after approximately N/r iterations all inputs have sampled. When that has happens, all points can be sampled again and the cycle restarts. Each such cycle is called an *epoch*. If $r = 1$, then eq. (4.40) is called stochastic gradient descent. Otherwise, we call it batch gradient descent. Most of the time $r = 32$ or $r = 64$. This size allows for computing $\hat{\mathcal{L}}^r_{MSE}$ using vector operations and makes sure that the networks are not updated too often during each epoch.

Algorithm 3 Stochastic/Batch Gradient Descent Method

Init: Choose initial parameters $\boldsymbol{\theta}^{(0)}$, batch size $r \in \mathbb{N}$, learning rate η and number of epochs K.
for $k = 1$ up to K **do**
  Mark entire dataset as unsampled.
  while dataset not fully marked as sampled **do**
    if r unsampled points left in dataset **then**
      Sample r points from dataset and mark them sampled.
    else
      Sample remaining points from dataset and mark them sampled.
    Compute $\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta})$
    Set $\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta \nabla_{\boldsymbol{\theta}} \hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)})$.

Exercises.

Exercise 4.3.7. Consider the function

$$f_{w,b}((x_1, x_2)) = x^2 + w x_2 \sin(x_1) + b \tag{4.41}$$

and the tuples $(\mathbf{x}_1, y_1) = ((1, \pi/4), 2)$ and $(\mathbf{x}_1, y_1) = ((-1, -\pi/4), -2)$. Pick initial parameters w and b, and a learning rate η. Update the parameters by applying

gradient descent to the loss function

$$\mathcal{L}(w,b) = \frac{1}{2}\sum_i (y_i - f_{w,b}(\mathbf{x}_i))^2 \tag{4.42}$$

Then, repeat the process with the same initial w and b using stochastic gradient descent with $r = 1$. Are the resulting w and b the same? If not, which result is better?

4.3.3. Adaptive gradient descent. One of the issues with gradient descent is that it can converge a bit slow when you choose too small of a learning rate, with the particular issue that it can get stuck in a local optimum. In the chapter 2 several higher order methods have been discussed. The downsides of these methods was their extra computational costs. Due to the sheer number of parameters for neural networks, this added computational cost exceeds any performance gains. To still improve upon the standard gradient descent, several variations on stochastic gradient descent have been developed to help with these issues. These heuristics take into account previous gradients too and not just the current gradient. The ones that we will discuss are momentum, Nesterov acceleration, AdaGrad, RMSProp and Adam.

4.3.3.1. *Momentum.* When using momentum, we change eq. (4.40) to

$$\boldsymbol{\delta}^{(k+1)} = m\boldsymbol{\delta}^{(k)} - \eta\nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)}), \quad \boldsymbol{\delta}^{(0)} = 0 \tag{4.43a}$$

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \boldsymbol{\delta}^{(k+1)} \tag{4.43b}$$

with $m > 0$ being the momentum coefficient. It preserves some of the previous gradient information. This means that the gradient descent will preserve some speed, when it started with a lot of big gradients and then got some small gradients. This can, for example, happen when the gradient descent encounters a saddle point. Near those $\nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)})$ can be small. You don't the algorithm to stop in this local optimum, but continue to a better one. $m > 0$ allows you to pass this saddle. Another situation that it is useful is if the gradient descent zigzags. In that case, the zigzags would average out in $\boldsymbol{\delta}^{(k)}$ an allow the optimisation method to take the path through the middle. There is also a downside of having momentum. If you are close to a desired optimum, then gradient descent would grind to a halt due to small gradients. With momentum, it might completely overshoot the optimum, like a stone in curling overshooting the house or a ping pong ball refusing to stay in the cup in beer pong. So, it should be large enough to sway the gradients and small enough to not let the descent run past any optima. Since the choice for learning rate η influences the choice for m, eq. (4.43) is also written as

$$\boldsymbol{\delta}^{(k+1)} = \beta\nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)}) + (1-\beta)\boldsymbol{\delta}^{(k)}, \quad \boldsymbol{\delta}^{(0)} = 0 \tag{4.44a}$$

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta\boldsymbol{\delta}^{(k+1)} \tag{4.44b}$$

with $\beta \in (0,1)$. Here, β determines how to balance the history and the current gradient in order to determine the next change in parameters. In this formulation, the choices for β and η are independent. However, there is no rigorous mathematical principal to determine β.

Algorithm 4 Stochastic/Batch Gradient Descent Method with momentum

Init: Choose initial parameters $\boldsymbol{\theta}^{(0)}$, momentum m, batch size $r \in \mathbb{N}$, learning rate η and number of epochs K.
Set $\boldsymbol{\delta}^{(0)} = 0$
for $k = 1$ up to K **do**
  Mark entire dataset as unsampled.
  while dataset not fully marked as sampled **do**
    if r unsampled points left in dataset **then**
      Sample r points from dataset and mark them sampled.
    else
      Sample remaining points from dataset and mark them sampled.
    Compute $\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta})$
    Set $\boldsymbol{\delta}^{(k+1)} = m\boldsymbol{\delta}^{(k)} - \eta\nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)})$
    Set $\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta\boldsymbol{\delta}^{(k+1)}$.

4.3.3.2. *Nesterov.* In Nesterov acceleration, we not only give the weights some mass m, but also compute the gradient at a location based on this mass,

$$\boldsymbol{\delta}^{(k+1)} = m\boldsymbol{\delta}^{(k)} - \eta\nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)} + m\boldsymbol{\delta}^{(k)}), \quad \boldsymbol{\delta}^{(0)} = 0 \tag{4.45a}$$

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \boldsymbol{\delta}^{(k+1)} \tag{4.45b}$$

By adding $m\boldsymbol{\delta}^{(k)}$ into the loss, we perform some kind of look ahead. The idea is that this look ahead makes the iterations more stable.

Remark. *In 1983, Nesterov wrote a scheme for convex functions that takes 5 lines to write down. Sutskever wrote it as the 2-liner given in eq.* (4.43) *using an asymptotic approximation. In that version* m *depends on the iteration using* $m^{(k)} = 1 - \frac{3}{t+5}$*. Nesterov's original paper was a big deal, since for convex functions it has a* $O(\frac{1}{k^2})$ *convergence rate instead of the* $O(\frac{1}{k})$ *gradient descent has.*

4.3.3.3. *Adagrad.* The previous methods all have a global learning rate. This may not be good enough. You can have that some variables have a large gradient and some have a consistently small gradient. No choice for momentum will keep the large ones in check, whilst making sure the smaller ones get buffed by the momentum. So we want to use learning rates that adapt to the needs of each parameter. One way is to do some "normalisation" of the learning rate. Adagrad does this using

$$\mathbf{c}^{(k+1)} = \mathbf{c}^{(k)} + \nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)})^{\odot 2}, \quad \mathbf{c}^{(0)} = 0 \tag{4.46a}$$

