Optimization Methods - Optimization Techinques for Machine Learning: Lecture Notes

Matteo Lapucci
matteo.lapucci@unifi.it

Department of Information Engineering, Università degli Studi di Firenze

Contents

1 Preliminaries

We report in this section some preliminary concepts, definitions and properties that will be used recurrently in these notes.

Notation
We denote by $e \in \mathbb{R}^n$ the vector of all ones in the n-dimensional Euclidean space. We denote by $e_i \in \mathbb{R}^n$ the i-th element of the canonical basis, i.e., the vector with all zero components except for the i-th component being equal to 1. Given two vectors $u, v \in \mathbb{R}^n$, the notation $u^T v$ is a way to denote the dot product between u and v, i.e., $u^T v = \sum_{i=1}^n u_i v_i$. By $\|\cdot\|$ we denote the norm function; if not specified otherwise, we implicitly assume that the Euclidean norm is considered (in that case $\|x\|^2 = x^T x$). We denote by $\mathcal{S}_n \subset \mathbb{R}^{n\times n}$ the set of symmetric square matrices of size $n \times n$. Given a matrix $A \in \mathcal{S}_n$ (whose eigenvalues we know being real numbers), we denote by $\lambda_{\min}(A)$ and $\lambda_{\max}(A)$ the smallest and the largest eigenvalues of A respectively. We denote by $\mathcal{L}_\alpha(f)$ the level set $\{x \in \mathbb{R}^n \mid f(x) = \alpha\}$.

Definition 1.1. A matrix $A \in \mathcal{S}_n$ is

- *positive semidefinite*, and we denote it by $A \succeq 0$, if $x^T Ax \geq 0$ for all $x \in \mathbb{R}^n$;
- *positive definite*, and we denote it by $A \succ 0$, if $x^T Ax > 0$ for all $x \in \mathbb{R}^n$, $x \neq 0$.

Proposition 1.1. *A matrix $A \in \mathcal{S}_n$ is positive semidefinite if and only if all eigenvalues of A are nonnegative, i.e., $\lambda_{min}(A) \geq 0$. Similarly, A is positive definite if and only if all eigenvalues of A are strictly positive, i.e., $\lambda_{min}(A) > 0$.*

Proposition 1.2. *Let $A \in \mathcal{S}_n$, $A \succeq 0$. For all $x \in \mathbb{R}^n$, we have*

- $\lambda_{min}(A)\|x\|^2 \leq x^T Ax \leq \lambda_{max}(A)\|x\|^2$;
- $\lambda_{min}(A)\|x\| \leq \|Ax\| \leq \lambda_{max}(A)\|x\|$.

Definition 1.2. A continuous function $f : \mathbb{R}^n \to \mathbb{R}$ is said *coercive* if $\lim_{k\to\infty} f(x^k) = +\infty$ for all sequences $\{x^k\} \subseteq \mathbb{R}^n$ such that $\lim_{k\to\infty} \|x^k\| = +\infty$.

Proposition 1.3. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuous function. Then, f is coercive if and only if all the level sets of f are compact.*

Definition 1.3. Let $f : \mathbb{R}^n \to \mathbb{R}$. We say that a function has *directional derivative* $\mathcal{D}_f(x, d)$ at a point $x \in \mathbb{R}^n$ along direction $d \in \mathbb{R}^n$ if the limit

$$\lim_{t\to 0^+} \frac{f(x + td) - f(x)}{t} = \mathcal{D}_f(x, d)$$

exists and is finite.

Definition 1.4. Let $f : \mathbb{R}^n \to \mathbb{R}$. We say that a function has *partial derivative* $\frac{\partial f(x)}{\partial x_j}$ at a point $x \in \mathbb{R}^n$ w.t.r. variable x_j if f has a directional derivative along e_j and

$$\mathcal{D}_f(x, e_j) = \frac{\partial f(x)}{\partial x_j}.$$

Definition 1.5. Let $f : \mathbb{R}^n \to \mathbb{R}$ and assume f admits at a point x the partial derivative w.r.t. each variable x_i, $i = 1, \ldots, n$. We define the *gradient* $\nabla f(x)$ of f at x as the vector given by

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{pmatrix}.$$

Definition 1.6. Let $f : \mathbb{R}^n \to \mathbb{R}$. We say that f is *continuously differentiable* over $\mathbb{R}^n$, and we denote it by $f \in \mathcal{C}^1(\mathbb{R}^n)$, if the gradient $\nabla f(x)$ exists for all $x \in \mathbb{R}^n$ and the function $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ is continuous on $\mathbb{R}^n$.

Proposition 1.4. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function. Then, for all $x \in \mathbb{R}^n$ and $d \in \mathbb{R}^n$, we have*

$$\mathcal{D}_f(x, d) = \nabla f(x)^T d.$$

Definition 1.7. Let $F : \mathbb{R}^n \to \mathbb{R}^m$ be a continuous vector function. We say that F is continuously differentiable if each component $F_i : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable and we define the *Jacobian matrix* $J_F : \mathbb{R}^n \to \mathbb{R}^{m \times n}$ as

$$J_F(x) = \begin{pmatrix} \nabla F_1(x)^T \\ \vdots \\ \nabla F_m(x)^T \end{pmatrix}.$$

Definition 1.8. Let $f : \mathbb{R}^n \to \mathbb{R}$, $f \in \mathcal{C}^1(\mathbb{R}^n)$. Assume, at a point $x \in \mathbb{R}^n$, that each component $\frac{\nabla f(x)}{\nabla x_i}$ of $\nabla f(x)$ admits the partial derivative w.r.t. each variable x_j, $j = 1, \ldots, n$. We define the *Hessian* matrix $\nabla^2 f(x)$ of f at x as the (symmetric) matrix given by the second order derivatives of f, i.e.,

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1^2} & \cdots & \frac{\partial f(x)}{\partial x_1 x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(x)}{\partial x_1 x_n} & \cdots & \frac{\partial f(x)}{\partial x_n^2} \end{pmatrix}.$$

Definition 1.9. Let $F : \mathbb{R}^n \to \mathbb{R}^m$. We say that f is *Lipschitz-continuous* with constant L if, for all $x, y \in \mathbb{R}^n$, we have

$$\|F(x) - F(y)\| \leq L\|x - y\|.$$

Definition 1.10. Let $f : \mathbb{R}^n \to \mathbb{R}$, $f \in \mathcal{C}^1(\mathbb{R}^n)$. We say that f is *L-smooth* if the gradient ∇f is a Lipschitz continuous function with Lipschitz constant L, i.e., if

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$$

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

Definition 1.11. Let $f : \mathbb{R}^n \to \mathbb{R}$. We say that f is *twice continuously differentiable* over $\mathbb{R}^n$, and we denote it by $f \in \mathcal{C}^2(\mathbb{R}^n)$, if the Hessian matrix $\nabla^2 f(x)$ exists for all $x \in \mathbb{R}^n$ and the function $\nabla f : \mathbb{R}^n \to \mathbb{R}^{n \times n}$ is continuous on $\mathbb{R}^n$.

