Dauphine | PSL
UNIVERSITÉ PARIS

Optimization for Machine Learning

Clément W. Royer

Lecture notes - M2 MIAGE ID Apprentissage - 2024/2025

- The last version of these notes can be found at:
  https://www.lamsade.dauphine.fr/∼croyer/ensdocs/OID/PolyOID.pdf.
- Comments, typos, etc, can be sent to `clement.royer@lamsade.dauphine.fr`.
  Thanks to Sébastien Kerleau and Florian Le Bronnec for their feedback on earlier versions of these notes.
- **Version history:**
  - 2024.09.23: First version of the lecture notes with Chapters 1 and 2.
- **Learning goals:**
  - Understand the specifics of optimization problems, and the interest of certain formulations over others.
  - Given an optimization problem, select an algorithm well suited for solving the problem.
  - Analyze the theoretical and practical properties of a particular algorithm.
  - Identify challenges posed by optimization in a data science context, and ways to address these challenges.

Note: This course was previously taught in M2 MIAGE ID Apprentissage under the name "Optimization for Data and Decision Sciences".

Contents

Chapter 1

Introduction to optimization

1.1 About optimization

Optimization is a field of study that is concerned with making the best possible decision out of a set of alternatives. *Mathematical* optimization provides a framework to model optimization problems using mathematical language. We provide below a broad definition of optimization problems in the mathematical sense, and connect it to the concept of solving such a problem.

1.1.1 The optimization process

Most optimization problems are readily formulated in plain language, and it can be difficult to extract the core ideas behind the optimization task. On the contrary, a mathematical optimization problem is a well-identified object, that corresponds to the following definition.

Definition 1.1.1 *A mathematical optimization problem consists of three key components:*

- *An* **objective function***, that quantifies the quality of a decision. If a better decision yields a smaller objective value, the problem is called a* minimization *problem. On the other hand, if a better decision yields a larger objective value, the problem is called a* maximization *problem.*
- *A number of* **decision variables***: those are the knobs that can be turned to change the decision and, as a result, the value of the objective function. The goal of optimization is to determine what are the best (optimal) values of the decision variables relatively to the objective function.*
- *A set of* **constraints***, that specify requirements on the decision variables that must be satisfied for the decision to be valid.*

An optimization procedure consists in minimizing or maximizing an objective function with respect to the decision variables subject to constraints on those variables.

Remark 1.1.1 *In practice, many optimization problems are subject to hidden or implicit constraints, that may not appear in the optimization formulation yet can have a significant effect on an algorithm's performance. The absence of these constraints in the formulation may be due to misspecification (not including a positivity constraint for a variable corresponding to a mass, for instance), but this phenomenon often has a more subtle cause. In simulation-based optimization, where the objective function corresponds to running expensive computational codes, an error may be triggered by certain*

values of the decision variables that may not be known a priori. *Such constraints must then be addressed by an algorithm in real time.*

Any concrete problem goes through a **modeling phase** that produces a mathematical optimization problem. In order to solve this problem, we then use **optimization algorithms**, often designed to be implemented and run on a computer. Such methods typically output a candidate solution (or, possibly, that no solution exists), which may or may not form an appropriate answer to the problem. If needed, the optimization process can go over another modeling phase followed by another solve. In practice, it is typical that the optimization process includes a **discussion phase** with experts from the application domain.

Remark 1.1.2 *Numerical optimization typically obeys the following principles:*

- *There is no universal algorithm. Every method can be efficient on a certain class of problems and perform poorly on another class. Studying the structure of a problem of interest helps in choosing the best algorithm for solving this problem.*

- *There may be a big gap between theory and practice. Finite-precision arithmetic can introduce round-off errors that result in worse practical performance than what the theory predicts. Conversely, theory (such as complexity bounds introduced below) can be overly pessimistic, and much better results can be observed in practice.*

- *Theory informs practice, and vice-versa. For most optimization problems, one can use mathematical formulas to assess whether a solution has been found: these expressions are at the heart of most of the methods we will describe in this course. More broadly, theoretical guarantees can help guiding the optimization process beyond pure heuristics and guesses. Meanwhile, some algorithms have demonstrated success without being endowed with theoretical properties: such phenomena are common, and motivate researchers to explain this behavior using mathematical tools.*

1.1.2 Modern optimization

Numerical optimization started during the 1940s. Major theoretical advances were achieved during the 1980s, along with significant algorithmic developments, that leverage the limited computing power of this era. The next two decades saw the rise of computing power, leading to numerous successes of optimization techniques. This trend continues to this day, albeit with a key change of paradigm.

Indeed, as data becomes more prevalent, optimization problems now involve massive amounts of data in the calculation of their objective. In addition to the challenges posed by the use of such datasets, possibly in a distributed fashion, other difficulties arise due to the fact that the solutions of these problems must generalize to yet unseen data.

In this course, we will go on a *tour d'horizon* of optimization problems and algorithms, with a focus on methods that have been successful in classical industrial settings as well as new paradigms such as machine learning. Our presentation will cover theoretical, algorithmic and practical aspects.

1.2 The optimization problem

We now introduce the mathematical foundations behind optimization. As a field of study, optimization is defined as the process of making the best decision out of a set of alternatives.

Mathematically, we write an optimization problem using three components:

- An **objective function**, i.e. a criterion that measures how good a given decision is, that we want to minimize or maximize depending on the context;
- **Decision variables**, that represent the knobs we can turn to change the decision;
- **Constraints**, i.e. conditions that the decision variables must satisfy in order for the decision to be acceptable.

The general form of the optimization problems considered in these notes will be the following

$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{minimize}}\, f(\boldsymbol{w}) \quad \text{subject to} \quad \boldsymbol{w} \in \mathcal{F}. \tag{1.2.1}$$

In problem (1.2.1), f is the objective function (that we want to minimize), $\boldsymbol{w}$ is the vector of decision variables defined over a definition set $\mathbb{R}^d$, and $\mathcal{F}$ is a set encompassing all the constraints on the decision variables. This set is called the feasible set, and is often described using mathematical expressions.

There are multiple ways of formulating an optimization problem given a description of this problem in plain language: constraints can be expressed in the definition set or in the objective, maximization can be preferred over minimization, redundant constraints can be added to the problem, etc. Some formulations will be well suited for theoretical analysis, while others will be efficiently solvable by modern software.

Definition 1.2.1 *Two mathematical optimization problems are called* equivalent *if the solution of one is readily obtained from the solution of the other, and vice-versa.*

For instance, the minimization problem (1.2.1) has an equivalent reformulation as a maximization problem when $f : \mathbb{R}^d \to \mathbb{R}$ (i.e. f outputs numerical values):

$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{maximize}} -f(\boldsymbol{w}) \quad \text{subject to} \quad \boldsymbol{w} \in \mathcal{F}.$$

1.2.1 Mathematical background

Optimization draws from several fields of mathematics, mostly pertaining to linear algebra, topology and differential calculus. We briefly review the key definitions below.

We will always consider $\mathbb{R}^d$ and $\mathbb{R}^{n\times d}$ as endowed with their canonical normed vector space structure; in particular, this means that we will be able to add two vectors (or two matrices), and to multiply a vector (or a matrix) by a scalar value. It also implies that we can measure the distance between two elements of these spaces using norms, the most classical of which is defined below.

Definition 1.2.2 (Euclidean norm on $\mathbb{R}^d$) *The Euclidean norm (or ℓ_2 norm) of a vector $\boldsymbol{w} \in \mathbb{R}^d$ is given by:*

$$\|\boldsymbol{w}\| := \sqrt{\sum_{i=1}^{d} w_i^2}.$$

Definition 1.2.3 (Scalar product on $\mathbb{R}^d$) *The scalar product is defined for every $\boldsymbol{w}, \boldsymbol{z} \in \mathbb{R}^d$ by:*

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{z} := \sum_{i=1}^{d} w_i\, z_i.$$

One thus has $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{z} = \boldsymbol{z}^{\mathrm{T}}\boldsymbol{w}$ and $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} = \|\boldsymbol{w}\|^2$.

There are natural counterparts to the Euclidean norm and its associated scalar product in a matrix space.

Definition 1.2.4 (Frobenius norm) *For any matrix $\boldsymbol{A} \in \mathbb{R}^{n\times d}$, the Frobenius norm is defined by*

$$\|\boldsymbol{A}\|_F := \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{d} A_{ij}^2} = \sqrt{\mathrm{trace}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})}.$$

The associated scalar product is the mapping

$$(\boldsymbol{A}, \boldsymbol{B}) \in \left(\mathbb{R}^{n\times d}\right)^2 \longmapsto \mathrm{trace}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{B}).$$

In this course, we will mainly rely on Euclidean norms to measure distance between vectors, and algorithmic behavior. However, other norms will be used.

Definition 1.2.5 (ℓ_1 norm) *The ℓ_1 norm of a vector $\boldsymbol{w} \in \mathbb{R}^d$ is defined by:*

$$\|\boldsymbol{w}\|_1 := \sum_{i=1}^{d} |w_i|.$$

Both the ℓ_1 and ℓ_2 norms are special cases of the ℓ_p norm, defined for any $p \in [1, \infty)$ by

$$\forall \boldsymbol{w} \in \mathbb{R}^d, \quad \|\boldsymbol{w}\|_p := \left[\sum_{i=1}^{d} |w_i|^p\right]^{1/p}.$$

Definition 1.2.6 (ℓ_∞ norm) *The ℓ_∞ norm of a vector $\boldsymbol{w} \in \mathbb{R}^d$ is defined by:*

$$\|\boldsymbol{w}\|_\infty := \max_{1\le i\le d} |w_i|.$$

Remark 1.2.1 *Interestingly, all norms can be defined as optimal values of optimization problems. This is clear from the definition of the ℓ_∞ norm. Below are two optimization problems that yield this optimal value:*

$$\underset{i\in\mathbb{R}}{\text{maximize}}\, |w_i| \quad \textit{s.t.} \quad i \in \{1, \dots, d\}.$$

$$\underset{v\in\mathbb{R}}{\text{maximize}}\, |v| \quad \textit{s.t.} \quad v \in \{w_1, \dots, w_d\}.$$

Notice that these two problems do not have the same decision variables or constraints, however they both possess a combinatorial structure. [1]

[1] By convention, in these notes, we will always consider optimization problems with real, continuous decision variables. As illustrated by the ℓ_∞ examples, it is indeed possible to model a discrete decision-making problem using continuous formulations. Our focus in these notes in on continuous optimization problems.

Definition 1.2.7 (Matrix inversion) *A matrix $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ is* invertible *if it exists $\boldsymbol{B} \in \mathbb{R}^{d\times d}$ such that $\boldsymbol{BA} = \boldsymbol{AB} = \mathbf{I}_d$, where $\mathbf{I}_d$ is the identity matrix of $\mathbb{R}^{d\times d}$.*

In this case, $\boldsymbol{B}$ is the unique matrix with this property: $\boldsymbol{B}$ is called the inverse matrix of $\boldsymbol{A}$*, and is denoted by $\boldsymbol{A}^{-1}$.*

Definition 1.2.8 (Positive (semi-)definiteness) *A symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ is* positive semidefinite *if*

$$\forall \boldsymbol{x} \in \mathbb{R}^n, \quad \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} \geq 0.$$

It is called positive definite *when $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} > 0$ for every nonzero vector $\boldsymbol{x}$.*

Definition 1.2.9 (Eigenvalues and eigenvectors) *Let $\boldsymbol{A} \in \mathbb{R}^{d\times d}$. A real λ is called an* eigenvalue of $\boldsymbol{A}$ *if*

$$\exists \boldsymbol{v} \in \mathbb{R}^d, \|\boldsymbol{v}\| \neq 0, \qquad \boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{v}.$$

The vector $\boldsymbol{v}$ is then called an eigenvector of $\boldsymbol{A}$ *associated to the eigenvalue λ.*

Theorem 1.2.1 *Any symmetric matrix in $\mathbb{R}^{d\times d}$ possesses d real eigenvalues.*

Notation 1.2.1 *Given two symmetric matrices $(\boldsymbol{A}, \boldsymbol{B}) \in \mathbb{R}^{d\times d}$, we introduce the following notations:*

- $\lambda_{\min}(\boldsymbol{A})/\lambda_{\max}(\boldsymbol{A})$*: smallest/largest eigenvalue of $\boldsymbol{A}$;*
- $\boldsymbol{A} \succeq \boldsymbol{B} \;\Leftrightarrow\; \lambda_{\min}(\boldsymbol{A}) \geq \lambda_{\max}(\boldsymbol{B})$*;*
- $\boldsymbol{A} \succ \boldsymbol{B} \Leftrightarrow \lambda_{\min}(\boldsymbol{A}) > \lambda_{\max}(\boldsymbol{B})$.

Following these notations, a matrix $\boldsymbol{A}$ is positive semi-definite (resp. positive definite) if and only if $\boldsymbol{A} \succeq 0$ (resp. $\boldsymbol{A} \succ 0$).

Differential calculus We will mostly consider minimization problems involving a smooth objective function: the term "smooth" can be loosely defined in the optimization or learning literature, but generally means that the function is as regular as needed for the desired algorithms and analysis to be applicable. In these notes, we will consider that a smooth function is at least continuously differentiable.

Definition 1.2.10 (Continuous function) *A function $f : \mathbb{R}^d \to \mathbb{R}^m$ is continuous at $\boldsymbol{w} \in \mathbb{R}^d$ if for every $\epsilon > 0$, it exists $\delta > 0$ such that*

$$\forall \boldsymbol{v} \in \mathbb{R}^d, \ \|\boldsymbol{v} - \boldsymbol{w}\| \leq \delta \implies \|f(\boldsymbol{v}) - f(\boldsymbol{w})\| \leq \epsilon.$$

Definition 1.2.11 (Lipschitz continuous function) *A function $f : \mathbb{R}^d \to \mathbb{R}^m$ is L-Lipschitz continuous over $\mathbb{R}^d$ if*

$$\forall (\boldsymbol{u}, \boldsymbol{v}) \in \left(\mathbb{R}^d\right)^2, \quad \|f(\boldsymbol{u}) - f(\boldsymbol{v})\| \leq L\,\|\boldsymbol{u} - \boldsymbol{v}\|,$$

where $L > 0$ is called a Lipschitz constant.

Note that every Lipschitz continuous function is continuous.

Derivatives are ubiquitous in continuous optimization, as they allow to characterize the local behavior of a function. We assume that the reader is familiar with the concept of derivative of a function from $\mathbb{R} \to \mathbb{R}$. A function $f : \mathbb{R}^d \to \mathbb{R}$ is called *differentiable* at $\boldsymbol{w} \in \mathbb{R}^d$ if all its partial derivatives at $\boldsymbol{w}$ exist.

Definition 1.2.12 (Classes of functions)
- *A function $f : \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable if its first-order derivative exists and is continuous. The set of continously differentiable functions is denoted by $\mathcal{C}^1(\mathbb{R}^d)$.*
- *A function $f : \mathbb{R}^d \to \mathbb{R}$ is twice continuously differentiable if $f \in \mathcal{C}^1(\mathbb{R}^d)$, its second-order derivative of f exists and is continuous. The set of twice continously differentiable functions is denoted by $\mathcal{C}^2(\mathbb{R}^d)$.*

Definition 1.2.13 (First-order derivative) *Let $f \in \mathcal{C}^1(\mathbb{R}^d)$ be a continuously differentiable function. For any $\boldsymbol{w} \in \mathbb{R}^d$, the* **gradient of f at $\boldsymbol{w}$** *is given by*

$$\nabla f(\boldsymbol{w}) := \left[\frac{\partial f}{\partial w_i}(\boldsymbol{w})\right]_{1 \leq i \leq d} \in \mathbb{R}^d.$$

Definition 1.2.14 (Second-order derivative) *Let $f \in \mathcal{C}^2(\mathbb{R}^d)$ be a twice continuously differentiable function. For any $\boldsymbol{w} \in \mathbb{R}^d$, the* **Hessian of f at $\boldsymbol{w}$** *is given by*

$$\nabla^2 f(\boldsymbol{w}) := \left[\frac{\partial^2 f}{\partial w_i \partial w_j}(\boldsymbol{w})\right]_{1 \leq i,j \leq d} \in \mathbb{R}^{d \times d}.$$

The Hessian matrix is symmetric.

Finally, we define an important class of problems involving a Lipschitz continuity assumption.

Definition 1.2.15 (Smooth functions with Lipschitz derivatives)
- *Given $L > 0$, the set $\mathcal{C}_L^{1,1}(\mathbb{R}^d)$ represents the set of all functions $f : \mathbb{R}^d \to \mathbb{R}$ that belong to $\mathcal{C}^1(\mathbb{R}^d)$ such that ∇f is L-Lipschitz continuous.*
- *Given $L > 0$, the set $\mathcal{C}_L^{2,2}(\mathbb{R}^d)$ represents the set of all functions $f : \mathbb{R}^d \to \mathbb{R}$ that belong to $\mathcal{C}^2(\mathbb{R}^d)$ such that $\nabla^2 f$ is L-Lipschitz continuous.*

An important property of such functions is that one can derive upper approximations on their values, as shown by the following theorem.