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \frac{\eta}{\sqrt{\epsilon + \mathbf{c}^{(k)}}} \odot \nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)}) \tag{4.46b}$$

where $0 < \epsilon \ll 1$ is a very small number to prevent you from dividing by zero, $\mathbf{c}^{(k)}$ is called the cache, $\odot$ means component-wise multiplication and $^{\odot 2}$ means component-wise squaring. The addition of $\mathbf{c}^{(k)}$ in the square root means that size of the previous gradients steers the learning rate η by reducing the learning rate for parameters with a high gradient. A downside of Adagrad is that a single large gradient for an element in a series of iterates k dominates the corresponding element

Algorithm 5 Nesterov accelerated Gradient Descent Method

Init: Choose initial parameters $\boldsymbol{\theta}^{(0)}$, momentum m, batch size $r \in \mathbb{N}$, learning rate η and number of epochs K.
Set $\boldsymbol{\delta}^{(0)} = 0$
for $k = 1$ up to K **do**
  Mark entire dataset as unsampled.
  while dataset not fully marked as sampled **do**
    if r unsampled points left in dataset **then**
      Sample r points from dataset and mark them sampled.
    else
      Sample remaining points from dataset and mark them sampled.
    Set $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta}^{(k)} + m\boldsymbol{\delta}^{(k)}$
    Compute $\hat{\mathcal{L}}^r_{MSE}(\tilde{\boldsymbol{\theta}})$
    Set $\boldsymbol{\delta}^{(k+1)} = m\boldsymbol{\delta}^{(k)} - \eta\nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}^r_{MSE}(\tilde{\boldsymbol{\theta}})$
    Set $\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta\boldsymbol{\delta}^{(k+1)}$.

of $\mathbf{c}^{(k)}$, since $\mathbf{c}^{(k)}$ keeps track of the gradients over all iterates. Furthermore, $\mathbf{c}^{(k)}$ can only increase, which means it can only decrease η. This limits its effectiveness.

Algorithm 6 Adagrad

Init: Choose initial parameters $\boldsymbol{\theta}^{(0)}$, momentum m, batch size $r \in \mathbb{N}$, learning rate η and number of epochs K.
Set $\mathbf{c}^{(0)} = 0$
for $k = 1$ up to K **do**
  Mark entire dataset as unsampled.
  while dataset not fully marked as sampled **do**
    if r unsampled points left in dataset **then**
      Sample r points from dataset and mark them sampled.
    else
      Sample remaining points from dataset and mark them sampled.
    Compute $\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)})$
    Set $\mathbf{c}^{(k+1)} = \mathbf{c}^{(k)} + \nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)})^{\odot 2}$
    Set $\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \frac{\eta}{\sqrt{\epsilon+\mathbf{c}^{(k)}}} \odot \nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)})$.

4.3.3.4. *RMSProp.* RMSProp tries to fix both problems by letting the cache $\mathbf{c}^{(k)}$ decay,

$$\mathbf{c}^{(k+1)} = \boldsymbol{\delta}\mathbf{c}^{(k)} + (1-\boldsymbol{\delta})\nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)})^{\odot 2}, \quad \mathbf{c}^{(0)} = 0 \tag{4.47a}$$

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \frac{\eta}{\sqrt{\epsilon+\mathbf{c}^{(k)}}} \odot \nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)}) \tag{4.47b}$$

with the decay $\boldsymbol{\delta} \in (0,1)$. Typically, $\boldsymbol{\delta} \in \{0.9, 0.99, 0.999\}$. This means that most of the cache is determined by the previous $\mathbf{c}^{(k)}$ and the historical influence decays exponentially. This significantly improves the performance is some cases and minorly improves the performance in most cases.

Algorithm 7 RMSProp

Init: Choose initial parameters $\boldsymbol{\theta}^{(0)}$, momentum m, batch size $r \in \mathbb{N}$, learning rate η and number of epochs K.
Set $\mathbf{c}^{(0)} = 0$
for $k = 1$ up to K **do**
  Mark entire dataset as unsampled.
  while dataset not fully marked as sampled **do**
    if r unsampled points left in dataset **then**
      Sample r points from dataset and mark them sampled.
    else
      Sample remaining points from dataset and mark them sampled.
    Compute $\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)})$
    Set $\mathbf{c}^{(k+1)} = \boldsymbol{\delta}\mathbf{c}^{(k)} + (1-\boldsymbol{\delta})\nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)})^{\odot 2}$
    Set $\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \frac{\eta}{\sqrt{\epsilon + \mathbf{c}^{(k)}}} \odot \nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)})$.

4.3.3.5. *Adam.* Adam looks like RMSProp with momentum,

$$\mathbf{v}^{(k+1)} = \beta_1 \mathbf{v}^{(k)} + (1-\beta_1)\nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)}), \quad \mathbf{v}^{(0)} = 0 \tag{4.48a}$$

$$\tilde{\mathbf{v}}^{(k+1)} = \mathbf{v}^{(k+1)}/(1-\beta_1^k) \tag{4.48b}$$

$$\mathbf{c}^{(k+1)} = \beta_2 \mathbf{c}^{(k)} + (1-\beta_2)\nabla_{\boldsymbol{\theta}}\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)})^{\odot 2}, \quad \mathbf{c}^{(0)} = 0 \tag{4.48c}$$

$$\tilde{\mathbf{c}}^{(k+1)} = \mathbf{c}^{(k+1)}/(1-\beta_2^k) \tag{4.48d}$$

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \frac{\eta}{\sqrt{\epsilon + \tilde{c}^{(k+1)}}} \odot \tilde{\mathbf{v}}^{(k+1)} \tag{4.48e}$$

with $\beta_1, \beta_2 \in (0,1)$ and $0 < \epsilon \ll 1$. Typically, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Equation (4.48a) is similar to eq. (4.44a) and eq. (4.48c) is similar to eq. (4.47a), but with β_1 and β_2 instead of β and δ. Without eq. (4.48b) and eq. (4.48d) the algorithm has a bias, and these terms correct that. The combination of both momentum and caching means that this method is one of the most used and best performing algorithms for neural networks.

Algorithm 8 Adam

Init: Choose initial parameters $\boldsymbol{\theta}^{(0)}$, momentum m, batch size $r \in \mathbb{N}$, learning rate η and number of epochs K.
Set $\mathbf{c}^{(0)} = 0$
Set $v^{(0)} = 0$
for $k = 1$ up to K **do**
    Mark entire dataset as unsampled.
    while dataset not fully marked as sampled **do**
        if r unsampled points left in dataset **then**
            Sample r points from dataset and mark them sampled.
        else
            Sample remaining points from dataset and mark them sampled.
        Compute $\hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)})$
        Set $v^{(k+1)} = \beta_1 v^{(k)} + (1-\beta_1)\nabla_{\boldsymbol{\theta}} \hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)})$
        Set $\tilde{v}^{(k+1)} = v^{(k+1)}/(1-\beta_1^k)$
        Set $\mathbf{c}^{(k+1)} = \beta_2 \mathbf{c}^{(k)} + (1-\beta_2)\nabla_{\boldsymbol{\theta}} \hat{\mathcal{L}}^r_{MSE}(\boldsymbol{\theta}^{(k)})^{\odot 2}$
        Set $\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \frac{\eta}{\sqrt{\epsilon + \tilde{c}^{(k+1)}}} \odot \tilde{v}^{(k+1)}$.