Note that the Hessian can be seen as the Jacobian matrix of the gradient $\nabla f(x)$, i.e., $\nabla^2 f(x) = J_{\nabla f}(x)$.

Proposition 1.5. *Let $f(x) = \frac{1}{2}x^TQx + c^Tx$, with $Q \in \mathcal{S}_n$ and $x \in \mathbb{R}^n$. Then we have:*

- $\nabla f(x) = Qx + c$;
- $\nabla^2 f(x) = Q$.

Proposition 1.6. *Let $f : \mathbb{R}^n \to \mathbb{R}$. Then:*

- *(**Mean value theorem**) If $f \in \mathcal{C}^1(\mathbb{R}^n)$, for all $x \in \mathbb{R}^n$ and $d \in \mathbb{R}^n$, there exists $t \in (0,1)$ such that $\xi = x + td$ and*

$$f(x+d) = f(x) + \nabla f(\xi)^T d;$$

moreover,

$$f(x+d) = f(x) + \nabla f(x)^T d + \beta(x,d),$$

where $\lim_{\|d\|\to 0} \frac{\beta(x,d)}{\|d\|} = 0$.

- *(**Taylor**) If $f \in \mathcal{C}^2(\mathbb{R}^n)$, for all $x \in \mathbb{R}^n$ and $d \in \mathbb{R}^n$, there exists $t \in (0,1)$ such that $\xi = x + td$ and*

$$f(x+d) = f(x) + \nabla f(x)^T d + \frac{1}{2} d^T \nabla^2 f(\xi) d;$$

moreover,

$$f(x+d) = f(x) + \nabla f(x)^T d + \frac{1}{2} d^T \nabla^2 f(x) d + \beta(x,d),$$

where $\lim_{\|d\|\to 0} \frac{\beta(x,d)}{\|d\|^2} = 0$.

Proposition 1.7 (Mean value theorem for integrals)**.** *Let $F : \mathbb{R}^n \to \mathbb{R}^m$ be a continuously differentiable function. Then, for all $x, y \in \mathbb{R}^n$, we have*

$$F(x) = F(y) + \int_0^1 J_F(y + t(x-y))(x-y)dt.$$

Proposition 1.8 (Descent Lemma)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be an L-smooth function. Then, for all $x \in \mathbb{R}^n$ and for all $d \in \mathbb{R}^n$, we have*

$$f(x+d) \leq f(x) + \nabla f(x)^T d + \frac{L}{2}\|d\|^2.$$

Definition 1.12. We say that a function $f : \mathbb{R}^n \to \mathbb{R}$ is

- *convex* if, for all $x, y \in \mathbb{R}^n$ and $\lambda \in [0,1]$, we have

$$f(\lambda(x) + (1-\lambda y)) \leq \lambda f(x) + (1-\lambda) f(y);$$

- *strictly convex* if, for all $x, y \in \mathbb{R}^n$ and $\lambda \in (0,1)$, we have

$$f(\lambda(x) + (1-\lambda y)) < \lambda f(x) + (1-\lambda) f(y);$$

- *strongly convex* if there exists $\mu > 0$ such that $f(x) - \mu\|x\|^2$ is a convex function, i.e., $f(x) = g(x) + \mu\|x\|^2$ with g being some convex function.

Proposition 1.9. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a strongly convex function. Then f is coercive and strictly convex.*

Proposition 1.10. *Let $f : \mathbb{R}^n \to \mathbb{R}$. The following properties hold:*

- *If $f \in \mathcal{C}^1(\mathbb{R}^n)$, then f is convex if and only if*

$$f(y) \geq f(x) + \nabla f(x)^T (y-x) \qquad \forall\, x, y, \in \mathbb{R}^n.$$

- *If $f \in \mathcal{C}^2(\mathbb{R}^n)$, then f is convex if and only*

$$\nabla^2 f(x) \succeq 0 \qquad \forall\, x \in \mathbb{R}^n.$$

Moreover, if $\nabla^2 f(x) \succ 0$ for all $x \in \mathbb{R}^n$, then f is strictly convex.

Definition 1.13. A function $f : \mathbb{R}^n \to \mathbb{R}$ is *strongly convex* if there exists $\mu > 0$ such that $f(x) - \mu\|x\|^2$ is a convex function, i.e., $f(x) = g(x) + \mu\|x\|^2$ being g a convex function.

2 Introduction to Optimization Problems

Operations Research (OR) is a field in mathematics focusing on modeling and finding solutions to real-world problems by means of mathematical tools. In particular, OR aims at describing real-world decision tasks as mathematical optimization problems and then solving them by some numerical procedure. *Mathematical Optimization* (or Mathematical Programming) is then a subfield within OR focused on the formal analysis of mathematical optimization problems and on the study of suitable algorithmic procedures to properly solve them. The workflow of OR approach and the subject of mathematical programming are summarized by the diagram in Figure1.

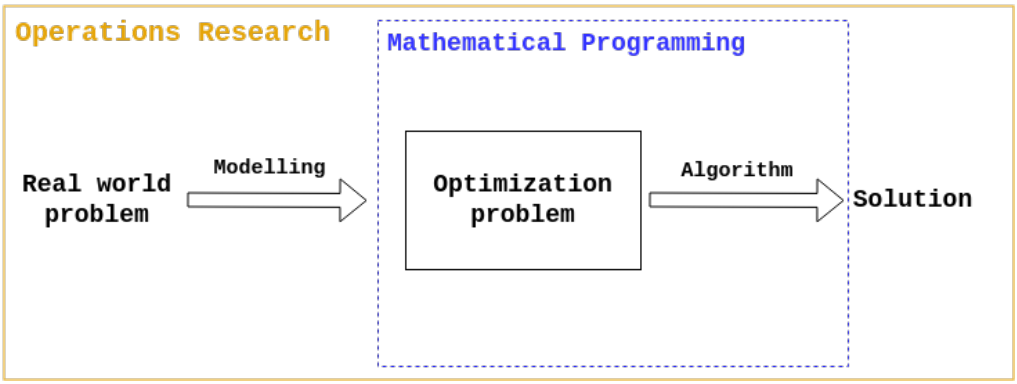


Figure 1: Operations Research and Mathematical Programming

The aim of these notes and of the course they originate from is to provide students with a thorough overview of some fundamental topics in mathematical programming. The core object around which all the forthcoming discussion revolves is thus the mathematical *optimization problem*. Formally, an optimization problem is defined as