Theorem 1.2.2 (First-order Taylor expansion) *Let $f \in C_L^{1,1}(\mathbb{R}^d)$ with $L > 0$. For any vectors $\boldsymbol{w}, \boldsymbol{z} \in \mathbb{R}^d$, one has:*

$$f(\boldsymbol{z}) \leq f(\boldsymbol{w}) + \nabla f(\boldsymbol{w})^{\mathrm{T}}(\boldsymbol{z} - \boldsymbol{w}) + \frac{L}{2}\|\boldsymbol{z} - \boldsymbol{w}\|^2. \tag{1.2.2}$$

1.2.2 First definitions

Having defined an optimization problem as a mathematical object, we are now able to consider solving this problem. For simplicity, we will focus on minimization problems of the form:

$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{minimize}}\, f(\boldsymbol{w}) \quad \text{s.t.} \boldsymbol{w} \in \mathcal{F}, \tag{1.2.3}$$

with $f : \mathbb{R}^d \to \mathbb{R}$ and $\mathcal{F} \subseteq \mathbb{R}^d$. Similar definitions hold for maximization problems.

Definition 1.2.16 (Feasible point) *A point $\boldsymbol{w} \in \mathbb{R}^d$ is called a feasible point of the optimization problem (1.2.3) if $\boldsymbol{w} \in \mathcal{F}$.*

If $\boldsymbol{w} \notin \mathcal{F}$, we say that $\boldsymbol{w}$ is infeasible. If $\mathcal{F}$ is empty, the problem (1.2.3) is said to be infeasible. For some optimization problems, the objective function may not be defined at infeasible points.

A solution of problem (1.2.3) must be a feasible point by definition. It should also lead to the best value in terms of cost function, hence the following definition.

Definition 1.2.17 (Global minimum) *A point $\boldsymbol{w}^* \in \mathbb{R}^d$ is called a solution or a global minimum of problem (1.2.3) if*

1. *$\boldsymbol{w}^*$ is feasible, i.e. $\boldsymbol{w}^* \in \mathcal{F}$;*
2. *$f(\boldsymbol{w}^*) \leq f(\boldsymbol{w})$ for any feasible point $\boldsymbol{w} \in \mathcal{F}$.*

The set of global minima of problem (1.2.3) will be denoted by

$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{argmin}} \{f(\boldsymbol{w}) \mid \boldsymbol{w} \in \mathcal{F}\} \subset \mathbb{R}^d, \tag{1.2.4}$$

and the minimal value of the problem will be defined as

$$\min_{\boldsymbol{w}\in\mathbb{R}^d} \{f(\boldsymbol{w}) \mid \boldsymbol{w} \in \mathcal{F}\} \in \mathbb{R} \cup \{-\infty, +\infty\}. \tag{1.2.5}$$

By convention, if f is unbounded below on $\mathcal{F} \neq \emptyset$, the minimal value is set to $-\infty$, while it is set to $+\infty$ if the problem is infeasible.

Definition 1.2.18 (Local minimum) *A point $\boldsymbol{w}^* \in \mathbb{R}^d$ is called a local minimum of problem (1.2.3) if*

1. *$\boldsymbol{w}^*$ is feasible, i.e. $\boldsymbol{w}^* \in \mathcal{F}$;*
2. *There exists $\epsilon > 0$ such that $f(\boldsymbol{w}^*) \leq f(\boldsymbol{w})$ for any feasible point $\boldsymbol{w} \in \mathcal{F}$ close to $\boldsymbol{w}^*$ in the sense that $\|\boldsymbol{w} - \boldsymbol{w}^*\| \leq \epsilon$.*

The notion of local minimum is weaker than that of global minimum. In practice, however, it is often more reasonable to seek local minima, as those can be characterized by mathematical conditions.

Optimality conditions In general, finding global or even local minima is a hard problem. For this reason, researchers in optimization have developed optimality conditions. These are mathematical expressions that can be checked at a given point (unlike the conditions above) and help assessing whether a given point is a local minimum or not.

$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{minimize}}\, f(\boldsymbol{w}), \tag{1.2.6}$$

Theorem 1.2.3 (First-order necessary condition) *Suppose that the objective function f in problem (1.2.6) belongs to $\mathcal{C}^1(\mathbb{R}^d)$. Then,*

$$[\,\boldsymbol{w}^* \text{ is a local minimum of } f\,] \implies \|\nabla f(\boldsymbol{w}^*)\| = 0. \tag{1.2.7}$$

Note that this condition is only necessary: there may exist points with zero gradient that are not local minima. Indeed, the set of points with zero gradient, called *first-order stationary points*, also includes local maxima and saddle points[2].

Provided we strengthen our smoothness requirements on f, we can establish stronger optimality conditions for problem (1.2.6).

Theorem 1.2.4 (Second-order necessary condition) *Suppose that the objective function f in problem (1.2.6) belongs to $\mathcal{C}^2(\mathbb{R}^d)$. Then,*

$$[\,\boldsymbol{w}^* \text{ is a local minimum of } f\,] \implies \left[\|\nabla f(\boldsymbol{w}^*)\| = 0 \quad \text{and} \quad \nabla^2 f(\boldsymbol{w}^*) \succeq \mathbf{0}\right]. \tag{1.2.8}$$

From Theorem 1.2.3, first-order stationary points that violate the condition $\nabla^2 f(\boldsymbol{w}^*) \succeq \mathbf{0}$ cannot be local minima. This new condition is thus more precise than the first-order condition, although it remains a necessary. For an arbitrary function, there may exist points with zero gradient and positive semidefinite Hessian (termed **second-order stationary points**) that are not local minima.

On the other hand, and unlike first-order conditions, it is possible to derive a *sufficient* version of the second-order optimality conditions, that can be used to certify local optimality.

Theorem 1.2.5 (Second-order sufficient condition) *Suppose that the objective function f in problem (1.2.6) belongs to $\mathcal{C}^2(\mathbb{R}^d)$. Then,*

$$\left[\|\nabla f(\boldsymbol{w}^*)\| = 0 \quad \text{and} \quad \nabla^2 f(\boldsymbol{w}^*) \succ \mathbf{0}\right] \implies [\,\boldsymbol{w}^* \text{ is a local minimum of } f\,] \tag{1.2.9}$$

By exploiting the second-order derivative, it is thus possible to certify whether a point is a local minima (note that there could be local minima such that $\nabla^2 f(\boldsymbol{w}^*) \succeq \mathbf{0}$). With further assumptions on the structure of the problem, these optimality conditions can be more informative about minima. This is the case when the objective function is convex: we detail this property in the next section.

1.2.3 Convexity

Convexity is at its core a geometric notion: before defining what a convex function is, we describe the corresponding property for a set.

[2] A vector is a saddle point of a function if it is a local minimum with respect to certain directions and a local maximum with respect to other directions of the space.

Definition 1.2.19 (Convex set) *A set $\mathcal{C} \in \mathbb{R}^d$ is called* **convex** *if*

$$\forall (\boldsymbol{u}, \boldsymbol{v}) \in \mathcal{C}^2, \ \forall t \in [0,1], \qquad t\boldsymbol{u} + (1-t)\boldsymbol{v} \in \mathcal{C}.$$

Example 1.2.1 (Examples of convex sets) *The following sets are convex:*

- $\mathbb{R}^d$;
- *Every line segment of the form* $\{t\boldsymbol{w} | t \in \mathbb{R}\}$ *for some* $\boldsymbol{w} \in \mathbb{R}^d$;
- *Every (Euclidean) ball of the form* $\left\{ \boldsymbol{w} \in \mathbb{R}^d \,\middle|\, \|\boldsymbol{w}\|_2^2 = \sum_{i=1}^d [\boldsymbol{w}]_i^2 \leq 1 \right\}$.

We now provide the basic definition of a convex function.

Definition 1.2.20 (Convex function) *A function $f : \mathbb{R}^d \to \mathbb{R}$ is* **convex** *if*

$$\forall (\boldsymbol{u}, \boldsymbol{v}) \in (\mathbb{R}^d)^2, \ \forall t \in [0,1], \qquad f(t\boldsymbol{u} + (1-t)\boldsymbol{v}) \leq t\, f(\boldsymbol{u}) + (1-t)\, f(\boldsymbol{v}). \tag{1.2.10}$$

Example 1.2.2 *The following functions are convex :*

- *Linear functions of the form* $\boldsymbol{w} \mapsto \boldsymbol{a}^{\mathrm{T}}\boldsymbol{w} + b$, *with* $\boldsymbol{a} \in \mathbb{R}^d$ *and* $b \in \mathbb{R}$;
- *Squared Euclidean norm:* $\boldsymbol{w} \mapsto \|\boldsymbol{w}\|_2^2 = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}$.

If we consider differentiable functions, it is possible to characterize convexity using the derivatives of the function.

Theorem 1.2.6 *Let $f : \mathbb{R}^d \to \mathbb{R}$ be an element of $\mathcal{C}^1(\mathbb{R}^d)$. Then, the function f is convex if and only if*

$$\forall \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d, \quad f(v) \geq f(u) + \nabla f(u)^{\mathrm{T}}(v-u). \tag{1.2.11}$$

Theorem 1.2.7 *Let $f : \mathbb{R}^d \to \mathbb{R}$ be an element of $\mathcal{C}^2(\mathbb{R}^d)$. Then, the function f is convex if and only if*

$$\forall \boldsymbol{w} \in \mathbb{R}^d, \quad \nabla^2 f(\boldsymbol{w}) \succeq \mathbf{0}. \tag{1.2.12}$$

Convex functions are particularly suitable for minimization problems as they satisfy the following property.

Theorem 1.2.8 *If f is a convex function, then every local minimum of f is a global minimum.*

If the function is differentiable, the optimality conditions as well as the characterization of convexity lead us to the following result.

Corollary 1.2.1 *If f is continuously differentiable, every point $\boldsymbol{w}^*$ such that $\|\nabla f(\boldsymbol{w}^*)\| = 0$ is a global minimum of f.*

Strong convexity The results above can be further improved by assuming that a convex function is strongly convex, as defined below.

Definition 1.2.21 (Strongly convex function) *A function $f : \mathbb{R}^d \to \mathbb{R}$ in $\mathcal{C}^1$ is μ-strongly convex (or* strongly convex of modulus $\mu > 0$*) if for all $(\boldsymbol{u}, \boldsymbol{v}) \in (\mathbb{R}^d)^2$ and $t \in [0, 1]$,*

$$f(t\boldsymbol{u} + (1-t)\boldsymbol{v}) \;\leq\; t\, f(\boldsymbol{u}) + (1-t) f(\boldsymbol{v}) - \frac{\mu}{2} t(1-t)\|\boldsymbol{v} - \boldsymbol{u}\|^2.$$

Theorem 1.2.9 *Any strongly convex function has a unique global minimizer.*

As for convex functions, there exist characterizations of strong convexity that involve derivatives.

Theorem 1.2.10 *Let $f : \mathbb{R}^d \to \mathbb{R}$ be an element of $\mathcal{C}^1(\mathbb{R}^d)$. Then, the function f is μ-strongly convex if and only if*

$$\forall \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d, \quad f(\boldsymbol{v}) \geq f(\boldsymbol{u}) + \nabla f(\boldsymbol{u})^{\mathrm{T}}(\boldsymbol{v} - \boldsymbol{u}) + \frac{\mu}{2}\|\boldsymbol{v} - \boldsymbol{u}\|^2. \tag{1.2.13}$$

Theorem 1.2.11 *Let $f : \mathbb{R}^d \to \mathbb{R}$ be an element of $\mathcal{C}^2(\mathbb{R}^d)$. Then, the function f is μ-strongly convex if and only if*

$$\forall \boldsymbol{w} \in \mathbb{R}^d, \quad \nabla^2 f(\boldsymbol{w}) \succeq \mu \mathbf{I}. \tag{1.2.14}$$

1.3 Examples of optimization problems in ML

1.3.1 Linear regression

Linear least squares is arguably the most classical problem in data analysis. We consider a dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ with $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. Our goal is to compute a linear model that best fits (or explains) the data. We define this model as a function $h : \mathbb{R}^d \to \mathbb{R}$, and we parameterize it through a vector $\boldsymbol{w} \in \mathbb{R}^d$, so that for any $\boldsymbol{x} \in \mathbb{R}^d$, we have $h(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}$. For every example $(\boldsymbol{x}_i, y_i)$ in the dataset, we evaluate how we fit the data based on the squared error $(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} - y_i)^2$. We then compute a model by solving the following optimization problem

$$\underset{\boldsymbol{w} \in \mathbb{R}^d}{\text{minimize}} \frac{1}{2n}\|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2. = \frac{1}{n}\sum_{i=1}^{n} \tfrac{1}{2}\left[(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} - y_i)^2 + \lambda\|\boldsymbol{w}\|^2\right], \tag{1.3.1}$$

where $\lambda > 0$ is a regularization parameter. From an optimizer's point of view, problem (1.3.1) is well understood: this is a strongly convex, quadratic problem, and its solution can be computed in close form.

In a typical linear regression setting, one assumes that there exists an underlying truth but that the measurements are noisy, i.e.

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w}^* + \boldsymbol{\epsilon},$$

where $\boldsymbol{\epsilon} \in \mathcal{N}(\mathbf{0}, \mathbf{I})$ is a vector with i.i.d. entries following a standard normal distribution: this is illustrated in Figure 1.1.

In this setting, we wish to compute the most likely value for $\boldsymbol{w}^*$, while being robust to variance in the data. To this end, we suppose that $\boldsymbol{y}$ follows a Gaussian distribution of mean $\boldsymbol{X}\boldsymbol{w}$ and of covariance matrix $\mathbf{I}$. We also assume a prior Gaussian distribution on the entries of $\boldsymbol{w}$, in order to

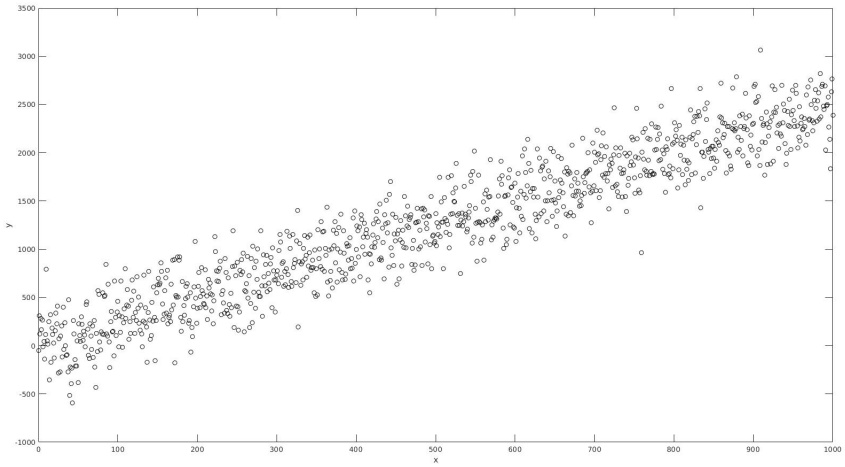

Figure 1.1: Noisy data generated from a linear model with Gaussian noise.

reduce the variance with respect to the data. As a result, an estimate of $\boldsymbol{w}^*$, called the maximum a posteriori estimator, can be computed by solving

$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{maximize}}\, L(y_1,\ldots,y_n;\boldsymbol{w}) := \left[\frac{1}{\sqrt{2\pi}}\right]^m \exp\left(-\frac{1}{2}\sum_{i=1}^{m}(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w}-y_i)^2 - \frac{\lambda}{2}\|\boldsymbol{w}\|^2\right). \tag{1.3.2}$$

The solutions of this maximization problem are the same than the solutions of the linear least-squares problem (1.3.1). The resulting solution can be shown to possess very favorable statistical properties: in particular, for λ close to 0, its expected value is close to $\boldsymbol{w}^*$.

Linear regression (with or without regularization) has been extensively studied in optimization and statistics; however, when the number of samples is extremely large, it still poses a number of challenges in practice, as the solution of the problem cannot be computed exactly.

1.3.2 Logistic regression

As in Section 1.3.1, we consider a dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ are feature vectors, and the y_is represent binary labels. We wish to build a linear classifier $\boldsymbol{x} \mapsto \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$ to perform this classification, i. e. identify the correct label from the feature. We first suppose that $y_i \in \{-1,+1\}$. To model these discrete-valued labels, we introduce an *odds-like* function

$$p(\boldsymbol{x};\boldsymbol{w}) = (1+e^{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}})^{-1} \in (0,1).$$

Given this function, our goal is to choose the model $\boldsymbol{w}$ such that

$$\begin{cases} p(\boldsymbol{x}_i;\boldsymbol{w}) \approx 1 & \text{if } y_i = +1; \\ p(\boldsymbol{x}_i;\boldsymbol{w}) \approx 0 & \text{if } y_i = -1. \end{cases}$$

Given this goal, we want to build an objective function that measures the error between our model and the labels according to the property above. Therefore, we penalize situations in which $y_i = +1$

and $p(\boldsymbol{x}_i; \boldsymbol{w})$ is close to 0, or $y_i = -1$ and $p(\boldsymbol{x}_i; \boldsymbol{w})$ is close to 1. This results in the so-called logistic loss, which is a function from $\mathbb{R}^d$ to $\mathbb{R}$ defined by

$$\forall \boldsymbol{w} \in \mathbb{R}^d, f(\boldsymbol{w}) = \frac{1}{n} \left\{ \sum_{y_i=-1} \ln\left(1 + e^{-\boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{w}}\right) + \sum_{y_i=+1} \ln\left(1 + e^{\boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{w}}\right) \right\}. \tag{1.3.3}$$

The motivation behind introducing the logarithm of the function p is twofold. On the one hand, it provides a statistical interpretation of the loss as a joint distribution; on the other hand, the derivatives of this function have a more favorable structure.

Given this objective function, the **logistic regression** problem is given by

$$\underset{\boldsymbol{w} \in \mathbb{R}^d}{\text{minimize}} \frac{1}{n} \left\{ \sum_{y_i=-1} \ln\left(1 + e^{-\boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{w}}\right) + \sum_{y_i=+1} \ln\left(1 + e^{\boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{w}}\right) \right\} \tag{1.3.4}$$

This is a convex, smooth problem (though not a strongly convex one), that can be made strongly convex by adding a regularizing term (see Chapter **??**).