Back matter

Bibliography

[1] A.V. Aho, J.E. Hopcraft and J.D. Ullman, The design and Analysis of Computer Algorithms, Addison-Wesley, Reading, (1974).

[2] A. Bazaraa, H. Sherali and C. Shetty, Nonlinear Programming, John Wiley, New York, (1993).

[3] A. Cohen, *Rate of convergence of several conjugate gradients algorithms*, SIAM Journal on Numerical Analysis, 9, 248-259, (1972).

[4] H.G. Daellenbach, Systems and Decision Making. John Wiley & Sons, Chichester (1994).

[5] J.E. Dennis and R.B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice Hall, London, (1983).

[6] J. Edmonds, *Systems of Distinct Representation and Linear Algebra*, Journal of Research of the National Bureau of Standards, 71B,(4), (1967)

[7] Faigle U., Kern W., Still G., Algorithmic Principles of Mathematical Programming, Kluwer, Dordrecht, (2002).

[8] J. Farkas, *Über die Theorie der einfachen Ungleichungen*, J. für die Reine und Angewandte Mathematik, 124, 1-27, (1902).

[9] W. Feller, *An Introduction to Probability Theory and Its Applications*, John Wiley, New York, 3rd. rev. ed., (1970).

[10] R. Fletcher, Practical Methods of Optimization, John Wiley & Sons, Chichester, (1987).

[11] J.B. Fourier, Solution d'une question particuliere du calcul des inégalités, Oeuvres II, 317-328, (1826).

[12] B. Ganter and R. Wille, *Formal Concept Analysis*, Springer-Verlag, Berlin, (1999).

[13] G.H. Golub and C.F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press 3rd. ed., (1996).

[14] P. Gordan, *Über die Auflösung linearer Gleichungen mit reellen Coefficienten*, Math. Ann. 6, 23-28, (1873).

[15] J. Gruska, Quantum Computing, McGraw Hill, London, (1999).

[16] R. Horst and H. Tuy, Global Optimization, Springer, Berlin, (1996).

[17] P. Lancaster and M. Tismenetsky, The Theory of Matrices, Academic Press, Boston, (1985).

[18] G. Lekkerkerker, Geometry of Numbers. North-Holland, Amsterdam, (1969).

[19] D.G. Luenberger, Optimization by Vector Space Methods, John Wiley & Sons, New York, (1969).

[20] D.G. Luenberger, Introduction to Linear and Nonlinear Programming, Addison-Wesley, (1980).

[21] T.S. Motzkin, *Beiträge zur Theorie der Linearen Ungleichungen*, Dissertation, University of Basel, Jerusalem, (1936).

[22] J. von Neumann, *Zur Theorie der Gesellschaftsspiele*, Mathematische Annalen 100, 295-320, (1928).

[23] J. von Neumann, *A certain zero-sum game equivalent to the optimal assignment problem.* In: Contributions to the Theory of Games I. H.W. Kuhn and A.W. Tucker, eds., Annals of Mathematics Studies 28, 5-12, (1953).

[24] J. Nocedal and S.J. Wright, Numerical Optimization, Springer (1999).

[25] C.R. Rao, Linear Statistical Inference and Its Applications, Wiley, New York, (1973).

[26] W. Rudin, *Principles of Mathematical Analysis*, (third edition), McGraw-Hill, (1976).

[27] A. Schrijver, Theory of Linear and Integer programming, John Wiley, New York, (1986).

[28] P. Spelucci, *Numerische Verfahren der nichtlinearen Optimierung*, Birkhäuser Verlag, Boston, (1993).
[29] D.E. Steward, Solving nonlinear equations, In: Numerical Analysis: A Graduate Course. CMS/CAIMS Books in Mathematics, vol 4. Springer, Cham, (2022).
[30] D. Welsh, *Codes and Cryptography*, Clardendon Press, Oxford, (1988).
[31] G.M. Ziegler, Lectures on Polytopes, Springer-Verlag, New York, (1999).

APPENDIX A

Real Vector Spaces

This appendix is a brief review of the basic notions and facts from linear algebra and analysis that we will use as tools in mathematical programming. The reader is assumed to be already familiar with most of the material in this chapter. The proofs we sketch here (as well as the exercises) are mainly meant as reminders.

A.1. Inner Products and Norms

An *inner product* on $\mathbb{R}^n$ is a map $\langle\cdot|\cdot\rangle : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ such that for all vectors $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^n$ and scalars $\lambda \in \mathbb{R}$,

$$\langle \mathbf{x}|\mathbf{y}\rangle = \langle \mathbf{y}|\mathbf{x}\rangle \tag{A.1}$$

$$\langle \lambda\mathbf{x}|\mathbf{y}\rangle = \lambda\langle \mathbf{x}|\mathbf{y}\rangle \tag{A.2}$$

$$\langle \mathbf{x}+\mathbf{y}|\mathbf{z}\rangle = \langle \mathbf{x}|\mathbf{z}\rangle + \langle \mathbf{y}|\mathbf{z}\rangle \tag{A.3}$$

$$\langle \mathbf{x}|\mathbf{x}\rangle > 0 \qquad \text{if } \mathbf{x} \neq \mathbf{0}. \tag{A.4}$$

By (A.1)-(A.3), an inner product is a *symmetric bilinear form* and, by (A.4), *positive definite*. We will usually be concerned with the so-called *standard inner product* for $\mathbf{x} = (x_1, \ldots, x_n)^T$ and $\mathbf{y} = (y_1, \ldots, y_n)^T$, which is a special case of the matrix product:

$$\langle \mathbf{x}|\mathbf{y}\rangle = \mathbf{x}^T\mathbf{y} = \sum_{j=1}^{n} x_j y_j \, .$$

Lemma A.1 (Cauchy-Schwarz)**.** *Let $\langle\cdot|\cdot\rangle$ be an inner product on $\mathbb{R}^n$. Then all vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ satisfy the inequality*

$$\langle \mathbf{x}|\mathbf{y}\rangle^2 \;\leq\; \langle \mathbf{x}|\mathbf{x}\rangle\langle \mathbf{y}|\mathbf{y}\rangle \, .$$

Equality holds if and only if $\mathbf{x}$ is a scalar multiple of $\mathbf{y}$.

Definition A.2 (Positive (semi)definite)**.** Given an inner product on $\mathbb{R}^n$ and a linear transformation $\mathbf{L} : \mathbb{R}^n \to \mathbb{R}^n$, we have the following notions.