$$\begin{aligned} &\min_{x} f(x) \\ &\text{s.t. } x \in S, \end{aligned} \tag{1}$$

where

- x is a vector of *variables* from some n-dimensional space; variables represent the quantities we can control in the real-world task and for which we have to decide the values to assign; as an example, x might represent the amount of hours per week a student decides to spend studying each subject in a semester;
- S is a subset of the variables space and is referred to as *feasible set*, defined by *constraints*; this set identifies the set of choices of values for the variables that are considered admissible to be employed in practice; a choice of variables values outside the feasible set is thus prohibited; constraints defining the feasible set might come by natural limitations (a student cannot study more than 24 hours a day) or by from hard design choices (a student might want not to study more than 40 hours per week);
- f is an *objective function* that, given a solution, numerically measures its quality; for example, given the choice of hours studied per subject, the objective is the average grade obtained at the exams; without loss of generality, we assume that lower values of f are associated with better solutions - we thus want to *minimize* the function; note that this is not restrictive: if we had to maximize $f(x)$, we could equivalently minimize $g(x) = -f(x)$.

Example 2.1 (Optimal Portfolio Selection Problem)**.** One of the most famous optimization problem is that of optimally choosing how to allocate financial assets. Roughly speaking, we there are n assets in the market we can invest a unit of budget. Each asset i is associated with an expected return of investment μ_i, i.e., the revenue of the investor, in expectation, per unit of investment. Moreover, for each pair of assets i and j, we have a covariance coefficient σ_{ij}: this coefficient is high if the return of two assets is positively correlated, negative if they are negatively correlated, zero if the trends of the two assets are independent. The quantity σ_{ii} denotes the variance i-th asset. The problem of selecting the investment based on these data can be modeled as an optimization problem:

- we define a vector of variables $x \in \mathbb{R}^n$; the value of each variable x_i denotes the quantity of capital invested in the i-th asset
- the feasible set is defined by the following constraints:
  - all investments must be nonnegative (we cannot sell shares of assets):
  $$x_i \geq 0 \; \forall i \qquad (x \geq 0)\,;$$
  - we have to spend the entire budget, i.e., the sum of the capital portions invested in all assets has to add up to 1:
  $$\sum_{i=1}^{n} x_i = 1 \qquad \left(e^T x = 1\right).$$
- The most widespread choice, for the objective function, is to require a balance between the total expected return of the investment and its risk. The expected return is, straightforwardly, given by the sum of the expected return of the assets multiplied by the capital invested in that asset, i.e.,
  $$\sum_{i=1}^{n} \mu_i x_i \qquad \left(\mu^T x\right);$$
  the risk is defined by the following sum
  $$\sum_{i=1}^{n}\sum_{j=1}^{n} \sigma_{ij} x_i x_j \qquad \left(x^T \Sigma x, \quad \Sigma_{ij} = \begin{cases} \frac{1}{2}\sigma_{ij} & \text{if } i \neq j \\ \sigma_{ii} & \text{if } i = j \end{cases}\right).$$
  The rationale of this definition of the risk is that we interpret as a risk investing into an asset with high variance (as there are high chances of getting a much lower return than expected); if σ_{ii} is high and x_i^2 is large we are taking a risk; moreover, if two assets are positively correlated $\sigma_{ij} > 0$, we are taking a risk investing in both, since there is a high chance that i goes worse than expected if j goes worse than expected; on the other hand, we want to promote the investment on negatively correlated assets, as one going bad would be compensated by the other rising up.

  Note that we want to minimize risk and maximize the return; in a minimization problem we would thus want to define an objective function which is a weighted sum of the risk and the negative of the expected return, i.e.,
  $$-\mu^T x + \lambda x^T \Sigma x,$$
  where λ is a scalar that balances the trade-off between risk and return.

All in all, the portfolio selection problem can finally be written (in vectorial form) as

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \; & -\mu^T x + \lambda\, x^T \Sigma x \\ \text{s.t. } & e^T x = 1, \\ & x \geq 0. \end{aligned}$$

In this notes we will be mainly interested in the Mathematical Programming aspects of Operations Research (see Figure 1). We will thus focus on the characterization of problems of the form (1) and on the algorithmic procedures to determine solutions of such problems.

Figure 2 shows a preliminary classification of optimization problems.

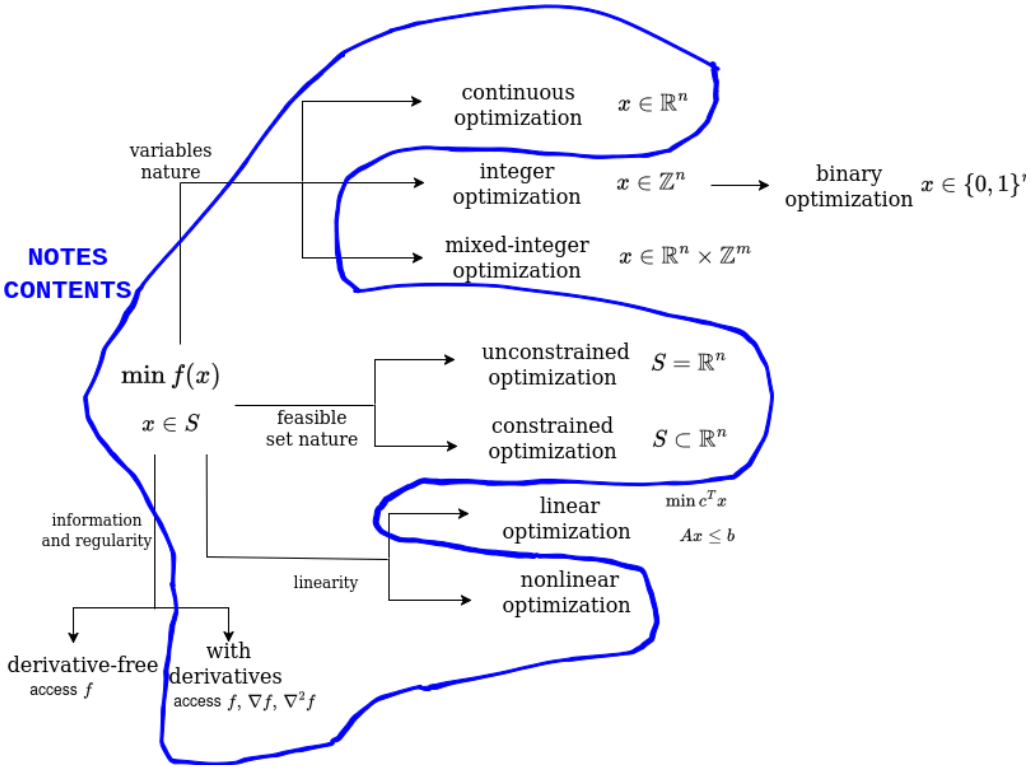


Figure 2: Classification of optimization problems and course contents.