1.3.3 Linear SVM

To illustrate the role of optimization in data-related applications, we consider a binary classification problem, illustrated in Figure 1.2.

The red circles and blue squares appear at the same locations on all three figures : they represent data samples identified by their coordinates, while their color or shape represents a certain class they belong to. Our goal is to compute a linear classifier, that is, a separator of the two classes corresponding to a linear function. Each of the three figures shows a separator that achieves the task of classifying the data (the separator is the same for the middle and right plots): in that sense, the task involving the *samples* has been performed. However, if we envision the samples as being part of a (much) larger dataset, represented by the blue and red blobs, it becomes clear that the best classifier is the one on the rightmost figure.

These observations can be modeled and summarized using a mathematical framework. Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ be n vectors of $\mathbb{R}^d$, and suppose that each vector $\boldsymbol{x}_i$ is given a label $y(\boldsymbol{x}_i) \in \{-1, 1\}$ (say, -1 for a red point and 1 for a blue point). Then, one can translate the binary classification problem into the following optimization problem:

$$\underset{\substack{\boldsymbol{u} \in \mathbb{R}^d \\ v \in \mathbb{R}}}{\text{minimize}} \frac{1}{n} \sum_{i=1}^{n} \max\left\{1 - y(\boldsymbol{x}_i)(\boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{u} - v), 0\right\} \tag{1.3.5}$$

Here we seek a linear function defined by $\boldsymbol{x} \mapsto \boldsymbol{x}^{\mathrm{T}} \boldsymbol{u} - v$: it thus suffices to compute its coefficients, given by the vector $\boldsymbol{u}$ and the scalar v (see Section **??** for a formal definition of these concepts).

1.3.4 Neural networks

Neural networks have enabled the most impressive, recent advances in perceptual tasks such as image recognition and classification. Thanks to the increase in computational capabilities over the past decade, it is now possible to train extremely deep and wide neural networks, so that they can learn efficient representations of the data.

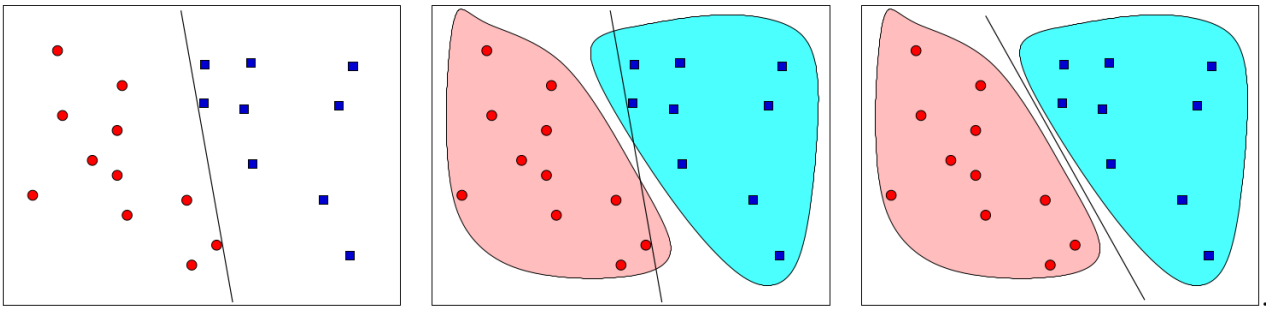

Figure 1.2: A sample for binary classification (red circles/blue squares) and the associated distribution (light red set/light blue set). A same linear classifier is shown on the left and middle plot: although it classifies the samples correctly, its closeness to several data points make it sensitive to the data, and prevents it from correctly classifying the distribution. On the contrary, the linear classifier on the right plot (that has a maximal margin of separation) provides a better classification, and is able to generalize to the distributions. *Source : S. J. Wright and B. Recht, Optimization for Data Analysis [3].*

Given an input vector $\boldsymbol{x}_i \in \mathbb{R}^{d_0}$, a neural network represents a prediction function $h : \mathbb{R}^{d_0} \to \mathbb{R}^{d_J}$, which applies a series of transformations in layers $\boldsymbol{x}_i = \boldsymbol{x}_i^{(0)} \mapsto \boldsymbol{x}_i^{(1)} \mapsto \cdots \mapsto \boldsymbol{x}_i^{(J-1)} \mapsto \boldsymbol{x}_i^{(J)}$. The j-th layer typically performs the following transformation:

$$\boldsymbol{x}_i^{(j)} \;=\; \boldsymbol{\sigma}\left(\boldsymbol{W}_j \boldsymbol{x}_i^{(j-1)} + \boldsymbol{b}_j\right) \in \mathbb{R}^{d_j}, \tag{1.3.6}$$

where $\boldsymbol{W}_j \in \mathbb{R}^{d_j \times d_{j-1}}$, $\boldsymbol{b}_j \in \mathbb{R}^{d_j}$ and $\boldsymbol{\sigma} : \mathbb{R}^{d_j} \to \mathbb{R}^{d_j}$ is a componentwise nonlinear function, e.g. $\boldsymbol{\sigma}(\boldsymbol{y}) = \left[\frac{1}{1+\exp(-y_i)}\right]_i$ (sigmoid function) or $\boldsymbol{\sigma}(\boldsymbol{y}) = [\max(0, y_i)]_i$. As a result, we have $\boldsymbol{x}_i^{(J)} = h(\boldsymbol{x}_i; \boldsymbol{w})$, where $\boldsymbol{w} \in \mathbb{R}^d$ gathers all the parameters $\{(\boldsymbol{W}_1, \boldsymbol{b}_1), \ldots, (\boldsymbol{W}_J, \boldsymbol{b}_J)\}$ of the layers.

The optimization problem corresponding to training this neural network architecture involves a training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ and the choice of a loss function ℓ. It usually results in the following formulation

$$\underset{\boldsymbol{w} \in \mathbb{R}^d}{\text{minimize}} \; \frac{1}{n} \sum_{i=1}^{n} \ell\left(h(\boldsymbol{x}_i; \boldsymbol{w}), y_i\right). \tag{1.3.7}$$

This optimization problem is highly nonlinear and nonconvex in nature, which makes it particularly difficult to solve using algorithms such as gradient descent. Moreover, it typically involves costly algebraic operations, as the number of layers and/or parameters is tremendously large in modern deep neural network architectures. Therefore, problem (1.3.7) also possesses characteristics that are not accounted for in its formulation. The optimization algorithms that efficiently tackle this problem are those that can both guarantee convergence and perform well in practice.

1.4 Optimization algorithms

The field of optimization can be broadly divided into three categories:

- Mathematical optimization is concerned with the theoretical study of complex optimization formulations, and the proof of well-posedness of such problems (for instance, prove that their exist solutions);

- Computational optimization deals with the development of software that can solve a family of optimization problems, through careful implementation of efficient methods;
- **Algorithmic** optimization lies in-between the previous two categories, and aims at proposing new algorithms that address a particular issue, with theoretical guarantees and/or validation of their practical interest.

These notes cover material from the third category of optimization activities. The design of optimization algorithms (also called methods, or schemes) is a particularly subtle process, as an algorithm must exploit the theoretical properties of the problem while being amenable to implementation on a computer.

1.4.1 The algorithmic process

Most numerical optimization algorithms do not attempt to find a solution of a problem in a direct way, and rather proceed in an *iterative* fashion. Given a current point, that represents the current approximation to the solution, an optimization procedure attempts to move towards a (potentially) better point: to this end, the method generally requires a certain amount of calculation.

Suppose we apply such a process to the problem $\min_{\boldsymbol{w}\in\mathbb{R}^d} f(\boldsymbol{w})$, resulting in a sequence of iterates $\{\boldsymbol{w}_k\}_k$. Ideally, these iterates obey one of the scenarios below:

1. The iterates produced get increasingly close to a solution, i. e.

$$\|\boldsymbol{w}_k - \boldsymbol{w}^*\| \to 0 \quad \text{when } k \to \infty.$$

   Although $\boldsymbol{w}^*$ is generally not known in practice, such results can be guaranteed by the theory, for instance on strongly convex problems.

2. The function values associated with the iterates get increasingly close to the optimum, i. e.

$$f(\boldsymbol{w}_k) \to f^* \quad \text{when } k \to \infty,$$

   As for the case above, f^* may not be known, but it can still be possible to prove convergence for certain algorithms and function classes (typically strongly convex, smooth functions).

3. The first-order optimality condition gets close to being satisfied, that is, $f \in \mathcal{C}^1(\mathbb{R}^d)$ and

$$\|\nabla f(\boldsymbol{w}_k)\| \to 0 \quad \text{when } k \to \infty.$$

Out of the three conditions, the last one is the easiest to track as the algorithm unfolds: it is, however, only a necessary condition, and does not guarantee convergence to a local minimum for generic, nonconvex functions. On the other hand, the first two conditions can only be measured approximately (by looking at the behavior of the iterates and enforcing decrease in the function values), but lead to stronger guarantees.

1.4.2 Convergence and convergence rates

The typical theoretical results that optimizers aim at proving for algorithms are asymptotic, as shown above: they only provide a guarantee in the limit. In practice, one may want to obtain more precise guarantees, that relate to a certain accuracy target that the practitioner would like to achieve. This led to the development of **global convergence rates**.

Example 1.4.1 (Global convergence rate for the gradient norm) *Given an algorithm applied to* $\min_{\boldsymbol{w}\in\mathbb{R}^d} f(\boldsymbol{w})$ *that produces a sequence of iterates* $\{\boldsymbol{w}_k\}$*, we say that the method is* $\mathcal{O}(1/k)$ *for the gradient norm, or* $\|\nabla f(\boldsymbol{w}_k)\| = \mathcal{O}\left(\frac{1}{k}\right)$ *if*

$$\exists C > 0, \|\nabla f(\boldsymbol{w}_k)\| \leq \frac{C}{k} \ \forall k.$$

Such rates allow to quantify how much effort (in terms of iterations) is needed to reach a certain target accuracy $\epsilon > 0$. This leads to the companion notion of **worst-case complexity bound**.

Example 1.4.2 (Worst-case complexity for the gradient norm) *Given an algorithm applied to* $\min_{\boldsymbol{w}\in\mathbb{R}^d} f(\boldsymbol{w})$ *that produces a sequence of iterates* $\{\boldsymbol{w}_k\}$*, we say that the method has a worst-case complexity of* $\mathcal{O}\left(\epsilon^{-1}\right)$ *for the gradient norm if*

$$\exists C > 0, \ \|\nabla f(\boldsymbol{w}_k)\| \leq \epsilon \text{ when } k \geq \frac{C}{\epsilon}.$$

Such results are quite common in theoretical computer science or statistics, which partly explain their popularity in machine learning. In optimization, they have been developed for a number of years in the context of convex optimization but have only gained momentum in general optimization over the last decade.

1.4.3 Popular optimization packages

Although a thorough numerical study is out of the scope of this course, we briefly mention popular choices for implementing optimization methods, either for industrial use or as research prototypes.

The most popular programming languages for optimization are C/C++/Fortran for high performance implementations, with Python and Julia raising increasing interest. The use of MATLAB/Octave is also widespread throughout the optimization community for prototyping (i.e. rapid and simple validation of an implementation), along with Python and Julia,

In addition to programming languages, optimizers have developed **modeling** languages that help bringing the code and the mathematical formulation of a problem closer. The broad-spectrum languages GAMS/AMPL/CVX are reknown examples; other languages, that are more domain-oriented, include MATPOWER and PyTorch.

Finally, there are many commercial solvers available, with CPLEX and Gurobi being arguably some of the most efficient for certain classes of problems. Some of those solvers are implemented in standard tools such as Microsoft Excel. Open-source codes are also quite popular, again fueled by the massive production of such implementations in the learning community. As far as optimization is concerned, the COIN-OR platform provides a good interface to all of these methods.

Chapter 2

Unconstrained optimization

In this chapter, we study more general nonlinear optimization problems of the form

$$\min_{\boldsymbol{w}\in\mathbb{R}^d} f(\boldsymbol{w}). \tag{2.0.1}$$

We make the following assumption on the objective function.

Assumption 2.0.1 *The objective function f in (2.0.1) is $\mathcal{C}_L^{1,1}(\mathbb{R}^d)$ for $L > 0$, and bounded below by $f_{low} \in \mathbb{R}$ (i.e. $f(\boldsymbol{w}) \geq f_{low}\ \forall \boldsymbol{w} \in \mathbb{R}^d$).*

We will design and analyze algorithms that exploit gradient information to move towards better points. Our theoretical results will consist in complexity bounds and convergence rates.

2.1 Gradient descent

The gradient descent algorithm is arguably the most classical technique in unconstrained, smooth optimization. It is based on the following principle, derived from the first-order optimality condition (1.2.7).

For any vector $\boldsymbol{w} \in \mathbb{R}^d$, two cases can occur:

1. Either $\nabla f(\boldsymbol{w}) = 0$ and $\boldsymbol{w}$ is possibly a local minimum (when f is convex, we know that $\boldsymbol{w}$ is necessarily a global minimum);
2. Or $\nabla f(\boldsymbol{w}) \neq 0$, and we can show that f must decrease *locally* in the direction of $-\nabla f(\boldsymbol{w})$.

The second property is formalized below, and is at the core of the gradient descent framework.

2.1.1 Algorithm

The gradient descent algorithm is an iterative process wherein every iteration has the following form:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla f(\boldsymbol{w}), \tag{2.1.1}$$

where $\alpha > 0$ is a parameter called stepsize or steplength. When $\nabla f(\boldsymbol{w}) = 0$, note that the formula (2.1.1) does not change the value of $\boldsymbol{w}$: this is consistent with the fact that the gradient

cannot be used to determine a better point in that case. On the contrary, when $\nabla f(\boldsymbol{w}) \neq 0$, there will exist values for α leading to a lower function value for the point $\boldsymbol{w} - \alpha \nabla f(\boldsymbol{w})$.

Repeated applications of the update (2.1.1) lead to Algorithm 1. This method is called **gradient descent**, or (less frequently) *steepest descent*.[1]

Algorithm 1: Gradient descent for minimizing the function f.

Initialization: Choose $\boldsymbol{w}_0 \in \mathbb{R}^d$.
For $k = 0, 1, ...$
1. Compute the gradient $\nabla f(\boldsymbol{w}_k)$.
2. Define a stepsize $\alpha_k > 0$.
3. Set $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \nabla f(\boldsymbol{w}_k)$.

EndFor

Algorithm 1 actually describes a framework rather than a specific method. There exist numerous variants upon the gradient descent paradigm: we review the main characteristics of these methods below.

Stopping criterion In practice, an algorithm is often subject to budget requirements (in terms of arithmetic opterations, running time, iteration number, etc). These limitations are typically enforced using a stopping criterion. In the context of Algorithm 1, it is typical to terminate the method after $k_{\max}$ iterations, in which case $\boldsymbol{w}_{k_{\max}}$ is returned as an approximate solution to the problem.

In addition, stopping criteria can be used to check whether the method converged to a solution of the problem (or an approximation thereof). Such criteria are often based on optimality conditions. For gradient descent, the most classical criterion is based on the gradient norm: one stops the algorithm as soon as

$$\|\nabla f(\boldsymbol{w}_k)\| < \epsilon, \tag{2.1.2}$$

where $\epsilon > 0$ represents a given, desired accuracy level (the condition becomes more expensive to satisfy as ϵ gets smaller).

Finally, it is always possible (and often recommended) to add "safety checks", that stop the method whenever no visible progress is measured. For instance, if the difference between two successive iterates ($\|\boldsymbol{w}_{k+1} - \boldsymbol{w}_k\|$) is of the order of machine precision, the algorithm is likely stalling and no further improvement is expected: in that case, it is often better to stop the method.

Choosing the initial point The performance of a given algorithm can be significantly improved using a suitable starting point. However, finding such a point can be a difficult task without domain expertise, but if such expertise exists, it should be leveraged to obtain a good initial point that the method tries to improve upon. Alternatively, one could use several starting points drawn at random and run a few iterations of gradient descent so as to determine a good starting point.

[1] The direction of $-\nabla f(\boldsymbol{w}_k)$, i.e. the unit vector $\frac{-\nabla f(\boldsymbol{w}_k)}{\|\nabla f(\boldsymbol{w}_k)\|}$ (or the zero vector if the gradient is zero) is called the steepest descent direction.

2.1.2 Choosing the stepsize

In this section, we describe the main techniques for selecting the step size sequence in gradient descent. We provide generic principles, and emphasize that any information about a particular problem can be extremely valuable in designing a better step size.

Constant step size One of the most common approaches consists in fixing the step size to a single value for all iterations, i.e. setting $\alpha_k = \alpha > 0$ for all k. Depending on the computational budget, one could run gradient descent with several values of α and select the best value for future use: this practice of *tuning* the step size is commonly adopted in data science problems.

Provided f satisfies Assumption 2.0.1, there exists an interval of values that lead to convergence of gradient descent. In particular, the choice

$$\alpha_k = \alpha = \tfrac{1}{L}, \tag{2.1.3}$$

where L is the Lipschitz constant for the gradient, is well suited for that problem. Note however that this choice requires the knowledge of the Lipschitz constant: this information is not always available in practice.

Decreasing step size Another classical technique for selecting the stepsize consists in defining a decreasing sequence $\{\alpha_k\}$ such that $\alpha_k \to 0$ *prior to running the method*. This choice can also lead to converging method, but it risks producing steps that are unnecessarily small in norm. In fact, a good decreasing strategy should drive α_k to 0 quickly enough for convergence, but slowly enough that the norm of the steps do not approach 0 too rapidly.