- $\mathbf{L}$ is called a *symmetric transformation* if for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ holds:

$$\langle \mathbf{L}\mathbf{x}|\mathbf{y}\rangle = \langle \mathbf{x}|\mathbf{L}\mathbf{y}\rangle.$$

- $\mathbf{L}$ is called a *positive semidefinite transformation*, if it is a symmetric transformation such that for all $\mathbf{x} \in \mathbb{R}^n$ holds:

$$\langle \mathbf{x}|\mathbf{L}\mathbf{x}\rangle \geq 0.$$

- $\mathbf{L}$ is called a *positive definite transformation*, if it is a symmetric transformation such that for all $\mathbf{x} \in \mathbb{R}^n$, with $\mathbf{x} \neq \mathbf{0}$ holds:

$$\langle \mathbf{x} | \mathbf{L}\mathbf{x} \rangle > 0.$$

Mostly, we use the notions of *symmetric* and *positive (semi)definite matrices*, where we view matrices as a linear transformation with respect to standard matrix-vector multiplication:

A square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is called *symmetric* if:

$$\mathbf{A} = \mathbf{A}^T .$$

A square symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is called *positive semidefinite (p.s.d.)*, denoted $\mathbf{A} \succeq \mathbf{0}$, if:

$$\mathbf{x}^T \mathbf{A}\mathbf{x} \geq 0 .$$

Moreover, we say that $\mathbf{A}$ is *positive definite (p.d.)*, denoted $\mathbf{A} \succ \mathbf{0}$, if additionally

$$\mathbf{x}^T \mathbf{A}\mathbf{x} = 0 \quad \Leftrightarrow \quad \mathbf{x} = \mathbf{0} .$$

Lemma A.3. *Let $\mathbf{S} \in \mathbb{R}^{n \times n}$ be a symmetric matrix. Then*

(a) $\mathbf{S}$ is p.s.d. if and only if there is a matrix $\mathbf{A}$ such that $\mathbf{S} = \mathbf{A}\mathbf{A}^T$.
(b) $\mathbf{S}$ is positive definite if and only if there is an invertible matrix $\mathbf{A}$ such that $\mathbf{S} = \mathbf{A}\mathbf{A}^T$. □

Theorem A.4 (Spectral Theorem for Symmetric Matrices)**.** *Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be a symmetric matrix. Then there exists a matrix $\mathbf{Q} \in \mathbb{R}^{n \times n}$, and eigenvalues $\lambda_1, \ldots, \lambda_n$ of $\mathbf{A}$ such that*

$$\mathbf{Q}^T \mathbf{Q} = \mathbf{I} \quad \text{and} \quad \mathbf{Q}^T \mathbf{A}\mathbf{Q} = \operatorname{diag}(\lambda_1, \ldots, \lambda_n) .$$

Lemma A.5. *A square symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ only has real-valued eigenvalues.*

Theorem A.6. *Let $\mathbf{A}$ be a symmetric matrix and $\mathbf{Q}$ an invertible matrix such that $\mathbf{D} = \mathbf{Q}\mathbf{A}\mathbf{Q}^T$ is diagonal. Then*

(a) $\mathbf{A}$ is p.s.d. if and only if all diagonal elements of $\mathbf{D}$ are non-negative.
(b) $\mathbf{A}$ is positive definite if and only if all diagonal elements of $\mathbf{D}$ are strictly positive.

Corollary A.7. *Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of the positive definite, respectively positive semidefinite, matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$. Then $\lambda_i \in \mathbb{R}$ and $\lambda_i > 0$, respectively $\lambda_i \geq 0$, for all $i \in \{1, \ldots, n\}$.*

Lemma A.8 (Identification of 2×2 positive (semi-)definite matrices)**.** *Let $\mathbf{A} \in \mathbb{R}^{2 \times 2}$ be a symmetric matrix. Then*

(a) $\mathbf{A}$ is positive semidefinite *if and only if all diagonal elements and the determinant of $\mathbf{A}$ are non-negative.*
(b) $\mathbf{A}$ is positive definite *if and only if all diagonal elements and the determinant of $\mathbf{A}$ are strictly positive.*

A.1.1. Norms. We can speak reasonably about the "length" of a vector in $\mathbb{R}^n$ only relative to a given *norm* on $\mathbb{R}^n$, that is, a map $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ such that for all vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and scalars $\lambda \in \mathbb{R}$,

$$\|\mathbf{x}\| > \mathbf{0} \qquad \text{for } \mathbf{x} \neq \mathbf{0}; \tag{A.5}$$

$$\|\lambda\mathbf{x}\| = |\lambda| \cdot \|\mathbf{x}\|; \tag{A.6}$$

$$\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|. \tag{A.7}$$

Inequality (A.7) is the *triangle inequality.*

Every inner product $\langle\cdot|\cdot\rangle$ on $\mathbb{R}^n$ gives rise to a norm *via*

$$\|\mathbf{x}\| \;=\; \sqrt{\langle\mathbf{x}|\mathbf{x}\rangle}\,.$$

The norm arising from the standard inner product on $\mathbb{R}^n$ is the *Euclidean norm*

$$\|\mathbf{x}\| \;=\; \sqrt{\mathbf{x}^T\mathbf{x}} = \sqrt{x_1^2 + \ldots + x_n^2}\,.$$

A.2. Continuous and Differentiable Functions

A.2.1. Topology of $\mathbb{R}^n$. In this section we consider the Euclidean norm and denote it simply $\|\cdot\| = \|\cdot\|_2$ (unless explicitly specified otherwise). For two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ we define their *(Euclidean) distance* as $\|\mathbf{x} - \mathbf{y}\|$. In the case $n = 1$, of course, we also use the familiar notation of the absolute value $|x - y|$.

The set of real numbers $\mathbb{R}$ has (by definition) the following so-called *completeness property* : A non-decreasing infinite sequence of real numbers $r_1 \leq r_2 \leq \ldots$ has a (unique) limit

$$r = \lim_{k\to\infty} r_k \in \mathbb{R}$$

if and only if there exists a *bound* $M \in \mathbb{R}$ such that $r_k \leq M$ holds for all k. As a consequence of this property , every subset $S \subseteq \mathbb{R}$ has a unique *infimum*, which is defined as the largest lower bound for all $s \in S$ and denoted by $\inf S$. (If S is not bounded from below, then $\inf S = -\infty$ and if $S = \emptyset$, then $\inf S = +\infty$.) The *supremum* $\sup S$ is defined similarly.

We say that a sequence $\mathbf{s}_1, \mathbf{s}_2, \ldots$ of points $\mathbf{s}_k \in \mathbb{R}^n$ *converges* if the exists some $\mathbf{s} \in \mathbb{R}^n$ such that

$$\lim_{k\to\infty} \|\mathbf{s} - \mathbf{s}_k\| \;=\; 0\,,$$

which is denoted by $\mathbf{s} = \lim_{k\to\infty} \mathbf{s}_k$ or just $\mathbf{s}_k \to \mathbf{s}$. A subset $S \subset \mathbb{R}^n$ is *bounded* if there exists some $M \in \mathbb{R}$ such that $\|\mathbf{s}\| \leq M$ holds for all $\mathbf{s} \in S$. The completeness property of $\mathbb{R}$ then implies the following:

- $S \subset \mathbb{R}^n$ is bounded if and only if every infinite sequence $\mathbf{s}_1, \mathbf{s}_2, \ldots$ of points $\mathbf{s}_k \in S$ admits a convergent subsequence $\mathbf{s}_{k_1}, \mathbf{s}_{k_2}, \ldots$.