Here, we will deal with continuous nonlinear optimization problems, both with and without constraints, having access to derivatives information.

Example 2.2. TBD - ESEMPIO: STIMA DI PARAMETRI IN MODELLI MATEMATICI

3 Optimization problems: characterization of solutions

TBD

3.1 Existence of Optimal solutions

TBD

3.2 Optimality Conditions

TBD

3.3 The unconstrained case

TBD

3.4 The constrained case

TBD

4 Unconstrained Optimization Algorithms

4.1 Iterative Optimization Methods

When we deal with unconstrained nonlinear optimization problems of the form

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

with $f : \mathbb{R}^n \to \mathbb{R}$ continuously differentiable, the best we can do in practice is to search candidate optimal solutions satisfying the *stationarity* condition $\nabla f(x) = 0$. In rare, fortunate cases, zeros of gradients can be found analytically, so that the problem can be solved in closed form.

In general, however, we will not have access to exact formulae for solving the problem. We thus have to rely on algorithms that construct a solution by means of an *iterative process*, i.e., methods that produce a sequence of solutions $\{x^k\}$ that shall get closer and closer to a stationary point. Iterative optimization algorithms are generally characterized by an update rule of the form

$$x^{k+1} = x^k + s_k,$$

i.e., the new point is obtained starting from the current solution and then shifting it by a displacement s_k. The algorithm then stops as soon as a the stationarity condition holds at some iterate x^k. There are several classes of algorithms, but the three arguably most relevant are the following ones (see also Figure 3):

(i) **line search based algorithms**: the update vector is structured as $s_k = \alpha_k d_k$, where $d_k \in \mathbb{R}^n$ is a *direction* in the Euclidean space and $\alpha_k \in \mathbb{R}_+$ is a scalar referred to as the *stepsize*; in this type of algorithms, the *search direction* d_k is first identified, then a suitable stepsize is chosen in order to properly set how large the shift along that direction shall be.

(ii) **trust region algorithms**: the update vector is defined as the best possible update for a suitable approximation (model) of the objective function in a neighborhood of the current solution:

$$s_k \in \arg\min_{x \in \Delta_k} m_k(x), \qquad m_k(x) \approx f(x) \; \forall x \in \Delta_k, \quad m_k(x^k) = f(x^k).$$

The acceptance of the update and the variation of the radius ρ_k of the *trust region* Δ_k depend on the improvement of the true objective function:

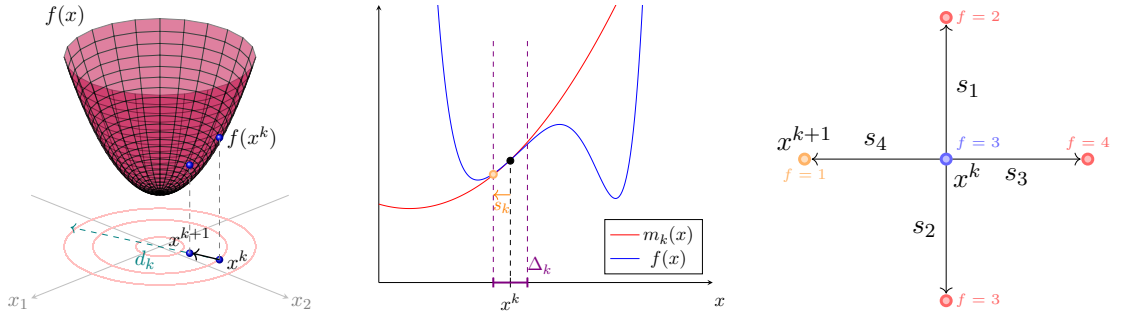
$$\begin{aligned} f(x^{k+1}) - f(x^k) \ll 0 &\implies x^{k+1} = x^k + s_k, \;\; \rho_{k+1} > \rho_k, \\ f(x^{k+1}) - f(x^k) < 0 &\implies x^{k+1} = x^k + s_k, \;\; \rho_{k+1} = \rho_k, \\ f(x^{k+1}) - f(x^k) \geq 0 &\implies x^{k+1} = x^k, \;\; \rho_{k+1} < \rho_k. \end{aligned}$$

(iii) **pattern search algorithms**: the update vector is the best one among a predefined set to be checked:

$$s_k \in \arg\min_{s_{i_k}, i_k = 1, \ldots, N_k} f(x^k + s_{i_k}).$$

Pattern search methods are often considered when we do not have access to the derivatives of the objective function, we talk in those cases about *derivative-free* (or *zeroth-order*) methods. On the contrary, line search and trust region methods usually make use of first-order (gradients, ∇f) and second-order (Hessians, $\nabla^2 f$) information, and in this perspective we talk about *first-order methods* and *second-order methods*.

We will delve deep into the details of line search based methods later; in the meantime, however, we shall discuss the properties we would like the sequence $\{x^k\}$, and the



(a) Line search methods (b) Trust region methods (c) Pattern search methods

Figure 3: Visualization of key concepts of iterative optimization algorithms.

corresponding sequences $\{f(x^k)\}$ and $\{\nabla f(x^k)\}$, to possess, regardless of the type of the update rule.

In an ideal scenario, an algorithm would end up on a stationary point after a finite number of iterations; in other words, it might be the case that, for some $\bar{k}$, we get $\nabla f(x^{\bar{k}}) = 0$ and the procedure stops. When a method is guaranteed to behave in this way, we say that it possesses *finite convergence* properties. Unfortunately, this property is achieved by few algorithms on very specific classes of problems.

The sequences in general are indeed infinite. We are therefore interested in the convergence properties of these infinite sequences.

4.1.1 Existence of accumulation points

The first key property an algorithm shall be guaranteed to satisfy is that the sequence it produces, or at least a part of it, is "going somewhere". In other words, we would not want the sequence $\{x^k\}$ to diverge altogether, meaning $\|x^k\| \to \infty$. Indeed, we are interested in getting in the end a solution, with precise, finite values, to employ within some real-world system.