Adaptive choice using a line search Line-search techniques are widely used in continuous optimization and scientific computing (though less popular in data science, for reasons that we will detail in Chapter 3 of these notes). At a given iteration of index k, we seek a stepsize α_k that leads to a decrease in the objective function along a suitably chosen direction (in the case of Algorithm 1, this would be the direction of $-\nabla f(\boldsymbol{w}_k)$). An exact line search results in the best possible decrease, but may be costly to perform. In practice, *inexact* approaches are preferred: Algorithm 2 details the most popular of such techniques, called backtracking.

Algorithm 2: Backtracking line search in direction d.

Inputs: $\boldsymbol{w} \in \mathbb{R}^d$, $\boldsymbol{d} \in \mathbb{R}^d$, $\alpha_0 \in \mathbb{R}^d$.
Initialization: Choose $\alpha = \alpha_0$.
While $f(\boldsymbol{w} + \alpha \boldsymbol{d}) > f(\boldsymbol{w})$
  $\alpha \to \frac{\alpha}{2}$.
End
Output: α.

The line-search procedure of Algorithm 2 can be used at Step 2 of Algorithm 1 using $\boldsymbol{w} = \boldsymbol{w}_k$, $\boldsymbol{d} = -\nabla f(\boldsymbol{w}_k)$ and (for instance) $\alpha_0 = 1$ as inputs. Many variants of this simple idea have been proposed in the literature: those are generally designed to guarantee that a better point (in terms of the objective function) is found. Still, these techniques require at least one additional function evaluation per stepsize value (possibly more), leading to an overall more expensive method.

2.1.3 Theoretical analysis for gradient descent

In this section, we present several convergence rates for gradient descent, in the case of a smooth objective function. We will see that the nonconvex, convex and strongly convex cases exhibit different behavior.

Proposition 2.1.1 *Consider the k-th iteration of Algorithm 1 applied to $f \in \mathcal{C}_L^{1,1}(\mathbb{R}^d)$, and suppose that $\nabla f(\boldsymbol{w}_k) \neq 0$. Then, if $0 < \alpha_k < \frac{2}{L}$, we have*

$$f(\boldsymbol{w}_k - \alpha_k \nabla f(\boldsymbol{w}_k)) < f(\boldsymbol{w}_k).$$

In particular, choosing $\alpha_k = \frac{1}{L}$ leads to

$$f(\boldsymbol{w}_k - \frac{1}{L}\nabla f(\boldsymbol{w}_k)) < f(\boldsymbol{w}_k) - \frac{1}{2L}\|\nabla f(\boldsymbol{w}_k)\|^2. \tag{2.1.4}$$

Proof. We use the inequality (1.2.2) with the vectors $(\boldsymbol{w}_k, \boldsymbol{w}_k - \alpha_k \nabla f(\boldsymbol{w}_k))$:

$$\begin{aligned} f(\boldsymbol{w}_k - \alpha_l \nabla f(\boldsymbol{w}_k)) &\leq f(\boldsymbol{w}_k) + \nabla f(\boldsymbol{w}_k)^{\mathrm{T}} [-\alpha_k \nabla f(\boldsymbol{w}_k)] + \frac{L}{2}\| - \alpha_k \nabla f(\boldsymbol{w}_k)\|^2 \\ &= f(\boldsymbol{w}_k) - \alpha_k \nabla f(\boldsymbol{w}_k)^{\mathrm{T}} \nabla f(\boldsymbol{w}_k) + \frac{L}{2}\alpha_k^2 \|\nabla f(\boldsymbol{w}_k)\|^2 \\ &= f(\boldsymbol{w}_k) + \left(-\alpha_k + \frac{L}{2}\alpha_k^2\right) \|\nabla f(\boldsymbol{w}_k)\|^2. \end{aligned}$$

If $-\alpha_k + \frac{L}{2}\alpha_k^2 < 0$, the second term on the right-hand side will be negative, thus we will have $f(\boldsymbol{w}_k - \alpha_l \nabla f(\boldsymbol{w}_k)) < f(\boldsymbol{w}_k)$. Since $-\alpha_k + \frac{L}{2}\alpha_k^2 < 0 \Leftrightarrow \alpha_k < \frac{2}{L}$ and $\alpha_k > 0$ by definition, this proves the first part of the result.

To obtain (2.1.4), one simply needs to use $\alpha_k = \frac{1}{L}$ in the series of equations above. □

The result of Proposition 2.1.1 will be instrumental to obtain complexity guarantees on Algorithm 1 in three different settings: nonconvex, convex, and strongly convex.

Nonconvex case In the nonconvex case, we aim at bounding the number of iterations required to drive the gradient norm below some threshold $\epsilon > 0$: this means that we should be able to show that the gradient norm actually goes below this threshold, which is a guarantee of convergence.

Theorem 2.1.1 (Complexity of gradient descent for nonconvex functions) *Let f be a nonconvex function satisfying Assumption 2.0.1. Suppose that Algorithm 1 is applied with $\alpha_k = \frac{1}{L}$. Then, for any $K \geq 1$, we have*

$$\min_{0 \leq k \leq K-1} \|\nabla f(\boldsymbol{w}_k)\| \leq \mathcal{O}\left(\frac{1}{\sqrt{K}}\right). \tag{2.1.5}$$

Proof. Let K be an iteration index such that for every $k = 0, \ldots, K-1$, we have $\|\nabla f(\boldsymbol{w}_k)\| > \epsilon$. From Proposition 2.1.1, we have that

$$\forall k = 0, \ldots, K-1, \quad f(\boldsymbol{w}_{k+1}) \leq f(\boldsymbol{w}_k) - \frac{1}{2L}\|\nabla f(\boldsymbol{w}_k)\|^2 \leq f(\boldsymbol{w}_k) - \frac{1}{2L}\left(\min_{0 \leq k \leq K-1} \|\nabla f(\boldsymbol{w}_k)\|\right)^2.$$

By summing across all such iterations, we obtain :

$$\sum_{k=0}^{K-1} f(\boldsymbol{w}_{k+1}) \leq \sum_{k=0}^{K-1} f(\boldsymbol{w}_k) - \frac{K}{2L}\left(\min_{0\leq k\leq K-1} \|\nabla f(\boldsymbol{w}_k)\|\right)^2.$$

Removing identical terms on both sides yields

$$f(\boldsymbol{w}_K) \leq f(\boldsymbol{w}_0) - \frac{K}{2L}\left(\min_{0\leq k\leq K-1} \|\nabla f(\boldsymbol{w}_k)\|\right)^2.$$

Using $f(\boldsymbol{w}_K) \geq f_{low}$ (which holds by Assumption 2.0.1) and re-arranging the terms leads to

$$\min_{0\leq k\leq K-1} \|\nabla f(\boldsymbol{w}_k)\| \leq \left[\frac{2L(f(\boldsymbol{w}_0) - f_{low})}{K}\right]^{1/2} = \mathcal{O}\left(\frac{1}{\sqrt{K}}\right).$$

□

Equivalently, we say that the worst-case complexity of gradient descent is $\mathcal{O}\left(\epsilon^{-2}\right)$, because for any $\epsilon > 0$, a reasoning similar to the proof of Theorem 2.1.1 guarantees that $\min_{0\leq k\leq K-1} \|\nabla f(\boldsymbol{w}_k)\| \leq \epsilon$ after at most

$$\left\lceil 2L(f(\boldsymbol{w}_0) - f_{low})\epsilon^{-2} \right\rceil = \mathcal{O}(\epsilon^{-2})$$

iterations.

Convex/Strongly convex case In addition to Assumption 2.0.1, if we further assume that the objective is convex or strongly convex, we can show that stronger guarantees than that of the nonconvex case can be obtained at a lower cost. This improvement illustrates the interest of convex functions in optimization.

In this paragraph, we let $f^* = \min_{\boldsymbol{w}\in\mathbb{R}^d} f(\boldsymbol{w})$ denote the minimal value of f (note that $f^* \geq f_{low}$) and we assume that there exists $\boldsymbol{w}^* \in \mathbb{R}^d$ such that $f(\boldsymbol{w}^*) = f^*$ (i.e. the set of minima is not empty). Given an accuracy threshold $\epsilon > 0$, we are interested in bounding the number of iterations necessary to reach an iterate $\boldsymbol{w}_k$ such that $f(\boldsymbol{w}_k) - f^* \leq \epsilon$.

Theorem 2.1.2 *Convergence of gradient descent for convex functions Let f be a convex function satisfying Assumption 2.0.1. Suppose that Algorithm 1 is applied with $\alpha_k = \frac{1}{L}$. Then, for any $K \geq 1$, the iterate $\boldsymbol{w}_K$ satisfies*

$$f(\boldsymbol{w}_k) - f^* \leq \mathcal{O}\left(\frac{1}{K}\right). \tag{2.1.6}$$

Proof. Let K be an index such that for every $k = 0, \ldots, K-1$, $f(\boldsymbol{w}_k) - f^* > \epsilon$.
For any $k = 0, \ldots, K-1$, the characterization of convexity (1.2.11) at $\boldsymbol{w}_k$ and $\boldsymbol{w}^*$ gives

$$f(\boldsymbol{w}^*) \geq f(\boldsymbol{w}_k) + \nabla f(\boldsymbol{w}_k)^{\mathrm{T}}(\boldsymbol{w}^* - \boldsymbol{w}_k).$$

Combining this property with (2.1.4), we obtain:

$$\begin{aligned} f(\boldsymbol{w}_{k+1}) &\leq f(\boldsymbol{w}_k) - \frac{1}{2L}\|\nabla f(\boldsymbol{w}_k)\|^2 \\ &\leq f(\boldsymbol{w}^*) + \nabla f(\boldsymbol{w}_k)^{\mathrm{T}}(\boldsymbol{w}_k - \boldsymbol{w}^*) - \frac{1}{2L}\|\nabla f(\boldsymbol{w}_k)\|^2. \end{aligned}$$

To proceed onto the next step, one notices that

$$\nabla f(\boldsymbol{w}_k)^{\mathrm{T}}(\boldsymbol{w}_k - \boldsymbol{w}^*) - \frac{1}{2L}\|\nabla f(\boldsymbol{w}_k)\|^2 = \frac{L}{2}\left(\|\boldsymbol{w}_k - \boldsymbol{w}^*\|^2 - \|\boldsymbol{w}_k - \boldsymbol{w}^* - \frac{1}{L}\nabla f(\boldsymbol{w}_k)\|^2\right).$$

Thus, recalling that $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \frac{1}{L}\nabla f(\boldsymbol{w}_k)$, we arrive at

$$\begin{aligned} f(\boldsymbol{w}_{k+1}) &\leq f(\boldsymbol{w}^*) + \frac{L}{2}\left(\|\boldsymbol{w}_k - \boldsymbol{w}^*\|^2 - \|\boldsymbol{w}_k - \boldsymbol{w}^* - \frac{1}{L}\nabla f(\boldsymbol{w}_k)\|^2\right) \\ &= f(\boldsymbol{w}^*) + \frac{L}{2}\left(\|\boldsymbol{w}_k - \boldsymbol{w}^*\|^2 - \|\boldsymbol{w}_{k+1} - \boldsymbol{w}^*\|^2\right). \end{aligned}$$

Hence,

$$f(\boldsymbol{w}_{k+1}) - f(\boldsymbol{w}^*) \leq \frac{L}{2}\left(\|\boldsymbol{w}_k - \boldsymbol{w}^*\|^2 - \|\boldsymbol{w}_{k+1} - \boldsymbol{w}^*\|^2\right). \tag{2.1.7}$$

By summing (2.1.7) on all indices k between 0 and $K-1$, we obtain

$$\sum_{k=0}^{K-1} f(\boldsymbol{w}_{k+1}) - f(\boldsymbol{w}^*) \leq \frac{L}{2}\left(\|\boldsymbol{w}_0 - \boldsymbol{w}^*\|^2 - \|\boldsymbol{w}_K - \boldsymbol{w}^*\|^2\right) \leq \frac{L}{2}\|\boldsymbol{w}_0 - \boldsymbol{w}^*\|^2.$$

Finally, using $f(\boldsymbol{w}_0) \geq f(\boldsymbol{w}_1) \geq ... \geq f(\boldsymbol{w}_K)$ (a consequence of Proposition 2.1.1, we obtain that

$$\sum_{k=0}^{K-1} f(\boldsymbol{w}_{k+1}) - f(\boldsymbol{w}^*) \geq K\left(f(\boldsymbol{w}_K) - f^*\right).$$

Injecting this formula into the previous equation finally yields the desired outcome:

$$f(\boldsymbol{w}_k) - f(\boldsymbol{w}^*) \leq \frac{L\|\boldsymbol{w}_0 - \boldsymbol{w}^*\|^2}{2}\frac{1}{K}.$$

□

Equivalently, we say that the worst-case complexity of gradient descent is $\mathcal{O}\left(\epsilon^{-1}\right)$, which means here that there exist a positive constant C (that depends on $\|\boldsymbol{w}_0 - \boldsymbol{w}^*\|$ and L) such that

$$f(\boldsymbol{w}_K) - f_{low} \leq \epsilon.$$

after at most $C\epsilon^{-1} = \mathcal{O}(\epsilon^{-1})$ iterations.

We now turn to the strongly convex case.

Theorem 2.1.3 *Convergence of gradient descent for strongly convex functions Let f be a μ-strongly convex function satisfying Assumption 2.0.1, with $\mu \in (0, L]$. Suppose that Algorithm 1 is applied with $\alpha_k = \frac{1}{L}$. Then, for any $K \in \mathbb{N}$, we have*

$$f(\boldsymbol{w}_k) - f^* \leq \mathcal{O}\left((1 - \tfrac{\mu}{L})^k\right). \tag{2.1.8}$$

We say that the convergence rate of gradient descent is $\mathcal{O}\left((1 - \frac{\mu}{L})^k\right)$.

Proof. We exploit the strong convexity property (1.2.13). For any $(\boldsymbol{x}, \boldsymbol{y}) \in (\mathbb{R}^n)^2$, we have

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}) + \frac{\mu}{2}\|\boldsymbol{y} - \boldsymbol{x}\|^2.$$

Minimizing both sides with respect to $\boldsymbol{y}$ lead to $\boldsymbol{y} = \boldsymbol{w}^*$ on the left-hand side, and $\boldsymbol{y} = \boldsymbol{x} - \frac{1}{\mu}\nabla f(\boldsymbol{x})$ on the right-hand side[2]. As a result, we obtain

$$\begin{aligned} f^* &\geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}\left[-\frac{1}{\mu}\nabla f(\boldsymbol{x})\right] + \frac{\mu}{2}\| - \frac{1}{\mu}\nabla f(\boldsymbol{x})\|^2 \\ f^* &\geq f(\boldsymbol{x}) - \frac{1}{2\mu}\|\nabla f(\boldsymbol{x})\|^2. \end{aligned}$$

By re-arranging the terms, we arrive at

$$\|\nabla f(\boldsymbol{x})\|^2 \geq 2\mu\left[f(\boldsymbol{x}) - f^*\right], \tag{2.1.9}$$

which is valid for any $\boldsymbol{x} \in \mathbb{R}^n$. Using (2.1.9) together with (2.1.4) thus gives

$$f(\boldsymbol{w}_{k+1}) \leq f(\boldsymbol{w}_k) - \frac{1}{2L}\|\nabla f(\boldsymbol{w}_k)\|^2 \leq f(\boldsymbol{w}_k) - \frac{\mu}{L}(f(\boldsymbol{w}_k) - f^*).$$

This leads to

$$f(\boldsymbol{w}_{k+1}) - f^* \leq \left(1 - \frac{\mu}{L}\right)(f(\boldsymbol{w}_k) - f^*),$$

which we can iterate in order to obtain

$$f(\boldsymbol{w}_K) - f^* \leq \left(1 - \frac{\mu}{L}\right)^K (f(\boldsymbol{w}_0) - f^*).$$

It then suffices to note that the bound is also valid for $K = 0$. □

Equivalently, we can show a worst-case complexity result: the method computes $\boldsymbol{w}_k$ such that $f(\boldsymbol{w}_k) - f^* \leq \epsilon$ in at most $\mathcal{O}(\frac{L}{\mu}\ln(\frac{1}{\epsilon}))$ iterations.

Similar results can be shown for the criterion $\|\boldsymbol{w}_k - \boldsymbol{w}^*\|$: in other words, the distance between the current iterate and the (unique) global optimum decreases at a rate $\mathcal{O}\left((1 - \frac{\mu}{L})^k\right)$.

Remark 2.1.1 *Proofs of convergence rates are typically more technical for convex and strongly convex problems: in order to obtain better bounds than in the nonconvex setting, one must make careful use of the (strong) convexity inequalities. In this course, we do not focus on these aspects, but rather draw insights from the final complexity bounds or convergence rates.*

2.2 Acceleration

2.2.1 Introduction: the momentum principle

In Section 2.1.3, we derive complexity bounds for the gradient descent algorithm, and we saw in particular that assuming that the function was convex (respectively, strongly convex) improved the complexity. These results are called *upper* complexity bounds, in the sense that they reflect the worst

[2] This is because the gradient of the quadratic function on the right-hand side with respect to $\boldsymbol{y}$ is $\nabla f(\boldsymbol{x}) + \mu(\boldsymbol{y} - \boldsymbol{x})$. This vector is equal to $\boldsymbol{0}$ if and only if $\boldsymbol{y} = \boldsymbol{x} - \frac{1}{\mu}\nabla f(\boldsymbol{x})$

possible convergence rate that this algorithm could exhibit on a given problem. The issue of *lower* bounds, that show a rate that cannot be improved upon, has been the subject to a lot of attention, particularly in the convex optimization community.