An *open ball* (of radius $r > 0$ and centered at $\mathbf{x}_0 \in \mathbb{R}^n$) is a subset of the form

$$U_r(\mathbf{x}_0) \;=\; \{\mathbf{x} \in \mathbb{R}^n \mid \|\mathbf{x} - \mathbf{x}_0\| < r\}\,.$$

A set $U \subseteq \mathbb{R}^n$ is *open* if U contains with any $\mathbf{x}_0$ also some open ball $U_r(\mathbf{x}_0)$. (In other words: An open set is precisely the union of all the open balls it contains.) A subset $C \subseteq \mathbb{R}^n$ is *closed* if $\mathbb{R}^n \backslash C$ is open.

(Arbitrary) unions of open sets are open and, correspondingly, intersections of closed sets are closed. In particular, every $S \subseteq \mathbb{R}^n$ has a unique smallest closed set containing S, namely the intersection cl S of all closed sets containing S. cl S is the so-called *closure* of S. Similarly, every $S \subseteq \mathbb{R}^n$ admits a unique maximal open set int S contained in S, namely the union of all open balls contained in S, the *interior* of S.

The *boundary* of $S \subseteq \mathbb{R}^n$ is defined as $\partial S = \text{cl } S \backslash \text{int } S$.

A subset $S \subset \mathbb{R}^n$ is *compact*, if it is bounded and closed. So S is compact if and only if every infinite sequence $(\mathbf{s}_k)$ in S has an accumulation point $\overline{\mathbf{s}} \in S$. (The existence of $\overline{\mathbf{s}}$ is equivalent to the boundedness of S and $\overline{\mathbf{s}} \in S$ is equivalent to the closedness of S.) This observation is sometimes referred to as the *Theorem of Bolzano-Weierstrass*.

A.2.2. Continuous Functions. Let S be a given subset of $\mathbb{R}^n$. Then the function $f : S \to \mathbb{R}^m$ is said to be *continuous* at the point $\mathbf{x}_0 \in S$ if for every $\varepsilon > 0$ there exists some $\delta > 0$ so that

$$\|f(\mathbf{x}_0) - f(\mathbf{x})\|_2 < \varepsilon \text{ holds whenever } \mathbf{x} \in S \text{ and } \|\mathbf{x}_0 - \mathbf{x}\|_2 < \delta,$$

or, equivalently, $f(U_\delta(\mathbf{x}_0) \cap S) \subseteq U_\varepsilon(f(\mathbf{x}_0))$. We denote this property by

$$\boxed{f(\mathbf{x}_0) = \lim_{\mathbf{x} \to \mathbf{x}_0} f(\mathbf{x})}$$

We say that f is *continuous on* S if f is continuous at every $\mathbf{x}_0 \in S$.

Sums and products of real-valued continuous functions are again continuous, as are compositions of continuous functions.

Lemma A.9. *Let $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ be an arbitrary norm on $\mathbb{R}^n$. Then $f(\mathbf{x}) = \|\mathbf{x}\|$ is a continuous function.*

If $C \subseteq \mathbb{R}^n$ is compact and $f : C \to \mathbb{R}^m$ is continuous, then the image $f(C)$ is compact in $\mathbb{R}^m$.

Theorem A.10. (Weierstrass) *Let $C \subset \mathbb{R}^n$ be a nonempty compact set and $f : C \to \mathbb{R}$ be continuous. Then f attains its maximum and minimum value on C,* i.e. *there exist $\mathbf{x}_0, \mathbf{x}_1 \in C$ with*

$$f(\mathbf{x}_0) \leq f(\mathbf{x}) \leq f(\mathbf{x}_1) \qquad \textit{for all } \mathbf{x} \in C.$$

A.2.3. Differentiable Functions. From a computational point of view, linear and affine functions are the easiest to deal with. Therefore, we are interested in the question when a not necessarily linear function can be, at least locally, approximated by a linear function. Again, we focus right directly on the vector spaces $\mathbb{R}^n$ and $\mathbb{R}^m$, equipped with the standard inner product and the Euclidean norm. In each case, we choose as reference the standard basis of unit vectors $\mathbf{e}_j = (\ldots, 0, 1, 0, \ldots)^T$.

Let U be an open subset in $\mathbb{R}^n$. The function $f : U \to \mathbb{R}^m$ is said to be *differentiable* at the point $\mathbf{x}_0 \in U$ if there exists a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and a function

$\varphi : U \to \mathbb{R}^m$ such that $\lim_{\mathbf{h}\to\mathbf{0}} \varphi(\mathbf{h}) = \mathbf{0}$ and

$$f(\mathbf{x}_0 + \mathbf{h}) \;=\; f(\mathbf{x}_0) + \mathbf{A}\mathbf{h} + \|\mathbf{h}\|\varphi(\mathbf{h}) \quad \text{for all } \mathbf{x}_0 + \mathbf{h} \in U \;.$$

A shorter way of expressing these conditions is offered by the notation

$$\boxed{f(\mathbf{x}_0 + \mathbf{h}) = f(\mathbf{x}_0) + \mathbf{A}\mathbf{h} + o(\|\mathbf{h}\|)}$$

(Recall that $o(\|\mathbf{h}\|^k)$ generally denotes a term of the form $\|\mathbf{h}\|^k\eta(\mathbf{h})$, where $\eta(\mathbf{h})$ is a function satisfying $\lim\limits_{\mathbf{h}\to\mathbf{0}} \eta(\mathbf{h}) = \mathbf{0}$.)

The definition says that the differentiable function f can be approximated near $\mathbf{x}_0$ *via* the affine function

$$\tilde{f}(\mathbf{h}) = f(\mathbf{x}_0) + \mathbf{A}\mathbf{h} \;.$$

The matrix $\mathbf{A}$ is called the *derivative* of f at $\mathbf{x}_0$ and is generally denoted by $\nabla f(\mathbf{x}_0)$ (and by $f'(x_0)$ in case $n = 1$). We call f *differentiable on* U if f is differentiable at every $\mathbf{x}_0 \in U$. The derivative $\nabla f(\mathbf{x}_0)$ turns out to be nothing but the *Jacobian* matrix associated with f (see p. 79).

Derivatives of Functions in One Variable. The analysis of functions of several variables can often be reduced to the one-dimensional case. Therefore, we briefly review some basic and important facts for functions f in one real variable x (with f' denoting the derivative).

Let f be defined on the open interval $(a, b) \subseteq \mathbb{R}$ and differentiable at the point $x_0 \in (a, b)$. The following is a key observation for many optimization problems.

Lemma A.11. *If $f'(x_0) > 0$, then there exists some $\delta > 0$ such that*

$$f(x_0 - h) < f(x_0) < f(x_0 + h) \qquad \textit{whenever } 0 < h < \delta \;.$$

An immediate consequence of Lemma A.11 is the *extremum principle*: If $f : (a, b) \to \mathbb{R}$ is differentiable at $x_0 \in (a, b)$ and if either $f(x_0) = \max_{x\in(a,b)} f(x)$ or $f(x_0) = \min_{x\in(a,b)} f(x)$, then

$$f'(x_0) \;=\; 0 \,.$$

Theorem A.12. (Mean Value Theorem) *Let $f : [a, b] \to \mathbb{R}$ be continuous on the closed interval $[a, b]$ and differentiable on the open interval (a, b). Then there exists some $\xi \in (a, b)$ such that*

$$f(b) - f(a) = (b - a)f'(\xi) \;.$$

As an application of the mean Value Theorem we obtain the second order *Taylor formula.*

Lemma A.13. *Let $U = (-t_0, t_0) \subseteq \mathbb{R}$ and assume $p : U \to \mathbb{R}$ is twice differentiable. Then, for any given $t \in U$ there exists $0 < \theta < 1$, such that*

$$p(t) = p(0) + tp'(0) + \frac{1}{2}t^2 p''(\theta t) \;.$$

Directional Derivatives and the Gradient. A function $f : U \to \mathbb{R}^m$ assigns to each vector $\mathbf{x} \in U$ a vector

$$f(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_m(\mathbf{x}))^T \in \mathbb{R}^m \ .$$

It follows from the definition that f is differentiable at $\mathbf{x}_0$ if and only if each of the real-valued component functions $f_i : U \to \mathbb{R}$ is differentiable at $\mathbf{x}_0$. So we may restrict our attention to the case $m = 1$.