This requirement formally translates into the sequence having accumulation points. We recall that an accumulation point of a sequence is the limit point of some subsequence, i.e., $\bar{x}$ is an accumulation point for $\{x^k\}$ if there exists a subsequence $K \subseteq \{0, 1, \dots, \}$ such that $x^k \to \bar{x}$ for $k \in K$, $k \to \infty$.

Ensuring the existence of at least an accumulation point is algorithmically rather simple, with very mild assumptions on the problem at hand, as we state in the following proposition.

Proposition 4.1. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuous function. Let $x^0 \in \mathbb{R}^n$ and let the level set $\mathcal{L}_0 = \{x \in \mathbb{R}^n \mid f(x) \leq f(x^0)\}$ be compact. Assume the sequence $\{x^k\}$ starting at x^0 is such that, for all k, $f(x^{k+1}) \leq f(x^k)$. Then, the sequence $\{x^k\}$ admits accumulation points, each one belonging to $\mathcal{L}_0$, and the sequence $\{f(x^k)\}$ converges to some value $\bar{f}$.*

Proof. By the assumptions, $f(x^{k+1}) \leq f(x^k)$ for all k. Then, by induction, $f(x^k) \leq f(x^0)$ for all $k = 0, 1, \dots$, which means that the entire sequence $\{x^k\}$ is contained in the level set $\mathcal{L}_0$. From the compactness of $\mathcal{L}_0$, we have that $\{x^k\}$ has accumulation points, all belonging to $\mathcal{L}_0$.

Now, the sequence $\{f(x^k)\}$ is monotone decreasing, thus it has some limit $\bar{f}$. By the boundedness of $\{x^k\} \subseteq \mathcal{L}_0$ and the continuity of f, the value of $\bar{f}$ is finite. □

The compactness condition on the initial level set is certainly satisfied, for example, if the objective function is coercive. On the other hand, imposing the monotonicity of the objective values sequence is quite easy algorithmically: we will focus on this aspect in the following.

An interesting byproduct of the proposition is that the sequence of values of f converges altogether; as a consequence, we have $f(\bar{x}) = \bar{f}$ for any accumulation point $\bar{x}$ (i.e., all accumulation points are equivalent in terms of objective value). Actually, this property immediately holds every time we guarantee a monotonic behavior with a function bounded below.

Now, while the existence of accumulation points is somewhat an essential minimal requirement, we have no guarantee that any of these accumulation points is actually meaningful for the problem we are trying to solve. This issue is what we are going to focus on next.

4.1.2 Convergence to stationarity

Reasonably, we would like the convergent (sub)sequences discussed in the previous section to asymptotically reach the stationarity condition. While this aspect is conceptually straightforward, its formal characterization needs some care. In particular, the stationarity property can be reached in different ways, that are listed and described here below in decreasing order of strength:

- $\lim_{k\to\infty} x^k = \bar{x}$ with $\nabla f(\bar{x}) = 0$ - the entire sequence is converging to a limit point which is a stationary point;
- $\lim_{k\to\infty} \|\nabla f(x^k)\| = 0$ - the entire sequence of gradients is going to zero; by the continuity of the norm and of ∇f, all the accumulation points of the sequence $\{x^k\}$ are stationary points;
- $\liminf_{k\to\infty} \|\nabla f(x^k)\| = 0$ - the gradients go to zero at least along a subsequence; if $\{x^k\}$ has no diverging subsequence, than at least one accumulation point is stationary.

The meaning of the three above situations can be better grasped looking at the following example.

Example 4.1. Consider the function of one variable $f(x)$ with gradient (derivative) given by $\nabla f(x) = (x-1)(x-2)$. Then:

- the sequence of values of
$$\{x^k\} = \{0.9, 0.99, 0.999, 0.9999, 0.99999, \ldots\}$$
is converging to the unique limit $\bar{x} = 1$, which is a stationary point for f;
- the sequence of values of
$$\{x^k\} = \{0.9, 2.1, 0.99, 2.01, 0.999, 2.001, 0.9999, 2.0001, 0.99999, 2.00001, \ldots\}$$
corresponds to the sequence of gradients
$$\{\nabla f(x^k)\} = \{0.11, 0.11, 0.0101, 0.0101, 0.001001, 0.001001, 0.00010001, 0.00010001, 0.0000100001, 0.0000100001, \ldots\}$$
that clearly converges to zero; the two accumulation points of $\{x^k\}$, 1 and 2, are indeed both stationary points.
- the sequence of values of
$$\{x^k\} = \{0.9, 3.1, 0.99, 3.01, 0.999, 3.001, 0.9999, 3.0001, 0.99999, 3.00001, \ldots\}$$

corresponds to the sequence of gradients

$$\{\nabla f(x^k)\} = \{0.11, 2.31, 0.0101, 2.0301, 0.001001, 2.003001, 0.00010001, \\ 2.00030001, 0.0000100001, 2.0000300001, \ldots\};$$

the sequence $\{|\nabla f(x^k)|\}$ has thus two converging subsequences, one with limit 0 and the other with limit 2; the limit inferior is thus 0 and there exists a subsequence of $\{x^k\}$ (the one converging to 1) that goes to a stationary point.

In practice, ensuring the third condition (the weakest one) is enough for computational purposes: indeed, by the continuity of the gradient, we know that if $\liminf_{k\to\infty} \|\nabla f(x^k)\| = 0$, then, for any $\epsilon > 0$, there exists $\bar{k}$ sufficiently large such that $\|\nabla f(x^{\bar{k}})\| \leq \epsilon$. This is important, as we are guaranteed that, if we employ for our algorithm a stopping condition based on a threshold on the norm of the gradient, the algorithm will certainly stop in finite time and will concretely provide us with a solution with the desired level of accuracy.

Guaranteeing this kind of convergence for the sequence of solutions is, however, not trivial at all; for instance, the strict decrease of the objective function is not sufficient to guarantee convergence to stationary points, not even in the convex case, as the following example shows.

Example 4.2. Let us consider the problem

$$\min_{x\in\mathbb{R}} \; f(x) = \frac{1}{2}x^2,$$

where the objective function is continuous and strictly convex, attaining the minimum value $f^\star = 0$ at the unique global optimizer $x^\star = 0$.