For nonconvex optimization, it is known that there exists a function for which gradient descent converges exactly at the $\mathcal{O}(\frac{1}{\sqrt{K}})$ rate: in this case, the lower bound matches the upper bound. On the contrary, for convex functions, the lower bound is actually $\mathcal{O}(\frac{1}{K^2})$, which is a sensible improvement over the bound in $\mathcal{O}(\frac{1}{K})$ of Theorem 2.1.2. There are methods that can achieve this bound, thanks to an algorithmic technique called **acceleration**.

The underlying idea of acceleration is that, at a given iteration and given the available information from previous iterations (in particular, the latest displacement), one can move along a better step than that given by the current gradient.

2.2.2 Nesterov's accelerated gradient method

Among the existing methods based on acceleration, the accelerated gradient algorithm proposed by Yurii Nesterov in 1983 is the most famous, to the point that it has been termed "Nesterov's algorithm".

Algorithm 3: Accelerated gradient method.

Initialization: $\boldsymbol{w}_0 \in \mathbb{R}^d$, $\boldsymbol{w}_{-1} = \boldsymbol{w}_0$.

for $k = 0, 1, \ldots$ **do**

1. Compute a steplength $\alpha_k > 0$ and a parameter $\beta_k > 0$.
2. Compute the new iterate as

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \nabla f\left(\boldsymbol{w}_k + \beta_k(\boldsymbol{w}_k - \boldsymbol{w}_{k-1})\right) + \beta_k(\boldsymbol{w}_k - \boldsymbol{w}_{k-1}). \tag{2.2.1}$$

end

Algorithm 3 provides a description of the method. Like the gradient descent method of Section 2.1, it requires a single gradient calculation per iteration; however, unlike in gradient descent, the gradient is not evaluated at the current iterate $\boldsymbol{w}_k$, but at a combination of this iterate with the previous step $\boldsymbol{w}_k - \boldsymbol{w}_{k-1}$: this term is called the **momentum term**, and is key to the performance of accelerated gradient techniques.

Another view of the accelerated gradient descent is that of a two-loop recursion: given $\boldsymbol{w}_0$ and $\boldsymbol{z}_0 = \boldsymbol{w}_0$, the update (2.2.1) can be rewritten as

$$\begin{cases} \boldsymbol{w}_{k+1} &= \boldsymbol{z}_k - \alpha_k \nabla f(\boldsymbol{z}_k) \\ \boldsymbol{z}_{k+1} &= \boldsymbol{w}_{k+1} + \beta_{k+1}(\boldsymbol{w}_{k+1} - \boldsymbol{w}_k). \end{cases} \tag{2.2.2}$$

This formulation decouples the two steps behind the accelerated gradient update: a gradient step on $\boldsymbol{z}_k$, combined with a momentum step on $\boldsymbol{w}_{k+1}$.

Choosing the parameters We now comment on the choice of the stepsize α_k and the momentum parameter β_k. The same techniques than those presented in Section 2.1.2 can be considered for the choice of α_k (stepsize parameter). As in the gradient descent case, the choice $\alpha_k = \frac{1}{L}$ is a standard one.

The choice of β_k is most crucial to obtaining the improved complexity bound. The standard values proposed by Nesterov depend on the nature of the objective function:

- If f is a μ-strongly convex, we set
$$\beta_k = \beta = \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}} \tag{2.2.3}$$
for every k. Note that this requires the knowledge of both the Lipschitz constant of the gradient and the strong convexity constant.
- For a general convex function f, β_k is computed in an adaptive way using two sequences, as follows:
$$t_{k+1} = \frac{1}{2}(1+\sqrt{1+4t_k^2}), t_0 = 0, \quad \beta_k = \frac{t_k - 1}{t_{k+1}}. \tag{2.2.4}$$

The following informal theorem summarizes the complexity results that can be proven for Algorithm 3.

Theorem 2.2.1 *Consider Algorithm 3 applied to a convex function f satisfying Assumption 2.0.1, with $\alpha_k = \frac{1}{L}$, and let $\epsilon > 0$. Then, for any $K \geq 1$, the iterate $\boldsymbol{w}_K$ computed by Algorithm 3 satisfies*

i) $f(\boldsymbol{w}_K) - f^* \leq \mathcal{O}(\frac{1}{K^2})$ for a generic convex function if β_k is set according to the adaptive rule (2.2.4);

ii) At most $f(\boldsymbol{w}_K) - f^* \leq \left((1-\sqrt{\frac{\mu}{L}})^K\right)$ for a μ-strongly convex function, provided β_k is set to the constant value given by (2.2.3).

Note that we can also derive worst-case complexity bounds for the accelerated gradient method, that show the same improvement. For instance, for strongly convex functions, we can establish that $f(\boldsymbol{w}_k) - f^* \leq \epsilon$ after at most $\mathcal{O}\left(\sqrt{\frac{L}{\mu}}\ln(\epsilon^{-1})\right)$ iterations, which represents an improvement over the $\mathcal{O}\left(\frac{L}{\mu}\ln(\epsilon^{-1})\right)$ complexity over gradient descent. Here the improvement is in terms of problem-dependent constants.

2.2.3 Other accelerated methods

Heavy ball method The heavy ball method is a precursor of the accelerated gradient algorithm, that was proposed by Boris T. Polyak in 1964. Its k-th iteration can be written as

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha\nabla f(\boldsymbol{w}_k) + \beta(\boldsymbol{w}_k - \boldsymbol{w}_{k-1}),$$

where the stepsize and momentum parameters are chosen to be constant values. The key difference between this iteration and Nesterov's lies in the gradient evaluation, which the heavy ball method performs at the current point: in that sense, the heavy ball method performs first the gradient update, then the momentum step, while Nesterov's method adopts the inverse approach. This method achieves the optimal rate of convergence on strongly convex quadratic functions, but can fail on general strongly convex functions.

Conjugate gradient The (linear) conjugate gradient method, proposed by Hestenes and Stiefel in 1952, has remained to this day one of the preferred methods to solve linear systems of equations and strongly convex quadratic minimization problems. Unlike Polyak's method, the conjugate gradient algorithm does not require knowledge of the Lipschitz constant L nor the parameter μ, because it exploits knowledge from the past iterations. The k-th iteration of conjugate gradient can be written as:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \alpha_k \boldsymbol{p}_k, \quad \boldsymbol{p}_k = -\nabla f(\boldsymbol{w}_k) + \beta_k \boldsymbol{p}_{k-1}.$$

In a standard conjugate gradient algorithm, α_k and β_k are computed using formulas tailored to the problem: this contributes to their convergence rate analysis, which leads to a rate similar to that of accelerated gradient. However, unlike accelerated gradient, the conjugate gradient is guaranteed to terminate after d iterations on a d-dimensional problem. When d is very large, the bound for conjugate gradient matches that of the other methods, and in that sense does not depend on the problem dimension.

Example 2.2.1 (Strongly convex quadratic minimization) *A strongly convex quadratic minimization problem is an optimization problem of the form*

$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{minimize}} \; q(\boldsymbol{w}) := \tfrac{1}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{w} - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{w}$$

where $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ is a symmetric positive definite matrix and $\boldsymbol{b} \in \mathbb{R}^d$. This problem is smooth (because the objective is polynomial in all of the decision variables) and $\nabla^2 f(\boldsymbol{w}) \succ \boldsymbol{0}$ for every $\boldsymbol{w}$, meaning that the problem is μ-strongly convex with μ denoting the minimum eigenvalue of $\boldsymbol{A}$. As a result, there exist a unique global minimum given by the solution of $\nabla q(\boldsymbol{w}) = \boldsymbol{A}\boldsymbol{w} - \boldsymbol{b} = 0$. This equation is a linear system but the cost of inverting this system and computing a solution can be prohibitive. For this reason, one can replace the exact solve by an iterative, gradient-based approach, and apply Algorithm 1 or Algorithm 3. Note that $q \in \mathcal{C}^{1,1}_{\|\boldsymbol{A}\|}(\mathbb{R}^d)$, hence the choice of steplength 2.1.3 is a valid one.

If gradient descent is applied, then an ϵ-accuracy in the objective value can be reached in at most $\mathcal{O}\left(\frac{L}{\mu}\ln(\frac{1}{\epsilon})\right)$ iterations, while if one applies the accelerated gradient or the heavy ball method with appropriately chosen parameters, this bound improves to $\mathcal{O}\left(\frac{L}{\mu}\ln(\frac{1}{\epsilon}\right)$. Finally, if we aim at using conjugate gradient, the result bound will be in $\mathcal{O}\left(\min\{d, \frac{L}{\mu}\ln(\frac{1}{\epsilon})\}\right)$.

Chapter 3

Stochastic gradient techniques

3.1 Introduction

In this chapter, we will develop method that fully exploit the structure of data science problems. To this end, we assume that we are given a dataset of n examples under the form $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$, where $\boldsymbol{x}_i \in \mathbb{R}^{d_x}$ and $\boldsymbol{y}_i \in \mathbb{R}^{d_y}$ are obtained from a certain data distribution. As in the linear regression example from the previous chapter, we seek a model function $h : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$ such that $h(\boldsymbol{x}_i) \approx \boldsymbol{y}_i$ for any $i = 1, \dots, n$. We will parameterize this model by a vector $\boldsymbol{w} \in \mathbb{R}^d$ ($h(\boldsymbol{x}_i) = h(\boldsymbol{x}_i; \boldsymbol{w})$) so that knowing the vector $\boldsymbol{w}$ suffices to be able to evaluate the model.

In order to quantify the model's ability to represent the data, we define a cost function (or loss function) of the form $\ell : (h, y) \mapsto \ell(h, y)$: this function aims at penalizing values (h, y) such that $h \neq y$. The mean-square or ℓ_2 loss defined by $(h, y) \mapsto \|h - y\|^2$ is a canonical example of a loss. We then define the loss at a given example by $\ell(h(\boldsymbol{w}; \boldsymbol{x}_i), y_i)$.

We wish to compute the best model according to our entire dataset, leading to average the losses over all examples. This finally leads to the following optimization problem.

Definition 3.1.1 (Finite sum problem) *Consider a dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ with $\boldsymbol{x}_i \in \mathbb{R}^{d_x}$ and $\boldsymbol{y}_i \in \mathbb{R}^{d_y}$, a model class $\{h(\boldsymbol{w}; \cdot) : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}\}_{\boldsymbol{w} \in \mathbb{R}^d}$ and a cost function ℓ. The* **finite-sum problem** *associated with this setup is defined by:*

$$\underset{\boldsymbol{w} \in \mathbb{R}^d}{\text{minimize}}\, f(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^n \ell(h(\boldsymbol{w}; \boldsymbol{x}_i), \boldsymbol{y}_i) = \frac{1}{n} \sum_{i=1}^n f_i(\boldsymbol{w}). \tag{3.1.1}$$

Provided the f_is are all $\mathcal{C}^1$ functions, the function f is also $\mathcal{C}^1$ and we can apply gradient descent to problem (3.1.1). The k-th iteration of this method is:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \nabla f(\boldsymbol{w}_k) = \boldsymbol{w}_k - \frac{\alpha_k}{n} \sum_{i=1}^n \nabla f_i(\boldsymbol{w}_k).$$

From the iteration, it is clear that every gradient descent iteration requires to access**the entire dataset** for a single gradient calculation. In a big data setting, the number of examples n can be extremely large, and gradient descent can be too expensive to be used in practice.

Remark 3.1.1 *In a purely stochastic or "online", examples may only be available in a streaming fashion: in this context, it may be impossible to compute a finite average over all samples, and we*

consider instead the stochastic optimization problem:

$$\underset{\boldsymbol{w}\in\mathbb{R}^d}{\text{minimize}}\, \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})}\left[f_{(\boldsymbol{x},\boldsymbol{y})}(\boldsymbol{w})\right]. \tag{3.1.2}$$

Even though the gradient of the objective may exist, it can be quite difficult to compute This formulation is often closer to the real goal of machine learning (i.e. have a good model for all possible examples in the distribution), yet it is often replaced by (3.1.1) *to account for the empirical setting.*

3.2 Stochastic gradient method

3.2.1 Algorithm

The idea behind stochastic gradient is remarkably simple. It considers the problem (3.1.1) under the assumption that every component function f_i possesses a gradient: every iteration then consists in choosing a random index i_k and taking a step in the direction of the negative gradient of the function f_{i_k}. Algorithm 4 details this process.

Algorithm 4: Stochastic gradient method

Initialization: $\boldsymbol{w}_0 \in \mathbb{R}^d$.
for $k = 0, 1, \ldots$ **do**

1. Define a stepsize $\alpha_k > 0$.
2. Draw an index $i_k \in \{1, \ldots, n\}$.
3. Compute the new iterate

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \nabla f_{i_k}(\boldsymbol{w}_k). \tag{3.2.1}$$

end

The vector $\nabla f_{i_k}(\boldsymbol{w}_k)$ is called a **stochastic gradient** for $\boldsymbol{w}_k$. The key property of iteration (3.2.1) is that it only requires one example from the dataset: as a result, **its cost in terms of accessing the data is n times lower compared to an iteration of gradient descent**. As a result, the comparison between the two methods is often conducted in terms of **epochs**.

Definition 3.2.1 (Epoch) *For problem* (3.1.1)*, an epoch represents n calculations of a sample gradient ∇f_i.*

As a result, the cost of one iteration of gradient descent is that of one epoch, an iteration of Algorithm 4 only costs a fraction ($\frac{1}{n}$) of an epoch.

To conclude this presentation of stochastic gradient, we make several key remarks on its behavior.

Remark 3.2.1 *In general, it is not possible to guarantee convergence of gradient descent. Consider the two-dimensional problem:*

$$\underset{w\in\mathbb{R}}{\text{minimize}}\, \frac{1}{2}(f_1(w) + f_2(w))$$

with $f_1(w) = 2w^2$ and $f_2 = -w^2$. If $w_0 > 0$ and $i_k = 2$ for any k, then the method will diverge.

It is easy to build problems and draws of indices for which stochastic gradient does not converge. In practice, however, finite-sum problems from data science involve component functions that are quite similar, corresponding to data samples that follow a similar distribution. As a result, improving the loss specific to a given sample may very well improve other losses with respect to similar samples. This observation partly explains the success of stochastic gradient methods in data-related settings.

Remark 3.2.2 *Algorithm 4 is often called* **Stochastic Gradient Descent**, *or SGD. This denomination is inaccurate, in that one cannot guarantee that the stochastic gradient method will behave like a descent method (i.e. that it will decrease the function value at every iteration). For this reason, we will call this method* **stochastic gradient** *in these notes, thereby following other authors' terminology [1]. However, we point out that many librairies refer to their implementation of Algorithm 4 as SGD.*

3.2.2 Convergence rate analysis

In this section, we describe the key steps to deriving convergence rates (and complexity bounds) for stochastic gradient, under a slightly altered version of Assumption 2.0.1.

Assumption 3.2.1 *The objective function $f = \frac{1}{n}\sum_{i=1}^n f_i$ of problem (3.1.1) is $\mathcal{C}_L^{1,1}(\mathbb{R}^d)$ for some $L > 0$, and bounded below by $f_{low} \in \mathbb{R}$. Moreover, every function f_i is of class $\mathcal{C}^1$.*

The key argument for analyzing gradient descent is the result of Proposition 2.1.1: in particular, the inequality

$$f(\boldsymbol{w}_{k+1}) \leq f(\boldsymbol{w}_k) - \left(\alpha_k - \frac{L}{2}\alpha_k^2\right)\|\nabla f(\boldsymbol{w}_k)\|^2$$

leads to decrease guarantees for a sufficiently small step size.

It is possible to obtain a similar inequality in the case of stochastic gradient (in a probabilistic sense). To this end, assumptions on the random indices (and thus the stochastic gradients used throughout the algorithm) are necessary.

Assumption 3.2.2 (Stochastic gradients' properties) *For every iteration k of Algorithm 4, the random index i_k is drawn so that*

i) i_k is independent of $i_0, \ldots, i_{k-1}$;

ii) $\mathbb{E}_{i_k}\left[\nabla f_{i_k}(\boldsymbol{w}_k)\right] = \nabla f(\boldsymbol{w}_k)$;

iii) $\mathbb{E}_{i_k}\left[\|\nabla f_{i_k}(\boldsymbol{w}_k)\|^2\right] \leq \|\nabla f(\boldsymbol{w}_k)\|^2 + \sigma^2$ with $\sigma^2 \in (0, \infty)$.

The second property of Assumption 3.2.2 implies that the stochastic gradient $\nabla f_{i_k}(\boldsymbol{w}_k)$ is an unbiased estimator of the true gradient $\nabla f(\boldsymbol{w}_k)$. The third property guarantees that the norm of this stochastic estimator cannot deviate too much from the true norm, which is critical in guaranteeing some form of convergence. The most classical strategy to satisfy these assumptions is given below.

Example 3.2.1 (Uniform sampling) *Suppose that $\{i_k\}_k$ is a sequence of independent, identically distributed indices uniformly drawn in $\{1, \ldots, n\}$. Then, Algorithm 4 satisfies Assumption 3.2.2.*

Using Assumption 3.2.2, one can establish a useful inequality for analyzing Algorithm 4.