Consider the differentiable real-valued function $f : U \to \mathbb{R}$ at the point $\mathbf{x}_0 \in U$ and fix a "direction" $\mathbf{d} \in \mathbb{R}^n$. Then f induces locally a function

$$p_{\mathbf{d}}(t) = f(\mathbf{x}_0 + t\mathbf{d})$$

of the real parameter t. Moreover, the representation

$$f(\mathbf{x}_0 + t\mathbf{d}) = f(\mathbf{x}_0) + t\nabla f(\mathbf{x}_0)\mathbf{d} + \|t\mathbf{d}\|\varphi(t\mathbf{d})$$

immediately shows that the derivative $p'_{\mathbf{d}}(0)$ exists. In fact, letting $\tilde{\varphi}(t) = \|\mathbf{d}\|\varphi(t\mathbf{d})$, we have

$$p_{\mathbf{d}}(t) = p_{\mathbf{d}}(0) + t\nabla f(\mathbf{x}_0)\mathbf{d} + |t|\tilde{\varphi}(t)$$

and hence $p'_{\mathbf{d}}(0) \ = \ \nabla f(\mathbf{x}_0)\mathbf{d}$.

Choosing the direction $\mathbf{d}$ as a unit vector $\mathbf{e}_j$, we recognize what the derivative matrix $\nabla f(\mathbf{x}_0) = (a_1, \ldots, a_n) \in \mathbb{R}^{1\times n}$ actually is and how it can be computed. Recall first that the *partial derivative* $\partial f/\partial x_j$ of f at $\mathbf{x}_0$ is defined to be the derivative $p'_{\mathbf{e}_j}(0)$ of the function $p_{\mathbf{e}_j}(t) = f(\mathbf{x}_0 + t\mathbf{e}_j)$, which implies

$$a_j = \nabla f(\mathbf{x}_0)\mathbf{e}_j = \frac{\partial f(\mathbf{x}_0)}{\partial x_j} \ .$$

Hence, the derivative of $f : U \to \mathbb{R}$ is given by

$$\nabla f(\mathbf{x}_0) = \left[\frac{\partial f(\mathbf{x}_0)}{\partial x_1}, \ldots, \frac{\partial f(\mathbf{x}_0)}{\partial x_n}\right]$$

and is called the *gradient* of f at $\mathbf{x}_0$.

In general, we note for $p_{\mathbf{d}}(t) = f(\mathbf{x}_0 + t\mathbf{d})$ the formula for the *directional derivative of f at $\mathbf{x}_0$ with respect to $\mathbf{d}$*:

$$\boxed{p'_{\mathbf{d}}(0) = \nabla f(\mathbf{x}_0)\mathbf{d} = \sum_{j=1}^{n} \frac{\partial f(\mathbf{x}_0)}{\partial x_j} d_j} \tag{A.8}$$

Formula (A.8) allows us to determine the direction with respect to which f offers the largest marginal change at $\mathbf{x}_0$. With $p_{\mathbf{d}}(t)$ defined as before, we want to solve the optimization problem

$$\max_{\mathbf{d}\in\mathbb{R}^n} \ |p'_{\mathbf{d}}(0)| \quad \text{subject to } \|\mathbf{d}\| = 1.$$

We assume $\nabla f(\mathbf{x}_0) \neq \mathbf{0}^T$. Applying the Cauchy-Schwarz inequality to (A.8), we deduce

$$|p'_{\mathbf{d}}(0)| \ \leq \ \|\nabla f(\mathbf{x}_0)\| \cdot \|\mathbf{d}\| = \|\nabla f(\mathbf{x}_0)\|$$

with equality if and only if $\mathbf{d}^T = \lambda \nabla f(\mathbf{x}_0)$ for some $\lambda \in \mathbb{R}$. Hence we find that the gradient $\nabla f(\mathbf{x}_0)$ yields the direction $\mathbf{d}$ into which f exhibits the largest marginal change at $\mathbf{x}_0$. Depending on the sign of λ, we obtain the direction of largest increase or largest decrease.

Moreover, we have the general *extremum principle*: If $\mathbf{x}_0 \in U$ is a *local minimizer* or *maximizer* of f in the sense that for some $\varepsilon > 0$

$$f(\mathbf{x}_0) = \max_{\mathbf{x} \in U_\varepsilon(\mathbf{x}_0)} f(\mathbf{x}) \qquad \text{or} \qquad f(\mathbf{x}_0) = \min_{\mathbf{x} \in U_\varepsilon(\mathbf{x}_0)} f(\mathbf{x}) \,,$$

the one-dimensional extremum principle says that $0 = p'_{\mathbf{d}}(0) = \nabla f(\mathbf{x}_0)\mathbf{d}$ must hold for all directions $\mathbf{d}$, which implies that $\mathbf{x}_0$ must be a *critical point*, *i.e.*, satisfy the *critical equation*

$$\boxed{\nabla f(\mathbf{x}_0) = \mathbf{0}^T} \tag{A.9}$$

For general $f : U \to \mathbb{R}^m$, the same reasoning as in the case $m = 1$ shows that the derivative $\nabla f(\mathbf{x}_0)$ has as rows exactly the gradients of the component functions f_i of f. Hence the derivative $\nabla f(\mathbf{x}_0)$ of f at $\mathbf{x}_0$ is the *Jacobian* matrix

$$\nabla f(\mathbf{x}_0) \;=\; \left(\frac{\partial f_i(\mathbf{x}_0)}{\partial x_j} \right) \in \mathbb{R}^{m \times n} \,.$$

The existence of the Jacobian $\nabla f(\mathbf{x})$ (*i.e.*, the existence of the partial derivatives $\partial f_i(\mathbf{x})/\partial x_j$) alone does not necessarily guarantee the differentiability of f at $\mathbf{x}$. A sufficient – and in practice quite useful – condition is the continuity of the (generally non-linear) map $\mathbf{x} \mapsto \nabla f(\mathbf{x})$.

When $m = 1$, we refer to the Jacobian as the *gradient*.