Let us now consider the iterative process starting at $x^0 = 2$ and following the update rule

$$x^{k+1} = x^k - \alpha_k f'(x^k) = x^k - \alpha_k x^k,$$

with α_k defined as

$$\alpha_k = \frac{x^k - 1}{2x^k},$$

so that

$$x^{k+1} = x^k - \frac{x^k - 1}{2} = \frac{x^k + 1}{2}.$$

We shall now observe that, for any $x^k \in (1, 2]$, we would get

$$2 \geq \frac{2+1}{2} \geq \frac{x^k + 1}{2} > \frac{1+1}{2} = 1,$$

i.e., x^{k+1} also belongs to the interval $(1, 2]$, and also

$$x^{k+1} = \frac{x^k + 1}{2} < \frac{x^k + x^k}{2} = x^k,$$

i.e., $x^{k+1} < x^k$. Recalling that $x^0 = 2$, we get by induction that $x^{k+1} < x^k$ holds for the entire sequence $\{x^k\}$. Then, we can see that

$$f(x^{k+1}) = \frac{1}{2}\left(x^{k+1}\right)^2 < \frac{1}{2}\left(x^k\right)^2 = f(x^k).$$

We hence defined a strictly decreasing sequence on a strongly convex function, and yet the entire sequence is contained in the interval $(1, 2]$ and thus cannot converge to the unique stationary point 0 - it actually converges to 1.

We shall conclude this discussion pointing out an important distinction for the type of the convergence properties possibly associated with an algorithm:

- we say that the convergence properties of an algorithm are *global* if they hold regardless of the starting solution for the iterative process;
- we say that the convergence properties are *local* if they only hold when the starting point is sufficiently close to one of the desired solutions.

Of course, convergence properties of global type are much preferable. This is true, in particular, since we do not know in practice how large the convergence neighborhood is and where it might be located, when we only have local properties. Yet, local convergence results are sometimes of interest, as better properties might be proved for certain methods when they are near a solution.

4.1.3 Efficiency of optimization solvers

When studying optimization algorithms, the primary interest lies in studying properties of local and global convergence towards stationary points: it is fundamental to ensure that they effectively produce candidate optimal solutions. However, efficiency of the algorithms is clearly also very important, especially in the large scale scenario where computing times may be really long. Efficiency of optimization algorithms is usually studied in terms of two different concepts: *convergence rate* and *complexity*.

Convergence rate intuitively denotes how fast the sequence of objective values $\{f(x^k)\}$ (or equivalently the sequence of iterates $\{x^k\}$ or gradients $\{\|\nabla f(x^k)\|\}$), approaches the limit point. Formally, we have the following possible cases.

Definition 4.1. Let $\{f(x^k)\}$ be the sequence of objective values generated by an iterative algorithm, with $f(x^k) \to f^\star$. Then, we say that the rate of convergence is

- *sublinear* if
$$\lim_{k\to\infty} \frac{f(x^{k+1}) - f^\star}{f(x^k) - f^\star} = 1;$$
- *linear* if, for some $\rho \in (0, 1)$,
$$\lim_{k\to\infty} \frac{f(x^{k+1}) - f^\star}{f(x^k) - f^\star} = \rho;$$
- *superlinear* if
$$\lim_{k\to\infty} \frac{f(x^{k+1}) - f^\star}{f(x^k) - f^\star} = 0;$$

In order to better understand the above definitions, let us examine the quantity taken to the limit:
$$\frac{f(x^{k+1}) - f^\star}{f(x^k) - f^\star};$$
at the numerator, we have the remaining gap to be filled to reach the limit value after the update of iteration k has been carried out; at the denominator, we have the gap at the beginning of the iteration; the ratio thus tells us how small is the gap after the iteration compared to what it was at the beginning; the convergence rate measures how this ratio behaves in the limit.

Having a sublinear convergence rate is bad: the longer you run the algorithm, the less progress it makes (the gap reduction progressively becomes irrelevant). Linear rate is ok, as a fixed percentage of the residual distance to the final value is covered at each iteration; superlinear convergence is great: for large k, one iteration is enough to bring the error down by orders of magnitude.

Another very popular approach to measuring the efficiency of optimization algorithms is based on **complexity**. Of course, even if algorithms have asymptotic convergence

properties, in practice they are stopped after a finite time. Now, the interesting question is how many iterations (and possibly how many function and gradient evaluations) are required to reach a particular accuracy level ϵ.

Firstly, we shall recall the meaning of the $\mathcal{O}$ notation.

Definition 4.2. Given two functions ϕ and g, we say that $\phi(n) = \mathcal{O}(g(n))$ if there exists $c > 0$ and $\bar{n}$ such that $\phi(n) \leq c\, g(n)$ for all $n \geq \bar{n}$.

Focusing for simplicity on the number of iterations as cost metric, we can give the following definitions.

Definition 4.3. Let $\{x^k\}$ be the sequence of iterates generated by an iterative method, with $f(x^k) \to f^\star$. We say that the algorithm has an *iteration error* of $\mathcal{O}(h(k))$ if:

$$f(x^k) - f^\star = \mathcal{O}(h(k));$$

equivalently, given an accuracy level ϵ, the algorithm has an *iteration complexity* of $\mathcal{O}(q(\epsilon))$ if

$$\min\{k \mid f(x^k) - f^\star \leq \epsilon\} = \mathcal{O}(q(\epsilon)).$$

As an "error" measure, instead of the quantity $f(x^k) - f^\star$, we might be interested in using the quantity $\|\nabla f(x^k)\|$, i.e., the distance from stationarity, since this is often the quantity used to define stopping condition; this is particularly well suited when analyzing algorithms in the nonconvex setting.

The worst case bound on iteration error gives us a measure about the size of the error we should expect after a given number of iterations; on the other hand, iteration complexity bound gives us an estimate about how long we shall expect the algorithm to run in the worst case.

Now, since we often measure the total cost of an optimization algorithm by the number of function and gradient evaluations carried out throughout the process, complexity bounds with respect to these cost metrics will also be important.

As remarked in the definition, there is a correspondence between iteration error and iteration complexity; assume an algorithm has an iteration error of $\mathcal{O}(\frac{1}{k})$ and that we want to obtain an ϵ-accurate solution, i.e., $f(x^k) - f^\star \leq \epsilon$; in the worst case, we have $f(x^k) - f^\star \leq \frac{C}{k}$; therefore, if $\frac{C}{k} \leq \epsilon$ we will be guaranteed to have reached the desired accuracy even in the worst cases; in other words, the solutions will surely be acceptable for all $k \geq \frac{C}{\epsilon}$. We can thus conclude that the first iteration such that $f(x^k) - f^\star \leq \epsilon$ happens within the first $\frac{C}{\epsilon}$ ones, i.e., the iteration complexity is $\mathcal{O}(\frac{1}{\epsilon})$.