Proposition 3.2.1 *Let Assumptions 2.0.1 and 3.2.2 hold, and consider the k-th iteration of Algorithm 4. Then,*

$$\mathbb{E}_{i_k}\left[f(\boldsymbol{w}_{k+1})\right] - f(\boldsymbol{w}_k) \leq \nabla f(\boldsymbol{w}_k)^{\mathrm{T}}\, \mathbb{E}_{i_k}\left[\boldsymbol{w}_{k+1} - \boldsymbol{w}_k\right] + \frac{L}{2}\, \mathbb{E}_{i_k}\left[\|\boldsymbol{w}_{k+1} - \boldsymbol{w}_k\|^2\right]. \tag{3.2.2}$$

As shown by (3.2.2), one can guarantee decrease over an iteration of stochastic gradient **in expectation** (provided the right-hand side of (3.2.2) is negative). This property is sufficient to obtain convergence rates (or complexity bounds) for stochastic gradient applied to strongly convex, convex and nonconvex problems. These results heavily depend on the choice for the stepsize sequence $\{\alpha_k\}_k$: in fact, tuning the stepsize is a major issue in machine learning (in which the stepsize is often referred to as a learning rate).

In the rest of this section, we mainly focus on strongly convex problems, and make the following assumption.

Assumption 3.2.3 *The objective function f of problem (3.1.1) is continuous and μ-strongly convex. It possesses a unique global minimum $\boldsymbol{w}^*$, and we let $f^* = f(\boldsymbol{w}^*)$.*

We first consider the case of a constant stepsize, for which we can establish the following result.

Theorem 3.2.1 (Stochastic gradient with constant stepsize) *Let Assumptions 2.0.1, 3.2.2 and 3.2.3 hold. Suppose that we apply Algorithm 4 with a constant stepsize*

$$\alpha_k = \alpha \in (0, \tfrac{1}{L}] \quad \forall k.$$

Then, for every $K \in \mathbb{N}$,

$$\mathbb{E}\left[f(\boldsymbol{w}_K) - f^*\right] \leq \frac{\alpha L \sigma^2}{2\mu} + (1 - \alpha\mu)^K \left[f(\boldsymbol{w}_0) - f^* - \frac{\alpha L \sigma^2}{2\mu}\right], \tag{3.2.3}$$

where the expected value is taken on all randomness up to iteration K.

The result of Theorem 3.2.1 shows a linear rate of convergence for stochastic gradient in expectation, which is comparable to that of gradient descent. However, this rate only guarantees that $\{f(\boldsymbol{w}_K)\}$ converges to a value in $\left[f^*, f^* + \frac{\alpha L \sigma^2}{2\mu}\right]$: even on average, stochastic gradient can only be guaranteed to converge towards a neighborhood of the optimal value. Indeed, the result (3.2.3) involves a constant term on its right-hand side, showing a possible gap to the optimum in $\frac{\alpha L \sigma^2}{4\mu}$. It is therefore not possible to guarantee that $\mathbb{E}\left[f(\boldsymbol{w}_k) - f^*\right] \leq \epsilon$ for any $\epsilon > 0$, but only for sufficiently large values of ϵ. However, the use of a constant stepsize guarantees a linear rate of convergence.

In the original version of stochastic gradient (proposed by Robbins and Monro in 1951), the sequence of stepsizes was required to satisfy

$$\sum_{k=0}^{\infty} \alpha_k = \infty \quad \text{and} \quad \sum_{k=0}^{\infty} \alpha_k^2 < \infty,$$

which lead to asymptotic convergence of the method. In order to satisfy these requirements, the sequence α_k must go to 0 as k goes to infinity. Therefore, we now describe a convergence result based on using a decreasing stepsize.

Theorem 3.2.2 (Stochastic gradient with decreasing stepsize) *Under Assumptions 2.0.1, 3.2.2 and 3.2.3, consider Algorithm 4 applied with a decreasing stepsize sequence of the form*

$$\alpha_k = \frac{\beta}{k+\gamma},$$

with $\beta > \frac{1}{\mu}$ and $\gamma > 0$ chosen so that $\alpha_0 = \frac{\beta}{\gamma} \leq \frac{1}{L}$. Then, for any $K \in \mathbb{N}$, we have

$$\mathbb{E}\left[f(\boldsymbol{w}_K) - f^*\right] \leq \frac{\nu}{\gamma + K}, \tag{3.2.4}$$

where

$$\nu := \max\left\{\frac{\beta^2 L\sigma^2}{2(\beta\mu - 1)}, \gamma(f(\boldsymbol{w}_0) - f^*)\right\}.$$

As for gradient descent, using a decreasing stepsize sequence poses the risk of producing tiny steps early in the algorithmic run. However, note that it guarantees convergence of $\{f(\boldsymbol{w}_k)\}$ to f^* in expectation, which is a strongly result than that of Theorem 3.2.1. Still, the convergence rate in $\mathcal{O}(1/K)$ is worse than the one we established for gradient descent, in $\mathcal{O}((1-\mu/L)^K)$. Note however that this comparison is related to the number of iterations: if we include the per-iteration cost (in terms of accesses to the data), the stochastic gradient method has a convergence rate of $\mathcal{O}(1/K)$ (1 access per iteration) while the gradient descent method yields a rate in $\mathcal{O}(n(1-\mu/L)^K)$. The latter can be significantly worse than the former for large n.

Remark 3.2.3 (A hybrid approach) *A common stepsize strategy used in deep learning consists in adopting a learning rate schedule. In this hybrid approach, one runs first stochastic gradient with a constant value α upon a certain point, then replaces α by $\alpha' < \alpha$: the process is repeated until a suitable point has been found, or the budget of epochs is exhausted.*

In the strongly convex setting, the result of Theorem 3.2.1 provides a useful criterion to reduce the stepsize. Indeed, starting with a stepsize α, one can guarantee that a neighborhood of size $\frac{\alpha L\sigma^2}{\mu}$ can be reached in a bounded number of iterations (or epochs). Therefore, once that point is reached, one can reduce α. This approach guarantees to converge to the true optimal value, at a sublinear rate: for any $\epsilon > 0$,

$$\mathbb{E}\left[f(\boldsymbol{w}_k) - f^*\right] \leq \epsilon \quad \textit{after} \quad \mathcal{O}(1/\epsilon) \textit{ iterations.}$$

In practice, knowing how the neighborhood is reached may not be possible: other criteria, such as detecting stalling in the algorithm (lack of progress in the iterates, in the function values, etc), can help in designing strategies: several learning rate schedules are implemented in modern libraries.

Stepsize choice and nonconvex optimization Stochastic gradient and its variants are the most common methods for training deep neural networks: the associated optimization problem is highly nonconvex, and the above analysis cannot be applied. Still, it is possible to derive convergence rates for the nonconvex case by studying the following quantities:

- $\mathbb{E}\left[\frac{1}{K}\sum_{i=1}^K \|\nabla f(\boldsymbol{w}_k)\|^2\right]$ for the variants of Algorithm 4 based on constant stepsizes.
- $\mathbb{E}\left[\frac{1}{\sum_{i=1}^K \alpha_k}\sum_{i=1}^K \alpha_k\|\nabla f(\boldsymbol{w}_k)\|^2\right]$ for the variants using a decreasing stepsize sequence.

As in the strongly convex case, the iteration results for stochastic gradient will be worse than that of gradient descent. For instance, applying 4 with a constant stepsize guarantees that

$$\mathbb{E}\left[\frac{1}{K}\sum_{i=1}^{K}\|\nabla f(\boldsymbol{w}_k)\|^2\right] \leq \epsilon$$

in at most $\mathcal{O}(\epsilon^{-4})$ iterations. This bound is worse than the $\mathcal{O}(\epsilon^{-2})$ bound for gradient descent. In addition, this result is only valid for sufficiently large ϵ because of the stochastic nature of the method.

Remark 3.2.4 (Advanced stochastic gradient techniques) *The most common stochastic gradient variants for deep learning (*SGD WITH MOMENTUM, ADAM, ADAGRAD, RMSPROP*) build on the stochastic gradient principle by adding a momentum term to the stochastic gradient and/or a scaling to every component of the step so as to reduce the need for intensive stepsize tuning. Analyzing these methods is significantly more complicated than in the deterministic setting, and theoretical results have yet to be reconciled with practical behavior.*

3.3 Variance reduction

The theory of Section 3.2.2 relies heavily on Assumption 3.2.2, and more precisely on a control over the norm of the stochastic gradient through the quantity σ^2. A higher value of σ leads to looser neighborhoods of the solution and, as a result, worse theoretical guarantees. Numerically, this translates into high variability of the method's output.

Variance reduction techniques have been designed to reduce the "variance" of gradient estimates used in stochastic gradient. The goal of this section is to summarize key approaches to reducing variance.

3.3.1 Batch methods

As explained in the previous section, the iteration of stochastic gradient is

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \nabla f_{i_k}(\boldsymbol{w}_k),$$

where i_k is a random index drawn within $\{1, \ldots, n\}$. This method relies on a **single** example to build a stochastic gradient estimate, and this estimate comes with a certain variance (illustrated by the parameter σ^2 in Assumption 3.2.2). On the contrary, gradient descent relies on an exact gradient computed using all the samples, which leads to a deterministic execution and no variance in the (exact) gradient estimate.

To improve the variance of this method, it appears natural to consider stochastic gradient estimates based on several samples: this is the underlying principle of **batch stochastic gradient methods**.

At every iteration of a batch method, a (random) set of indices S_k with indices between 1 and n is drawn, and the following update is performed:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \frac{1}{|S_k|}\sum_{i\in S_k}\nabla f_i(\boldsymbol{w}_k) \tag{3.3.1}$$

When $|S_k| = 1$, we recover the classical stochastic gradient formula. When $|S_k| = n$ and the indices are drawn without replacement, we obtain $S_k = \{1, \dots, n\}$, and iteration (3.3.1) corresponds to that of gradient descent.

More broadly, we can identify two regimes of batch sizes:

- $|S_k| \approx n$: the per-iteration cost of such a variant is close to that of gradient descent, hence these batch methods often exhibit a behavior comparable to that of gradient descent.
- $1 < |S_k| = n_b << n$: this regime, called mini-batching, is often thought as a good way to reduce variance while keeping the per-iteration cost to an affordable level.

Assuming that the batch size is constant over all iterations, i.e. that $|\mathcal{S}_k| = n_b \ \forall k$, it is possible to show that a mini-batch variant reaches a closer neighborhood than stochastic gradient. Every iteration of the mini-batch method is of course more expensive than an iteration of stochastic gradient (in fact, an epoch corresponds to n/n_b iterations of a batch stochastic gradient method with a fixed batch size of n_b). Note that the cost of batch methods can be mitigated by exploiting parallel computing resources. Indeed, in a distributed data setup, stochastic gradients can often be evaluated in parallel, which reduces the effective cost of batch techniques.

Proposition 3.3.1 *Under Assumptions 2.0.1 and 3.2.2, we have*

$$\mathbb{E}_{S_k}\left[\frac{1}{|S_k|}\sum_{i\in S_k}\nabla f_i(\boldsymbol{w}_k)\right] = \nabla f(\boldsymbol{w}_k)$$

and

$$\mathbb{E}_{S_k}\left[\left\|\frac{1}{|S_k|}\sum_{i\in S_k}\nabla f_i(\boldsymbol{w}_k)\right\|_2^2\right] \leq \frac{\sigma^2}{n_b}.$$

Finally, we mention that (mini)-batch approaches remain more expensive than "vanilla" stochastic gradient, while being more sensitive to redundancies in the data, and introducing an new hyperparameter to be tuned (the batch size). For this reason, the classical stochastic gradient approach can still be observed to be more efficient on some example, and it remains a popular algorithmic choice in practice.

3.3.2 Other variance reduction techniques

Gradient aggregation techniques are variance reduction approaches with well-established convergence rates, that are provably better than that of stochastic gradient. They have attracted significant interest from the academic machine learning and optimization community, though their use in modern data science settings like deep learning has not been successful (these techniques are still efficient on simple models, and are part of librairies like `scikit-learn`). They essentially consist in maintaing a **full gradient estimate** throughout the iterations, which requires at least one full gradient calculation: the gradient estimate used to make a step then combines the full gradient estimator with a new stochastic gradient, leading to a corrected step, and a method with provably smaller variance.

Iterate averaging is another way to reduce the variance in stochastic gradient at no additional cost in terms of accesses to data points. The idea consists in analyzing the properties of a running

average $\frac{1}{K}\sum_{k=0}^{K-1} \boldsymbol{w}_k$. In certain cases (in particular, when $\alpha = \frac{1}{\mu(k+1)}$ and f is μ-strongly convex), this sequence possesses favorable properties and its behavior is less variable. However, computing this average either requires to store all iterates (which becomes expensive in terms of storage) or necessitates to maintain a running average (subject to numerical errors).

3.4 Stochastic gradient methods for deep learning

In this section, we review the main stochastic gradient techniques that are used to train deep learning models. Our focus remains on finite-sum problems of the form (3.1.1) under Assumption 3.2.1. Our goal is to study several variants on the iterative scheme

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha \boldsymbol{g}_k, \tag{3.4.1}$$

where $\alpha > 0$ is a fixed stepsize (or learning rate), and $\boldsymbol{g}_k$ is a stochastic gradient estimator, that may correspond to taking a single term from the finite sum (as in stochastic gradient) or to a batch of indices.

To encompass all the variants of interest, we consider a more general iteration of the form

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha \boldsymbol{m}_k \oslash \boldsymbol{v}_k, \tag{3.4.2}$$

where $\alpha > 0$, $\boldsymbol{m}_k, \boldsymbol{v}_k \in \mathbb{R}^d$ and $\oslash$ denotes the componentwise division operator:

$$\boldsymbol{m}_k \oslash \boldsymbol{v}_k := \left[\frac{[\boldsymbol{m}_k]_i}{[\boldsymbol{v}_k]_i}\right]_{i=1,\dots,d}.$$

To see that (3.4.2) generalizes (3.4.1), set $\boldsymbol{m}_k = \boldsymbol{g}_k$ and $\boldsymbol{v}_k = \mathbf{1}_{\mathbb{R}^d}$. The iteration (3.4.2) is particularly convenient to express the popular methods in deep learning using a single format.

3.4.1 Stochastic gradient with momentum

Most practical implementations of stochastic gradient combine the basic step (3.4.1) together with a momentum term, similarly to the accelerated methods described in Chapter 2. An iteration of gradient descent with momentum reads

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha(1-\beta)\boldsymbol{g}_k + \alpha\beta\left(\boldsymbol{w}_k - \boldsymbol{w}_{k-1}\right), \tag{3.4.3}$$

where $\beta \in (0,1)$ is a momentum parameter (when $\beta = 0$ we again obtain the standard stochastic gradient iteration). Iteration (3.4.3) can be seen as a version of Polyak's method where the gradient-type step is combined with the previous direction. As for the heavy-ball method, the idea is to incorporate information from the previous step through momentum. In practice, the iteration (3.4.3) often leads to accumulation of good steps (in terms of optimization), whereas bad directions and bad steps tend to cancel out.

The basic method (3.4.3) can be recovered from (3.4.2), by setting $\boldsymbol{v}_k = \mathbf{1}_{\mathbb{R}^d}$ and defining $\boldsymbol{m}_k$ in a recursive manner through $\boldsymbol{m}_{-1} = \mathbf{0}_{\mathbb{R}^d}$ and

$$\boldsymbol{m}_k = (1-\beta)\boldsymbol{g}_k - \beta \boldsymbol{m}_{k-1} \quad \forall k \in \mathbb{N}.$$

where $\beta \in (0,1)$.

Stochastic gradient with momentum is implemented in most deep learning librairies such as PyTorch. It has shown great success in training deep neural networks on computer vision problems, and is partly responsible for the rise of deep learning in the early 2010s.

Remark 3.4.1 *Theoretical guarantees for the iteration (3.4.3) are much more difficult to obtain than for accelerated gradient (in particular, it is not well understood whether such a method can be provably faster than SGD). Nevertheless, stochastic gradient techniques with momentum are widely used in practice, even on nonconvex problems such as neural network training.*

3.4.2 AdaGrad

The adaptive gradient method, or AdaGrad, was proposed in 2011 to address the issue of setting the learning rate α in stochastic gradient. Rather than using costly procedures such as line search, AdaGrad scales every coordinate of the stochastic gradient using information from the values of that coordinate in the previous iterations. Mathematically, the method maintains a sequence $\{\boldsymbol{r}_k\}_k$ defined by

$$\forall i = 1, \ldots, d, \quad \left\{ \begin{array}{l} [\boldsymbol{r}_{-1}]_i = 0 \\ [\boldsymbol{r}_k]_i = [\boldsymbol{r}_{k-1}]_i + [\boldsymbol{g}_k]_i^2 \quad \forall k \geq 0, \end{array} \right. \tag{3.4.4}$$

The AdaGrad iteration can then be written as

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha \boldsymbol{g}_k \oslash \sqrt{\boldsymbol{r}_k}, \tag{3.4.5}$$

where the square root is applied componentwise to $\boldsymbol{r}_k$. This iteration is a special cas of (3.4.2), where $\boldsymbol{m}_k = \boldsymbol{g}_k$ and $\boldsymbol{v}_k = \sqrt{\boldsymbol{r}_k}$. The novelty in AdaGrad does not lie in the use of momentum, but in the use of one stepsize per coordinate. The stepsize sequence thus has the form

$$\left\{ \left[\frac{\alpha}{\sqrt{[\boldsymbol{r}_k]_i}} \right]_{i=1}^{d} \right\}_k .$$

The resulting *diagonal scaling* on the coordinates of $\boldsymbol{g}_k$ leads to stepsizes that adapt to coordinates that can vary by orders of magnitude (which would require a careful choice of α in basic stochastic gradient). On the other hand, the accumulation process at work in the definition of $\boldsymbol{r}_k$ results in stepsizes that are monotonically decreasing, and that often converge quickly towards 0.