Lemma A.14. *Let $U \subseteq \mathbb{R}^n$ be open and $f : U \to \mathbb{R}^m$ have continuous partial derivative functions $\mathbf{x} \mapsto \partial f_i(\mathbf{x})/\partial x_j$ for all $i = 1 \ldots, m$ and $j = 1, \ldots, n$ (i.e., the function $\mathbf{x} \mapsto \nabla f(\mathbf{x})$ exists and is continuous). Then f is differentiable on U.*

If $f : U \to \mathbb{R}^m$ has continuous partial derivatives, we call f a *C^1-function*. In general, C^k denotes the class of functions with continuous partial derivatives up to order k.

The Chain Rule. Let $U \subseteq \mathbb{R}^n$ and $S \subseteq \mathbb{R}^m$ be open sets and assume that the function $f : U \to S$ is differentiable at $\mathbf{x}_0 \in U$. If $g : S \to \mathbb{R}^k$ is differentiable at $\mathbf{y}_0 = f(\mathbf{x}_0)$, the composite function $h : U \to \mathbb{R}^k$, given by $h(\mathbf{x}) = g(f(\mathbf{x}))$, can be linearly approximated at $\mathbf{x}_0$ by the composition of the respective derivatives.

More precisely, $h = g \circ f$ is differentiable at $\mathbf{x}_0$ and the Jacobian of h at $\mathbf{x}_0$ equals the matrix product of the Jacobian matrices of f at $\mathbf{x}_0$ and g at $\mathbf{y}_0$:

$$\boxed{\nabla h(\mathbf{x}_0) = \nabla g(\mathbf{y}_0) \nabla f(\mathbf{x}_0)} \tag{A.10}$$

Formula (A.10) is known as the *chain rule* for differentiable functions.

The Product Rule. The chain rule is an often very useful tool for the computation of derivatives. Let, for example, $f_1, f_2 : (a,b) \to \mathbb{R}$ be differentiable on the open interval $(a,b) \subseteq \mathbb{R}$ and consider $h(t) = f_1(t) \cdot f_2(t)$. Then

$$\boxed{h'(t) = f_2(t)f_1'(t) + f_1(t)f_2'(t)}$$

A.2.4. Second Derivatives and Taylor's Formula. The differentiable function $f : U \to \mathbb{R}$ gives rise to the function $\nabla f : U \to \mathbb{R}^n$ via the assignment $\mathbf{x} \mapsto [\nabla f(\mathbf{x})]^T$. Let us assume that also $\nabla f(\mathbf{x})$ is differentiable. Then the partial derivatives of the partial derivatives of f exist and define the second derivative matrix

$$\nabla^2 f(\mathbf{x}) = \left(\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \right) ,$$

called the *Hessian* matrix of f at $\mathbf{x} \in U$.

If all second partial derivatives are continuous on U, *i.e.*, f is a C^2-function, one can show for all $\mathbf{x} \in U$ and all i, j,

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \;=\; \frac{\partial^2 f(\mathbf{x})}{\partial x_j \partial x_i} ,$$

which means that the Hessian $\nabla^2 f(\mathbf{x})$ is symmetric.

Consider the case where all second partial derivatives of f exist and are continuous. Then Lemma A.14 tells us that ∇f is differentiable. Additionally, $p_{\mathbf{u}}(t) = f(\mathbf{x}_0 + t\mathbf{u})$ is twice differentiable.

In the subsequent discussion, we consider vectors $\mathbf{u}$ of unit length $\|\mathbf{u}\| = 1$. Lemma A.13 guarantees the existence of some $0 < \theta_{\mathbf{u}} < 1$, such that

$$p_{\mathbf{u}}(t) = p_{\mathbf{u}}(0) + p_{\mathbf{u}}'(0)t + \frac{t^2}{2} p_{\mathbf{u}}''(\theta_{\mathbf{u}} t) ,$$

provided $|t| > 0$ is so small that $U_{|t|}(\mathbf{x}_0) \subseteq U$. We want to derive an analogous representation for f.

Given $\varepsilon > 0$, the assumed continuity of the Hessian matrix $\nabla^2 f(\mathbf{x})$ allows us to choose $|t| > 0$ so small that for every $\mathbf{x} \in U$ with $\|\mathbf{x}_0 - \mathbf{x}\| < |t|$,

$$\left| \frac{\partial^2 f(\mathbf{x}_0)}{\partial x_i \partial x_j} - \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \right| < \varepsilon .$$

Recalling $p_{\mathbf{u}}''(t) = \mathbf{u}^T \nabla^2 f(\mathbf{x}_0 + t\mathbf{u})\mathbf{u}$ and observing $|u_i u_j| \leq \|\mathbf{u}\|^2 = 1$ for every two components u_i and u_j of $\mathbf{u}$, we obtain

$$|p_{\mathbf{u}}''(0) - p_{\mathbf{u}}''(\theta_{\mathbf{u}} t)| \leq n^2 \varepsilon ,$$

which is valid for all $\mathbf{d} = t\mathbf{u}$ whenever the norm $\|\mathbf{d}\| = |t|$ is small enough (independent of the unit direction $\mathbf{u}$!). With $p_{\mathbf{u}}'(0) = \nabla f(\mathbf{x}_0)\mathbf{u}$, we thus arrive at *Taylor's formula* for real-valued functions in several variables:

$$\boxed{f(\mathbf{x}_0 + \mathbf{d}) \;=\; f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)\mathbf{d} + \tfrac{1}{2}\mathbf{d}^T \nabla^2 f(\mathbf{x}_0)\mathbf{d} + o(\|\mathbf{d}\|^2)}$$

or with some $\tau \in (0,1)$:

$$\boxed{f(\mathbf{x}_0 + \mathbf{d}) \;=\; f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)\mathbf{d} + \tfrac{1}{2}\mathbf{d}^T \nabla^2 f(\mathbf{x}_0 + \tau\mathbf{d})\mathbf{d}}$$

APPENDIX B

Convex Sets

This appendix is meant as a reference for the reader who needs to freshen up on the concept of convex sets.

Let $P \subseteq \mathbb{R}^n$ be the intersection of (possibly infinitely many) closed halfspaces. (We do allow $P = \emptyset$ and $P = \mathbb{R}^n$ as special cases.) So $\mathbf{y} \notin P$ is true for a given $\mathbf{y} \in \mathbb{R}^n$ if and only if there exists a closed halfspace that contains P but does not contain $\mathbf{y}$, *i.e.*, there exists some $\mathbf{a} \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$ such that for all $\mathbf{x} \in P$,

$$\mathbf{a}^T\mathbf{x} \leq \alpha < \mathbf{a}^T\mathbf{y} \; .$$

In other words, the closed halfspace $H^{\leq} = \{\mathbf{x} \in \mathbb{R}^n \,|\, \mathbf{a}^T\mathbf{x} \leq \alpha\}$ contains the set P but does not contain the vector $\mathbf{y}$ and we say that

$$H = \{\mathbf{x} \in \mathbb{R}^n \,|\, \mathbf{a}^T\mathbf{x} = \alpha\}$$

is a *separating hyperplane* with respect to P and $\mathbf{y}$.