Thus, if the iteration error is $\mathcal{O}(\frac{1}{k})$, the iteration complexity is $\mathcal{O}(\frac{1}{\epsilon})$; similarly, if the iteration error is $\mathcal{O}(\frac{1}{k^2})$, the iteration complexity is $\mathcal{O}(\frac{1}{\sqrt{\epsilon}})$.

Note that $\mathcal{O}(\frac{1}{\epsilon})$ is not a good iteration complexity. We can indeed think of $\log(\frac{1}{\epsilon})$ as the "number of digits of accuracy" wanted. With an $\mathcal{O}(\frac{1}{\epsilon})$ complexity, if we need 10 iterations for a digit of accuracy, then we might need 100 iterations for 2 digits, 1000 iterations for 3 digits, and so on: cost is exponential.

This can be put in relation with convergence speed:

- An iteration complexity of $\mathcal{O}(\frac{1}{\epsilon})$ is equivalent to an error of $\mathcal{O}(\frac{1}{k})$, which can be substituted into the definition of convergence rate[1]:

$$\lim_{k\to\infty} \frac{f(x^{k+1}) - f^\star}{f(x^k) - f^\star} = \lim_{k\to\infty} \mathcal{O}\left(\frac{k}{k+1}\right) = 1,$$

  the rate is sublinear!

- An error of $\mathcal{O}(\rho^k)$ with $\rho < 1$ leads to linear convergence rate, with an iteration complexity of $\mathcal{O}(\log(\frac{1}{\epsilon}))$; the cost of adding a new digit of accuracy is "polynomial".

[1] warning: not rigorous math here!

- An error of the kind $\mathcal{O}(\rho^{2^k})$ with $\rho < 1$ leads to a superlinear convergence rate, with an iteration complexity of $\mathcal{O}(\log(\log(\frac{1}{\epsilon})))$; the cost of adding a new digit of accuracy is "constant"!

ϵ	$\mathcal{O}(\frac{1}{\epsilon^2})$	$\mathcal{O}(\frac{1}{\epsilon})$	$\mathcal{O}(\frac{1}{\sqrt{\epsilon}})$	$\mathcal{O}(\log(\frac{1}{\epsilon}))$	$\mathcal{O}(\log(\log(\frac{1}{\epsilon})))$
1	1	1	1	0	-
0.1	100	10	$\sqrt{10}$	1	0
0.01	10^4	100	10	2	$\log 2$
10^{-3}	10^6	10^3	$\sqrt{10^3}$	3	$\log 3$
10^{-10}	10^{20}	10^{10}	10^5	10	1

Table 1: Examples of complexity types. The values in the table should help visualizing trends; however recall that the bounds hold asymptotically, i.e., are more accurate for small values of ϵ.

k	$\mathcal{O}(\frac{1}{\sqrt{k}})$	$\mathcal{O}(\frac{1}{k})$	$\mathcal{O}(\frac{1}{k^2})$	$\mathcal{O}((\frac{1}{2})^k)$	$\mathcal{O}((\frac{1}{2})^{2^k}))$
1	1	1	1	0.5	0.25
2	0.7	0.5	0.25	0.25	0.06
3	0.57	0.33	0.11	0.12	0.004
4	0.5	0.25	0.06	0.06	10^{-5}
10	0.31	0.1	0.01	10^{-3}	10^{-308}
100	0.1	0.01	10^{-4}	10^{-5}	0
1000	0.03	10^{-3}	10^{-6}	10^{-300}	0

Table 2: Examples of iteration error types. The values in the table should help visualizing trends; however recall that the bounds hold asymptotically, i.e., are more accurate for large values of k.

4.2 Descent Methods based on line search

The principal class of algorithms we will be interested in is that of line-search based approaches. As already mentioned, iterative methods within this family are characterized by updates of the form

$$x^{k+1} = x^k + \alpha_k d_k, \tag{2}$$

where d_k is a search direction (in particular, a descent direction satisfying $\nabla f(x^k)^T d_k < 0$) and α_k is a positive scalar stepsize. For algorithms of this kind, a suitable choice for both defining elements - the direction and the stepsize - is crucial to enforce convergence properties.

Concerning the importance of the stepsize choice, consider once again Example 4.2; in that particular case, the sequence $\{x^k\}$ is indeed defined according to a rule of the form (2), where at each iteration the search direction, given by $-f'(x^k)$, is the only descent direction at x^k; the failure to converge has thus to be attributed to an erroneous choice of the stepsize.

On the other hand, we see in the following example a situation where convergence issues are clearly related to the choice of the direction.

Example 4.3. Let us consider the following optimization problem:

$$\min_{x,y,z} f(x,y,z) = \frac{1}{2}x^2 + \frac{1}{2}y^2 - \frac{1}{2}xy + z^2;$$

furthermore, assume we apply an algorithm of the form (2) to this problem, starting from the point $(x^0, y^0, z^0) = (1, 1, 1)$ and using search directions defined as followed:

$$d_k = \begin{cases} \left(-\nabla_x f(x^k, y^k, z^k), 0, 0\right)^T = (-x^k + \frac{1}{2}y^k, 0, 0)^T & \text{if } k \text{ is even,} \\ \left(0, -\nabla_y f(x^k, y^k, z^k), 0\right)^T = (0, -y^k + \frac{1}{2}x^k, 0)^T & \text{otherwise.} \end{cases}$$

We thus have at each iteration one of the following updates,

$$\begin{pmatrix} x^{k+1} \\ y^{k+1} \\ z^{k+1} \end{pmatrix} = \begin{pmatrix} x^k - \alpha_k x^k + \frac{\alpha_k}{2} y^k \\ y^k \\ z^k \end{pmatrix}, \qquad \begin{pmatrix} x^{k+1} \\ y^{k+1} \\ z^{k+1} \end{pmatrix} = \begin{pmatrix} x^k \\ y^k - \alpha_k y^k + \frac{\alpha_k}{2} x^k \\ z^k \end{pmatrix},$$

the former one being for even iterations and the latter one being for odd iterations.