Remark 3.4.2 *In practice, $\boldsymbol{r}_k$ is replaced by $\boldsymbol{r}_k + \eta \mathbf{1}_{\mathbb{R}^d}$ where $\eta > 0$ is a small value that helps with numerical stability.*

The AdaGrad method is particularly interesting for problems with sparse gradients, in which stochastic gradients tend to have many zero coordinates. In this situation, using $\boldsymbol{r}_k$ will only modify the stepsizes corresponding to nonzero gradient coordinates. Many problems in recommendation systems have a sparse structure, and AdaGrad is considered to be an efficient method for this class of problems.

3.4.3 RMSProp

The RMSProp (*Root Mean Square Propagation*) algorithm is similar in spirit to AdaGrad, in that it scales the gradient components at every step. This method relies on a sequence $\{\boldsymbol{r}_k\}_k$ defined by

$$\forall i = 1, \ldots, d, \quad \left\{ \begin{array}{l} [\boldsymbol{r}_{-1}]_i = 0 \\ [\boldsymbol{r}_k]_i = (1-\lambda)[\boldsymbol{r}_{k-1}]_i + \lambda [\boldsymbol{g}_k]_i^2 \quad \forall k \geq 0, \end{array} \right. \tag{3.4.6}$$

where $\lambda \in (0,1)$. The value of λ is used to put more weight on the current gradient coordinates than on the coordinates from past iterations (this information being contained in $\boldsymbol{r}_{k-1}$). This simple idea slows down the decrease of the stepsizes to 0, compared to the stepsizes of AdaGrad.

With the definition (3.4.6), the RMSProp iteration corresponds has the same form as that of AdaGrad, that is, a special case of (3.4.2) with $\boldsymbol{m}_k = \boldsymbol{g}_k$ and $\boldsymbol{v}_k = \sqrt{\boldsymbol{r}_k}$.

Remark 3.4.3 *As for* AdaGrad, *standard practice replaces* $\boldsymbol{r}_k$ *by* $\boldsymbol{r}_k + \eta \mathbf{1}_{\mathbb{R}^d}$, *where* $\eta > 0$ *is a small quantity.*

The RMSProp method has been found quite successful for training very deep neural networks.

3.4.4 Adam

The Adam optimization method was proposed in 2013. This method can be thought as combining the idea of momentum (used in the stochastic gradient method of Section 3.4.1) with the diagonal scaling procedure on which both AdaGrad and RMSProp are based. An iteration of Adam falls into the generic scheme (3.4.2) by setting

$$\boldsymbol{m}_k = \frac{(1-\beta_1)\sum_{j=0}^{k} \beta_1^{k-j} \boldsymbol{g}_j}{1-\beta_1^{k+1}} \tag{3.4.7}$$

and

$$\boldsymbol{v}_k = \sqrt{\frac{(1-\beta_2)\sum_{j=0}^{k} \beta_2^{k-j} \boldsymbol{g}_j \odot \boldsymbol{g}_j}{1-\beta_2^{k+1}}}. \tag{3.4.8}$$

Here $\beta_1, \beta_2 \in (0,1)$, and $\odot$ denotes the Hadamard or componentwise product

$$\boldsymbol{g}_k \odot \boldsymbol{g}_k = \left[[\boldsymbol{g}_k]_i^2\right]_{i=1}^{d}.$$

Remark 3.4.4 *In practice,* $\boldsymbol{v}_k + \eta \mathbf{1}_{\mathbb{R}^d}$ *(with small* $\eta > 0$*) is used in lieu of* $\boldsymbol{v}_k$.

The above formulas describe the two components of Adam. On one hand, a weighted combination of the previous steps that puts the emphasis on the most recent steps (and the current stochastic gradient) defines the direction of the next step. On the other hand, a diagonal scaling is applied to the coordinates of this direction, again according to a weighted average of the coordinates from the previous iterations. This important feature, that has a statistical motivation, appears to be responsible for the success of Adam in practice. The impressive performance of Adam on training neural networks has contributed to its popularity, and it remains the preferred method today in numerous applications. In particular, Adam and its variant AdamW (based on regularization principle) are quite efficient on natural language processing models.

3.5 Conclusion

Stochastic gradient methods rely on partial gradient information selected in a random fashion: as such, they are not guaranteed to succeed deterministically, and their convergence guarantees are typically weaker than that of gradient descent. Nevertheless, they have proven very successful in data-driven problems, wherein computing the gradient involves accessing a massive amount of data

points. In this setting, stochastic gradient approaches have an attractive, low per-iteration cost, and typically make fast, significant progress early on in the algorithmic run. The nature of the data is key to the performance of stochastic gradient: on standard datasets from machine and deep learning, it appears beneficial to resort to such techniques.

Multiple variants of stochastic gradient have been developed in academic and industrial contexts. Batch stochastic gradient techniques are among the most popular, as they allow to incorporate more than one sample into the gradient approximation: this leads to variance reduction, a concept that is driving some of the latest advances in stochastic gradient.

Bibliography

[1] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization Methods for Large-Scale Machine Learning. *SIAM Rev.*, 60:223–311, 2018.

[2] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, Cambridge, United Kingdom, 2004.

[3] S. J. Wright and B. Recht. *Optimization for Data Analysis*. Cambridge University Press, 2022.

Appendix A

Notations and mathematical tools

A.1 Notations

A.1.1 Generic notations

- Scalars (i.e. reals) are denoted by lowercase letters: $a, b, c, \alpha, \beta, \gamma$.
- Vectors are denoted by **bold** lowercase letters: $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$.
- Matrices are denoted by **bold** uppercase letters: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$.
- Sets are denoted by **bold** uppercase cursive letters : $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}, \boldsymbol{\mathcal{C}}$.
- A new operator or quantity is defined using $:=$.
- The following quantifiers are used throughout the notes: $\forall$ (for every), $\exists$ (it exists), $\exists!$ (it exists a unique), $\in$ (belongs to), $\subseteq$ (subset of), $\subset$ (proper subset).
- The Σ operator is used for sums. To lighten the notation, and in the absence of ambiguity, we may omit the first and last indices, or use one sum over multiple indices. As a result, the notations $\sum_{i=1}^{m} \sum_{j=1}^{n}$, $\sum_i \sum_j$ and $\sum_{i,j}$ may be used interchangeably.
- The Π operator is used for products. To lighten the notation, and in the absence of ambiguity, we may omit the first and last indices, or use one sum over multiple indices. As a result, the notations $\prod_{i=1}^{m} \prod_{j=1}^{n}$, $\prod_i \prod_j$ and $\prod_{i,j}$ may be used interchangeably.
- The notation $i = 1, \ldots, m$ indicates that the variable i takes all integer values between 1 and m.

A.1.2 Scalar and vector notations

- The set of natural numbers (nonnegative integers) is denoted by $\mathbb{N}$; the set of integers is denoted by $\mathbb{Z}$.
- The set of real numbers is denoted by $\mathbb{R}$. Our notations for the subset of nonnegative real numbers and the set of positive real numbers are $\mathbb{R}_+$ and $\mathbb{R}_{++}$, respectively. We also define the extended real line $\overline{\mathbb{R}} := \mathbb{R} \cup \{-\infty, \infty\}$.

- The notation $\mathbb{R}^d$ is used for the set of vectors with $d \in \mathbb{N}$ real components; although we do not explicitly indicate it in the rest of these notes, we always assume that $d \geq 1$.

- A vector $\boldsymbol{w} \in \mathbb{R}^d$ is thought as a column vector, with $w_i \in \mathbb{R}$ denoting its i-th coordinate in the canonical basis of $\mathbb{R}^d$. We thus write $\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}$, or, in a compact form, $\boldsymbol{w} = [w_i]_{1 \leq \imath \leq d}$.

- Given a column vector $\boldsymbol{w} \in \mathbb{R}^d$, the corresponding row vector is denoted by $\boldsymbol{w}^{\mathrm{T}}$, so that $\boldsymbol{w}^{\mathrm{T}} = [w_1 \ \cdots \ w_d]$ and $[\boldsymbol{w}^{\mathrm{T}}]^{\mathrm{T}} = \boldsymbol{w}$.

- For any integer $d \geq 1$, the vectors $\mathbf{0}_d$ and $\mathbf{1}_d$ correspond to the vectors of $\mathbb{R}^d$ for which all elements are 0 or 1, respectively. For simplicity, we may write $\boldsymbol{w} \geq 0$ to indicate that all components of $\boldsymbol{w}$ are nonnegative.

A.1.3 Matrix notations

- We use $\mathbb{R}^{m \times n}$ to denote the set of real rectangular matrices with m rows and n columns, where m et n will always be assumed to be at least 1. If $m = n$, $\mathbb{R}^{n \times n}$ refers to the set of square matrices of size n.

- We identify a matrix in $\mathbb{R}^{m \times 1}$ with its corresponding column vector in $\mathbb{R}^m$.

- Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, A_{ij} or $[\boldsymbol{A}]_{ij}$ refers to the coefficient from the i-th row and the j-th column of $\boldsymbol{A}$. Provided this notation is not ambiguous, we use the notations $\boldsymbol{A}$, $[A_{ij}]_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}}$ and $[A_{ij}]$ interchangeably.

- Depending on the context, we may use $\boldsymbol{a}_i^{\mathrm{T}}$ to denote the i-th row of $\boldsymbol{A}$ or $\boldsymbol{a}_j$ to denote the j-th column of $\boldsymbol{A}$, leading to $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{a}_m^{\mathrm{T}} \end{bmatrix}$ or $\boldsymbol{A} = [\boldsymbol{a}_1 \ \cdots \ \boldsymbol{a}_n]$, respectively.

- The diagonal of a square matrix $\boldsymbol{A} \in \mathbb{R}^{d \times d}$ is given by the coefficients A_{ii}. The trace of such a matrix is $\mathrm{trace}(\boldsymbol{A}) := \sum_{i=1}^{d} A_{ii}$.

- Given $\boldsymbol{A} = [\boldsymbol{A}_{ij}] \in \mathbb{R}^{m \times n}$, the *transpose of matrix* $\boldsymbol{A}$, denoted by $\boldsymbol{A}^{\mathrm{T}}$ (read "$\boldsymbol{A}$ transpose"), is defined as the matrix in $\mathbb{R}^{n \times m}$ (or "n-by-m matrix") such that

$$\forall i = 1 \ldots m, \ \forall j = 1 \ldots n, \quad [\boldsymbol{A}^{\mathrm{T}}]_{ji} = A_{ij}.$$

  Note that this generalizes the notation used for row vectors.

- For every $n \geq 1$, $\mathbf{I}_n$ refers to the identity matrix in $\mathbb{R}^{n \times n}$ (with 1s on the diagonal and 0s elsewhere).

A.2 Mathematical tools

Optimization has mathematical roots in real analysis, mostly through differential calculus. Linear algebra structures also play a major role in optimization (and data science). In this section, we list the basic results that will be used in the course.

For a deeper dive into these notions, the following links are recommended.

- For linear algebra:
  - https://www.ceremade.dauphine.fr/∼carlier/polyalgebre.pdf (in Frnech);
  - http://vmls-book.stanford.edu/vmls.pdf (Chapters 1 to 3, in English).
- For differential calculus:
  - https://www.ceremade.dauphine.fr/∼bouin/ens1819/Cours_Bolley.pdf (in French);
  - https://sebastianraschka.com/pdf/books/dlb/appendix_d_calculus.pdf (in English).

A.2.1 Vector linear algebra

We always consider vectors in the normed vector space $\mathbb{R}^d$, of dimension d. The following operations are defined in this space:

- For any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$, the sum of $\boldsymbol{x}$ and $\boldsymbol{y}$ is denoted by $\boldsymbol{x} + \boldsymbol{y} = [x_i + y_i]_{1 \leq i \leq d}$;
- For any $\lambda \in \mathbb{R}$, we define $\lambda \boldsymbol{x} \overset{n}{=} \lambda \cdot \boldsymbol{x} = [\lambda x_i]_{1 \leq i \leq d}$. In this context, the real value λ is called a *scalar*.

Using these operations, we can build **linear combinations** of vectors in $\mathbb{R}^d$ that produce a vector in $\mathbb{R}^d$ of the form $\sum_{i=1}^p \lambda_i \boldsymbol{x}_i$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $\lambda_i \in \mathbb{R}$ for any $i = 1, \ldots, p$.

The matrix space $\mathbb{R}^{n \times d}$ can also be endowed with a vector space structure of dimension nd:

- For any $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n \times d}$, the sum of $\boldsymbol{A}$ and $\boldsymbol{B}$ is denoted by $\boldsymbol{A} + \boldsymbol{B} = [\boldsymbol{A}_{ij} + \boldsymbol{B}_{ij}]_{\substack{1 \leq i \leq d \\ 1 \leq j \leq d}}$;
- For any scalar $\lambda \in \mathbb{R}$, we define $\lambda \boldsymbol{A} \overset{n}{=} \lambda \cdot \boldsymbol{A} = [\lambda \boldsymbol{A}_{ij}]_{\substack{1 \leq i \leq n \\ 1 \leq j \leq d}}$.

Definition A.2.1 *A set $\mathcal{S} \subseteq \mathbb{R}^d$ satisfying the conditions*

1. $\mathbf{0}_d \in \mathcal{S}$;
2. $\forall (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S},\ \boldsymbol{x} + \boldsymbol{y} \in \mathcal{S}$;
3. $\forall \boldsymbol{x} \in \mathcal{S},\ \forall \lambda \in \mathbb{R},\ \lambda \boldsymbol{x} \in \mathcal{S}$.

is called a (linear) subspace *of $\mathbb{R}^d$.*

Definition A.2.2 *Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p$ be p vectors in $\mathbb{R}^d$. The span (or linear span) of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p$, denoted by $\mathrm{Span}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p)$, is the subspace of $\mathbb{R}^d$ defined by*

$$\mathrm{Span}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p) := \left\{ \boldsymbol{x} = \sum_{i=1}^p \alpha_i \boldsymbol{x}_i \middle| \alpha_i \in \mathbb{R}\ \forall i \right\}.$$

We now recall various properties of vector sets.

Definition A.2.3
- *The vectors in a set* $\{\boldsymbol{x}_i\}_{i=1}^k \subset \mathbb{R}^n$ *are called linearly independent if for any scalars* $\lambda_1, \ldots, \lambda_k$ *satisfying* $\sum_{i=1}^k \lambda_i \boldsymbol{x}_i = 0$*, we have* $\lambda_1 = \cdots = \lambda_k = 0$*. In that case,* $k \leq n$*.*
- *If the above property does not hold, the vectors are called linearly dependent.*
- *A spanning set is a set of vectors* $\{\boldsymbol{x}_i\} \subset \mathbb{R}^n$ *such that their span is* $\mathbb{R}^n$*.*
- *A set of vectors* $\{\boldsymbol{x}_i\}_{i=1}^n \subset \mathbb{R}^n$ *is a basis if it is both linearly independent and a spanning set. In that case, any vector in* $\mathbb{R}^n$ *can be written as a uniquely defined linear combination of the* $\boldsymbol{x}_i$*s. Any basis in* $\mathbb{R}^n$ *has exactly* n *vectors.*

Since the size of a basis in $\mathbb{R}^n$ is n, we say that the dimension of the space is n. Consequently, any subspace of $\mathbb{R}^n$ has dimension at most n.

Example A.2.1 *Any vector* $\boldsymbol{x}$ *in* $\mathbb{R}^n$ *can be written as* $\boldsymbol{x} = \sum_{i=1}^n x_i \boldsymbol{e}_i$*, where* $\boldsymbol{e}_i = [0 \cdots 0 \ 1 \ 0 \cdots 0]^{\mathrm{T}}$ *is the* i*th vector of the canonical basis (with a* 1 *in the* i*th coordinate).*

Norm and scalar product Using a Euclidean norm and its associated scalar product allows to compare vectors by measuring the distance between them. This ability is particularly useful to establish that a sequence of vector generated by an optimization method converges toward the solution of a given problem.

Definition A.2.4 *The* **Euclidean norm** $\|\cdot\|$ *on* $\mathbb{R}^n$ *is defined by*

$$\forall \boldsymbol{x} \in \mathbb{R}^n, \quad \|\boldsymbol{x}\| := \sqrt{\sum_{i=1}^n x_i^2}.$$

Remark A.2.1 *This is indeed a norm, since it fulfills the four axioms that define what a norm is:*

1. $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n, \ \|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$*;*
2. $\|\boldsymbol{x}\| = 0 \ \Leftrightarrow \boldsymbol{x} = \boldsymbol{0}_{\mathbb{R}^n}$*;*
3. $\forall \boldsymbol{x}, \ \|\boldsymbol{x}\| \geq 0$*;*
4. $\forall \boldsymbol{x} \in \mathbb{R}^n, \forall \lambda \in \mathbb{R}, \|\lambda \boldsymbol{x}\| = |\lambda| \|\boldsymbol{x}\|$.

A vector $\boldsymbol{x} \in \mathbb{R}^n$ is called a unit vector if $\|\boldsymbol{x}\| = 1$.

Definition A.2.5 *For any vectors* $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$*, the* **scalar product** *derived from the Euclidean norm is a function of* $\boldsymbol{x}$ *and* $\boldsymbol{y}$*, denoted by* $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$*, defined as follows:*

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} := \sum_{i=1}^n x_i y_i.$$

Two vectors $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *are called* orthogonal *if* $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} = 0$.

Note that $\boldsymbol{y}^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$, hence the scalar product defines a "product" between a row vector and a column vector.