On the other hand, if $S \subseteq \mathbb{R}^n$ is such that for every $\mathbf{y} \notin S$ there exists a separating hyperplane $H_{\mathbf{y}}$ with respect to S and $\mathbf{y}$, then S is the intersection of closed halfspaces, namely the halfspaces $H_{\mathbf{y}}^{\leq}$ associated with the separating hyperplanes:

$$S = \bigcap_{\mathbf{y} \notin S} H_{\mathbf{y}}^{\leq} \; .$$

Summarising, we therefore note:

Lemma B.1. *A set $S \subseteq \mathbb{R}^n$ is an intersection of halfspaces if and only if for each $\mathbf{y} \notin S$ there exists a separating hyperplane with respect to S and $\mathbf{y}$.*

□

Recall that a set P is called *convex* if P contains with any $\mathbf{x}, \mathbf{y} \in P$ also the whole *line segment*

$$[\mathbf{x}, \mathbf{y}] = \{\mathbf{x} + \lambda(\mathbf{y} - \mathbf{x}) \,|\, 0 \leq \lambda \leq 1\} \; .$$

Lemma B.2. *The intersection of closed convex sets is a closed convex set.*

A halfspace $H^{\leq}$ is easily seen to be closed and convex. Hence in view of Lemma B.2, if P is the intersection of closed halfspaces, P will be a closed convex set. It turns out that the converse is also true: Every closed convex set is the intersection of closed halfspaces (Corollary B.4). This fact is an immediate consequence of the existence of separating hyperplanes that we will establish first.

Theorem B.3. *Let $P \subseteq \mathbb{R}^n$ be a non-empty closed convex set. Then for every $\mathbf{y} \notin P$, there exists some $\mathbf{a} \in \mathbb{R}^n$ and $\mathbf{x}_0 \in P$ such that for all $\mathbf{x} \in P$,*

$$\mathbf{a}^T\mathbf{x} \leq \mathbf{a}^T\mathbf{x}_0 < \mathbf{a}^T\mathbf{y} \; .$$

Proof. Choose $R > 0$ so large that $C = \{\mathbf{x} \in P \,|\, \|\mathbf{y} - \mathbf{x}\| \leq R\} \neq \emptyset$. Since C is compact and $\mathbf{x} \mapsto \|\mathbf{y} - \mathbf{x}\|$ is a continuous map, there exists some minimiser $\mathbf{x}_0 \in C$ such that $\|\mathbf{y} - \mathbf{x}_0\| = \min_{\mathbf{x}\in P} \|\mathbf{y} - \mathbf{x}\|$ (*cf.* Theorem A.10).

Consider $\mathbf{a} = \mathbf{y} - \mathbf{x}_0$. Since $\mathbf{x}_0 \in P$ and $\mathbf{y} \notin P$, we know $\mathbf{a} \neq \mathbf{0}$ and find

$$0 < \|\mathbf{a}\|^2 = \mathbf{a}^T(\mathbf{y} - \mathbf{x}_0) = \mathbf{a}^T\mathbf{y} - \mathbf{a}^T\mathbf{x}_0, \quad \textit{i.e.}, \quad \mathbf{a}^T\mathbf{y} > \mathbf{a}^T\mathbf{x}_0 \; .$$

Let $\mathbf{x} \in P$ be arbitrary. It remains to show $\mathbf{a}^T\mathbf{x} \leq \mathbf{a}^T\mathbf{x}_0$. By the convexity of P, $\mathbf{z}(\lambda) = \mathbf{x}_0 + \lambda(\mathbf{x} - \mathbf{x}_0) \in P$ holds for all $0 < \lambda < 1$, which implies

$$\|\mathbf{a} - \lambda(\mathbf{x} - \mathbf{x}_0)\|^2 = \|\mathbf{y} - \mathbf{z}(\lambda)\|^2 \geq \; \|\mathbf{y} - \mathbf{x}_0\|^2 = \|\mathbf{a}\|^2 \; .$$

Multiplying out, we obtain from the latter inequality

$$-2\lambda\mathbf{a}^T(\mathbf{x} - \mathbf{x}_0) + \lambda^2\|\mathbf{x} - \mathbf{x}_0\|^2 \geq \; 0 \; .$$

Dividing now by 2λ and then considering $\lambda \to 0$, shows $\mathbf{a}^T(\mathbf{x} - \mathbf{x}_0) \leq 0$.

□

A separating hyperplane that contains some point $\mathbf{x}_0 \in P$ is said to be *supporting* P in $\mathbf{x}_0$. Hence Theorem B.3 actually says that a closed convex set is the intersection of the halfspaces associated with its supporting hyperplanes. Combined, Lemma B.1 and Theorem B.3 lead to the following characterisation of closed convex sets.

Corollary B.4. *The set $P \subseteq \mathbb{R}^n$ is closed and convex if and only if P is the intersection of (closed) halfspaces.*

□

Recall that $\mathbf{x}_0 \in P$ is a *boundary point* of P if for every $\varepsilon > 0$ we can find some $\mathbf{y} \notin P$ such that $\|\mathbf{y} - \mathbf{x}_0\| < \varepsilon$. Our next observation complements Theorem B.3 and exhibits every boundary point of a closed convex set P as an optimal solution for the problem of optimising some suitable non-trivial linear function $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x}$ over P. Geometrically, to each boundary point $\mathbf{x}_0$ of P there exists a supporting hyperplane.

Theorem B.5. *Let $P \subseteq \mathbb{R}^n$ be closed and convex. Then for every boundary point $\mathbf{x}_0 \in P$, there exists some vector $\mathbf{c} \neq \mathbf{0}$ in $\mathbb{R}^n$ such that*

$$\mathbf{c}^T\mathbf{x}_0 = \max_{\mathbf{x}\in P} \; \mathbf{c}^T\mathbf{x} \; ,$$

i.e. a hyperplane $H = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{c}^T\mathbf{x} = \mathbf{c}^T\mathbf{x}_0\}$ that supports P at $\mathbf{x}_0$.

Proof. Since $\mathbf{x}_0$ is a boundary point of P there exists a sequence $\mathbf{y}_k \to \mathbf{x}_0$ of points $\mathbf{y}_k \notin P$. By Theorem B.3, we can find, for every k, a separating hyperplane $H_k = \{\mathbf{x} \in \mathbb{R}^n \,|\, \mathbf{a}_k^T\mathbf{x} = \alpha_k\}$, such that for all $\mathbf{x} \in P$,

$$\mathbf{a}_k^T\mathbf{x} \leq \alpha_k < \mathbf{a}_k^T\mathbf{y}_k \; .$$

We can assume without loss of generality that $\|\mathbf{a}_k\| = 1$ holds (otherwise, we simply divide each $\mathbf{a}_k$ by its length $\|\mathbf{a}_k\|$). Now the points $\mathbf{a}_k$ form an infinite subset of the compact set $S = \{\mathbf{x} \in \mathbb{R}^n \,|\, \|\mathbf{x}\| = 1\}$ and, therefore, admit a convergent subsequence $\mathbf{a}_{k_i} \to \mathbf{c}$ with $\mathbf{c} \in S$ (*cf.* Section A.2.1). Letting $i \to \infty$ and recalling $\mathbf{y}_{k_i} \to \mathbf{x}_0$, we find for every (fixed) $\mathbf{x} \in P$,

$$\mathbf{c}^T\mathbf{x} \leq \mathbf{c}^T\mathbf{x}_0 \; .$$

□

Index