Proposition A.2.1 *Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be two vectors in $\mathbb{R}^n$. Then, the following properties hold*

i) $\|\boldsymbol{x}+\boldsymbol{y}\|^2 = \|\boldsymbol{x}\|^2 + 2\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} + \|\boldsymbol{y}\|^2$;

ii) $\|\boldsymbol{x}-\boldsymbol{y}\|^2 = \|\boldsymbol{x}\|^2 - 2\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} + \|\boldsymbol{y}\|^2$;

iii) $\|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2 = \frac{1}{4}\left(\|\boldsymbol{x}+\boldsymbol{y}\|^2 + \|\boldsymbol{x}-\boldsymbol{y}\|^2\right)$;

iv) **Cauchy-Schwarz inequality** *:*

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n, \qquad \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} \leq \|\boldsymbol{x}\|\|\boldsymbol{y}\|.$$

Remark A.2.2 *The last inequality is a key result in both linear algebra and analysis. In this course, it will play a major role in deriving Taylor-type inequalities.*

A.2.2 Matrix linear algebra

We can define the product of two matrices that have compatible dimensions. More precisely, for any $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{B} \in \mathbb{R}^{n\times p}$, the product matrix $\boldsymbol{AB}$ is defined as the matrix $\boldsymbol{C} \in \mathbb{R}^{m\times p}$ such that

$$\forall i = 1,\dots,m,\ \forall j = 1,\dots,p, \quad \boldsymbol{C}_{ij} = \sum_{k=1}^{n} \boldsymbol{A}_{ik}\boldsymbol{B}_{kj}.$$

Using this definition, the product of a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ with a (column) vector $\boldsymbol{x} \in \mathbb{R}^n$ is the vector $\boldsymbol{y} \in \mathbb{R}^m$ given by

$$\forall i = 1,\dots,m,\ y_i = \sum_{j=1}^{n} \boldsymbol{A}_{ij}x_j.$$

Remark A.2.3 *Note that the scalar product on $\mathbb{R}^n$ corresponds to the matrix product for matrices of sizes $1\times n$ and $n\times 1$: the result of this operation is a 1×1 matrix, that is, a scalar.*

When one work with matrices, the following subspaces are of interest.

Definition A.2.6 (Matrix subspaces) *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$.*

- *The* null space *of $\boldsymbol{A}$ is the subspace*

$$\mathrm{Null}(\boldsymbol{A}) \ := \ \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}_m\}$$

- *The* range space *of $\boldsymbol{A}$ is the subspace*

$$\mathrm{Range}(\boldsymbol{A}) \ := \ \{\boldsymbol{y} \in \mathbb{R}^m \mid \exists \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}\}$$

  The dimension of this subspace is called the **rank** *of $\boldsymbol{A}$. We denote it by $\mathrm{rank}(\boldsymbol{A})$. One always has $\mathrm{rank}(\boldsymbol{A}) \leq \min\{m,n\}$.*

Theorem A.2.1 (Rank-nullity theorem) *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$. Then,*

$$\dim(\ker(\boldsymbol{A})) + \operatorname{rang}(\boldsymbol{A}) = n.$$

Definition A.2.7 (Matrix norms) *Consider the space $\mathbb{R}^{m\times n}$. The operator norm $\|\cdot\|$ and the Frobenius norm $\|\cdot\|_F$ are defined by*

$$\forall \boldsymbol{A} \in \mathbb{R}^{m\times n}, \begin{cases} \|\boldsymbol{A}\| & := \max\limits_{\substack{\boldsymbol{x}\in\mathbb{R}^n \\ \boldsymbol{x}\neq \boldsymbol{0}_n}} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|} = \max\limits_{\substack{\boldsymbol{x}\in\mathbb{R}^n \\ \|\boldsymbol{x}\|=1}} \|\boldsymbol{A}\boldsymbol{x}\| \\ \|\boldsymbol{A}\|_F & := \sqrt{\sum_{\substack{1\leq i\leq m \\ 1\leq j\leq n}} \boldsymbol{A}_{ij}^2}. \end{cases}$$

Definition A.2.8 (Symmetric matrix) *A square matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is called symmetric if $\boldsymbol{A}^{\mathrm{T}} = \boldsymbol{A}$.*

The set of symmetric matrices in $\mathbb{R}^{n\times n}$ is denoted by $\mathcal{S}^n$.

Definition A.2.9 (Invertible matrix) *A square matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is called invertible if there exists $\boldsymbol{B} \in \mathbb{R}^{n\times n}$ such that $\boldsymbol{B}\boldsymbol{A} = \boldsymbol{A}\boldsymbol{B} = \boldsymbol{I}_n$ (where we recall that $\boldsymbol{I}_n$ denotes the identity matrix in $\mathbb{R}^{n\times n}$).*

When it exists, such a matrix $\boldsymbol{B}$ is unique. It is then called **the inverse of $\boldsymbol{A}$** *and denoted by $\boldsymbol{A}^{-1}$.*

Definition A.2.10 (Positive (semi)definite matrix) *A square, symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is called positive semidefinite if*

$$\forall \boldsymbol{x} \in \mathbb{R}^n, \quad \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} \geq 0,$$

which we write $\boldsymbol{A} \succeq 0$.

Such a matrix is called positive definite when $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} > 0$ for any nonzero vector $\boldsymbol{x}$. We write this as $\boldsymbol{A} \succ 0$.

Definition A.2.11 (Orthogonal matrix) *A square matrix $\boldsymbol{P} \in \mathbb{R}^{n\times n}$ is called orthogonal if $\boldsymbol{P}^{\mathrm{T}} = \boldsymbol{P}^{-1}$.*

More generally, a matrix $\boldsymbol{Q} \in \mathbb{R}^{m\times n}$, where $m \leq n$, is called orthogonal if $\boldsymbol{Q}\boldsymbol{Q}^{\mathrm{T}} = \boldsymbol{I}_m$ (the columns of $\boldsymbol{Q}$ are orthonormal in $\mathbb{R}^m$).

When $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$ is orthogonal, then so is its transpose $\boldsymbol{Q}^{\mathrm{T}}$ (this result only applies to square matrices). Orthogonal matrices have the following desirable property.

Lemma A.2.1 *Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{U} \in \mathbb{R}^{m\times m}$, $\boldsymbol{V} \in \mathbb{R}^{n\times n}$ be two orthogonal matrices. Then,*

$$\|\boldsymbol{A}\| = \|\boldsymbol{U}\boldsymbol{A}\| = \|\boldsymbol{A}\boldsymbol{V}\| \quad \text{and} \quad \|\boldsymbol{A}\|_F = \|\boldsymbol{U}\boldsymbol{A}\|_F = \|\boldsymbol{A}\boldsymbol{V}\|_F,$$

i.e. multiplying by an orthogonal matrix preserves the norm.

As a corollary of the previous lemma, we observe that an orthogonal matrix $\boldsymbol{Q} \in \mathbb{R}^{m\times n}$ with $m \leq n$ must satisfy $\|\boldsymbol{Q}\| = 1$ and $\|\boldsymbol{Q}\|_F = \sqrt{m}$.

Definition A.2.12 (Eigenvalue) *Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$. A scalar $\lambda \in \mathbb{R}$ is called an* **eigenvalue of $\boldsymbol{A}$** *if*

$$\exists \boldsymbol{v} \in \mathbb{R}^n, \boldsymbol{v} \neq \boldsymbol{0}_n, \quad \boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{v}.$$

The vector $\boldsymbol{v}$ is called an **eigenvector associated with the eigenvalue λ**. *The set of eigenvalues of $\boldsymbol{A}$ is the spectrum of $\boldsymbol{A}$.*

The span of eigenvectors associated to the same eigenvalue is called the eigenspace. Its dimension corresponds to the multiplicity of the eigenvalue relatively to the matrix.

Proposition A.2.2 *For any matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, the following holds:*

- *$\boldsymbol{A}$ has n complex eigenvalues.*
- *If $\boldsymbol{A}$ is symmetric positive semidefinite (resp. definite), then its eigenvalues are real nonnegative (resp. real positive).*
- *The null space of $\boldsymbol{A}$ is spanned by the eigenvectors associated with the 0 eigenvalue.*

Theorem A.2.2 (Eigenvalue decomposition theorem) *Any symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ has an* **eigenvalue decomposition** *of the form*

$$\boldsymbol{A} = \boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^T,$$

where $\boldsymbol{P} \in \mathbb{R}^{n\times n}$ is an orthogonal matrix,and $\boldsymbol{\Lambda} \in \mathbb{R}^{n\times n}$ is a diagonal matrix that contains the n eigenvalues of $\boldsymbol{A}$ $\lambda_1, \ldots, \lambda_n$ on its diagonal.

The eigenvalue decomposition is not unique, but the set of eigenvalues that appears in the decomposition is uniquely defined.

Remark A.2.4 *There are matrices that possess an eigenvalue decomposition of the form $\boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^{-1}$, where $\boldsymbol{P}$ is invertible (but not necessarily orthogonal). Those matrices are called diagonalizable.*

Link with singular value decomposition Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$. In general, $m \neq n$ and the notion of eigenvalue that we introduced above does not apply. However, we can always consider the eigenvalues of

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} \in \mathbb{R}^{n\times n} \qquad \text{and} \qquad \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}} \in \mathbb{R}^{m\times m}.$$

These matrices are real and symmetric, hence they can be diagonalized. This property is what gives rise to the singular value decomposition (or *SVD*).

A.2.3 Calculus

Note: This section gives additional background to the concepts and properties introduced in Chapter 1.

Definition A.2.13 (Continuity) *A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is called* **continuous in $\boldsymbol{x} \in \mathbb{R}^n$** *if*

$$\forall \epsilon > 0,\ \exists \delta > 0,\ \forall \boldsymbol{y} \in \mathbb{R}^n, \quad \|\boldsymbol{y} - \boldsymbol{x}\| < \delta \quad \Rightarrow \quad \|f(\boldsymbol{y}) - f(\boldsymbol{x})\| < \epsilon.$$

The function f is continuous on a set $\mathcal{A} \subseteq \mathbb{R}^n$ if it is continuous at every point of $\mathcal{A}$. When $\mathcal{A} = \mathbb{R}^n$, we simply say that f is continuous.

Remark A.2.5 *In certain textbooks, the notion above is termed* uniform continuity*. For simplicity of exposure, we will use it as our definition of continuity.*

An alternate characterization of continuity based on sequences is given below. Sequences typically appear when considering iterative algorithms, hence the relevance of this notion here.

Definition A.2.14 (Continuity (sequential definition)) *A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is continuous at $\boldsymbol{x} \in \mathbb{R}^n$ if*

$$\forall \{\boldsymbol{x}_n\} \in (\mathbb{R}^n)^{\mathbb{N}},\ \{\boldsymbol{x}_n\} \to \boldsymbol{x}, \quad \lim_{n\to\infty} f(\boldsymbol{x}_n) = f(\boldsymbol{x}).$$

Example A.2.2 *A linear map $f : \mathbb{R}^n \to \mathbb{R}^m$, where $f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$ for any $\boldsymbol{x} \in \mathbb{R}^n$ with $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$, is a continuous function on $\mathbb{R}^n$.*

Definition A.2.15 (Differentiability Jacobian matrix) *A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is called* **differentiable** *at a point $\boldsymbol{x} \in \mathbb{R}^n$ if there exists a matrix $\boldsymbol{J}_f(\boldsymbol{x}) \in \mathbb{R}^{m\times n}$ such that*

$$\lim_{\substack{\boldsymbol{z}\to\boldsymbol{x} \\ \boldsymbol{z}\neq\boldsymbol{x}}} \frac{\|f(\boldsymbol{z}) - f(\boldsymbol{x}) - \boldsymbol{J}_f(\boldsymbol{x})(\boldsymbol{z}-\boldsymbol{x})\|}{\|\boldsymbol{z}-\boldsymbol{x}\|} = 0.$$

- *$\boldsymbol{J}_f(\boldsymbol{x})$ is called the* **Jacobian** *of f at $\boldsymbol{x}$, and is uniquely defined.*
- *If $f(\cdot) = [f_1(\cdot), \ldots, f_m(\cdot)]^{\mathrm{T}}$, then*

$$\forall 1 \le i \le m,\ \forall 1 \le j \le n,\ [\boldsymbol{J}_f(\boldsymbol{x})]_{ij} = \frac{\partial f_i}{\partial x_j}(\boldsymbol{x}).$$

The following special cases are instrumental to optimization and basic analysis.

Corollary A.2.1

- *When $m = 1$, we define the (column) vector $\nabla f(\boldsymbol{x}) \equiv \boldsymbol{J}_f(\boldsymbol{x})^{\mathrm{T}}$, called the* **gradient** *of f at $\boldsymbol{x}$. In this case, the gradient is the vector of partial derivatives of f:*

$$\forall i = 1, \ldots, n, \qquad \nabla f(\boldsymbol{x}) = \left[\frac{\partial f}{\partial x_i}(\boldsymbol{x})\right]_{1\le i\le n}.$$

- *When $n = m = 1$, both the Jacobian and the gradient are equivalent to a scalar la matrice Jacobienne et le vecteur $f'(x) \equiv \nabla f(x) \equiv \boldsymbol{J}_f(\boldsymbol{x})^{\mathrm{T}}$, called the derivative of f at $\boldsymbol{x}$.*

In these notes, we assume familiarity with the common derivative formulas for functions from $\mathbb{R}$ to $\mathbb{R}$. More complex formulas are typically obtained thanks to the rule below.

Theorem A.2.3 (Chain rule) *If $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ and $g : \mathbb{R}^m \mapsto \mathbb{R}^p$ are both differentiable, respectively on $\mathbb{R}^n$ and $\mathbb{R}^m$, then $h : \mathbb{R}^n \mapsto \mathbb{R}^p$ is differentiable on $\mathbb{R}^n$ and*

$$\forall \boldsymbol{x} \in \mathbb{R}^n, \quad \boldsymbol{J}_h(\boldsymbol{x}) = \boldsymbol{J}_g(f(\boldsymbol{x}))\boldsymbol{J}_f(\boldsymbol{x}).$$

Remark A.2.6 *Special cases of the chain rule:*

- *$m = p = 1$: $\nabla h(\boldsymbol{x}) = g'(f(\boldsymbol{x}))\nabla f(\boldsymbol{x})$;*
- *$n = m = p = 1$: $h'(x) = g'(f(x))f'(x)$.*

Theorem A.2.4 (Mean-value theorem in dimension 1) *Let $f : [a, b] \to \mathbb{R}$. If f is continuous on $[a, b]$ and differentiable on (a, b), there exists $c \in (a, b)$ such that*

$$\frac{f(b) - f(a)}{b - a} = f'(c).$$

Definition A.2.16 (Taylor expansion) *Let $f : [a, b] \mapsto \mathbb{R}$ be $\mathcal{C}^1$ on $[a, b]$, then*

$$\begin{aligned} f(b) &= f(a) + f'(c)(b - a) \qquad \text{where } c \in [a, b] \\ f(b) &= f(a) + \int_0^1 f'(a + t(b - a))(b - a)\, dt. \end{aligned}$$

Theorem A.2.5 (Mean-value theorem in dimension d) *Let $f : \mathbb{R}^d \to \mathbb{R}$ $f \in \mathcal{C}^1(\mathbb{R}^d)$. For any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$, $\boldsymbol{x} \neq \boldsymbol{y}$, there exists $t \in (0, 1)$ such that*

$$f(\boldsymbol{y}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x} + t(\boldsymbol{y} - \boldsymbol{x}))^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}).$$

Definition A.2.17 (Lipschitz continuity) *A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is L-**Lipschitz continuous** on $\mathcal{A} \subset \mathbb{R}^n$ if*

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{A}^2, \quad \|f(\boldsymbol{x}) - f(\boldsymbol{y})\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|.$$

Proposition A.2.3 *Any Lipschitz continuous function on a set is continuous on this set.*

Definition A.2.18 (Function classes) *Let $f : \mathbb{R}^d \to \mathbb{R}$.*

- *We say that f is $\mathcal{C}^p(\mathbb{R}^d)$ (or simply $\mathcal{C}^p$) if it is differentiable p times with a continuous pth-order derivative (in which case all derivatives up to order p are continuous). The class of $\mathcal{C}^\infty$ functions is the intersection of all $\mathcal{C}^p$ with $p \in \mathbb{N}$.*
- *We say that f is $\mathcal{C}_L^{p,p}(\mathbb{R}^d)$ (or simply $\mathcal{C}_L^{p,p}$) if it is differentiable p times and its pth-order derivative is L-Lipschitz continuous.*

Theorem A.2.6 (Taylor expansion of order 1) *Let $f \in \mathcal{C}^1(\mathbb{R}^d)$. For any vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ of $\mathbb{R}^d$, we have*

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

Moreover, if $f \in \mathcal{C}_L^{1,1}(\mathbb{R}^d)$, then

$$f(\boldsymbol{y}) \leq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}) + \frac{L}{2}\|\boldsymbol{y} - \boldsymbol{x}\|^2. \tag{A.2.1}$$

Theorem A.2.7 (Taylor expansion of order 2) *Let $f \in \mathcal{C}^2(\mathbb{R}^d)$. For any vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ of $\mathbb{R}^d$, one has*

$$f(\boldsymbol{y}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}) + \frac{1}{2}\int_0^1 (\boldsymbol{y} - \boldsymbol{x})^{\mathrm{T}} \nabla^2 f(\boldsymbol{x} + t\,(\boldsymbol{y} - \boldsymbol{x}))(\boldsymbol{y} - \boldsymbol{x})\, dt.$$

Moreover, if $f \in \mathcal{C}_L^{2,2}(\mathbb{R}^d)$, then

$$f(\boldsymbol{y}) \leq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{x}) + \frac{1}{2}(\boldsymbol{y} - \boldsymbol{x})^{\mathrm{T}} \nabla^2 f(\boldsymbol{x})(\boldsymbol{y} - \boldsymbol{x}) + \frac{L}{2}\|\boldsymbol{y} - \boldsymbol{x}\|^3. \tag{A.2.2}$$