As for the stepsize α_k, assume at each iteration we choose $\alpha_k = 1$; we can show that this is the optimal stepsize at each iteration. Let us consider any even iteration; looking at the objective as a function of the stepsize, we have

$$\phi(\alpha) = f(x^k - \alpha_k x^k + \frac{\alpha}{2} y^k, y^k, z^k) = \\ \frac{1}{2}\left(x^k + \alpha(-x^k + \frac{1}{2}y^k)\right)^2 + \frac{1}{2}(y_k)^2 - \frac{1}{2}y^k(x^k + \alpha(-x^k + \frac{1}{2}y^k) + (z_k)^2,$$

which is a convex function (the quadratic coefficient $(-x^k + \frac{1}{2}y^k)^2$ is certainly nonnegative). We can thus minimize it by setting to zero the derivative:

$$0 = \phi'(\alpha) = \left(x^k + \alpha\left(-x^k + \frac{1}{2}y^k\right)\right)\left(-x^k + \frac{1}{2}y^k\right) - \frac{1}{2}y^k\left(-x^k + \frac{1}{2}y^k\right) = \\ \left(-x^k + \frac{1}{2}y^k\right)\left(x^k - \frac{1}{2}y^k + \alpha\left(-x^k + \frac{1}{2}y^k\right)\right)$$

which is in fact solved by $\alpha = 1$. An analogous derivation can be done for the case of odd k. The update rules thus become

$$\begin{pmatrix} x^{k+1} \\ y^{k+1} \\ z^{k+1} \end{pmatrix} = \begin{pmatrix} \frac{1}{2}y^k \\ y^k \\ z^k \end{pmatrix}, \qquad \begin{pmatrix} x^{k+1} \\ y^{k+1} \\ z^{k+1} \end{pmatrix} = \begin{pmatrix} x^k \\ \frac{1}{2}x^k \\ z^k \end{pmatrix},$$

for the even and odd case respectively.

We shall also note that, for even iterations, direction d_k is surely a descent direction if $x^k \neq \frac{1}{2}y^k$:

$$\nabla f(x^k, y^k, z^k)^T d_k = \left(x^k - \frac{1}{2}y^k, y^k - \frac{1}{2}x^k, 2z^k\right)^T \left(-x^k + \frac{1}{2}y^k, 0, 0\right) = \\ -\left(x^k - \frac{1}{2}y^k\right)^2 < 0$$

Similarly, d_k is a descent direction for all odd iterations where $y^k \neq \frac{1}{2}x^k$.

Let us finally consider the sequence of solutions produced by the algorithm; we have

$$(x^1, y^1, z^1) = \left(\frac{y^0}{2}, y^0, z^0\right) = \left(\frac{1}{2}, 1, 1\right)$$
$$(x^2, y^2, z^2) = \left(x^1, \frac{x^1}{2}, z^1\right) = \left(\frac{1}{2}, \frac{1}{4}, 1\right)$$
$$(x^3, y^3, z^3) = \left(\frac{y^2}{2}, y^2, z^2\right) = \left(\frac{1}{8}, \frac{1}{4}, 1\right)$$
$$(x^4, y^4, z^4) = \left(x^3, \frac{x^3}{2}, z^3\right) = \left(\frac{1}{8}, \frac{1}{16}, 1\right)$$
$$\vdots$$

The sequence is constructed using a descent direction and the best possible stepsize at each iteration, and it is clearly $\{x^k, y^k, z^k\}$ is clearly converging to $(0, 0, 1)$. The limit point, however, is not a stationary point for the problem, as $\nabla f(0, 0, 1) = (0, 0, 2) \neq 0$.

At this point, it should be clear that we shall be very careful in the design of update rules of the form (2) for descent methods. In the following sections we will discuss this issue in detail.

4.2.1 Line Searches

Given a descent direction d_k, i.e., a direction that satisfies $\nabla f(x^k)^T d_k < 0$, we are interested in finding a suitable stepsize α_k along the direction to be used in an update rule of the form (2), so that the objective function decreases in a suitable way.

In principle, an idea might be that of choosing the best possible stepsize along the direction, that is, the one that leads to the minimum possible value of f. In this case, we would be carrying out an *exact line search* along d_k, which can be formally characterized as follows:

$$\alpha_k \in \arg\min_{\alpha>0} \varphi(\alpha) = f(x^k + \alpha d_k),$$

where the function φ of the scalar variable α characterizes the evolution of the objective function along the search direction. This operation is what we have done, for instance, in Example 4.3, and is depicted in Figure 4a

This strategy is viable, more generally, with strictly convex quadratic optimization problems of the form

$$\min_x \ f(x) = \frac{1}{2} x^T Q x + c^T x, \qquad Q \succ 0. \tag{3}$$

Applying second order Taylor's expansion to $f(x^k + \alpha d_k)$, which holds with the equality being f quadratic, recalling that $\nabla^2 f(x^k) = Q$, we get

$$\varphi(\alpha) = f(x^k + \alpha d_k) = f(x^k) + \alpha \nabla f(x^k)^T d_k + \frac{\alpha^2}{2} d_k^T Q d_k,$$

which is a convex parabola (the quadratic coefficient $d_k^T Q d_k$ is positive from the positive definiteness of Q). We can thus find the optimal stepsize setting to zero the derivative $\varphi'(\alpha)$:

$$0 = \varphi'(\alpha) = \nabla f(x^k)^T d_k + \alpha d_k^T Q d_k,$$

which is solved by

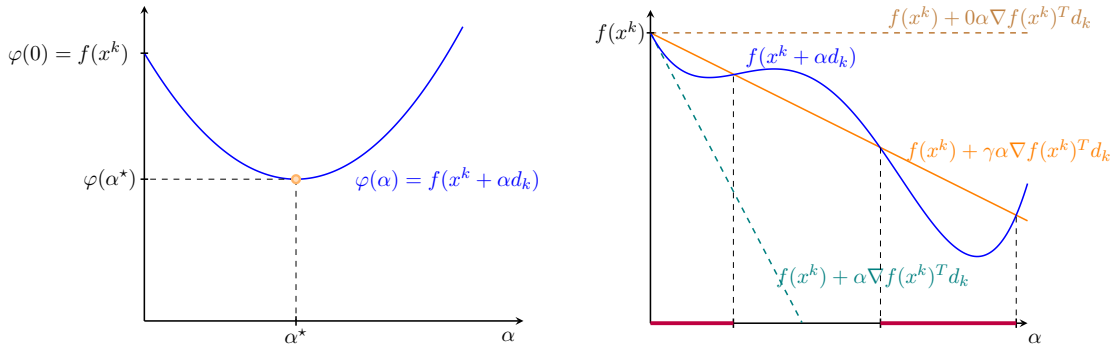
$$\alpha_k = -\frac{\nabla f(x^k)^T d_k}{d_k^T Q d_k}.$$

However, except for very special cases like this one, exact line searches have to be excluded because of two main reasons:

(a) For a general function f, the minimizer of to $\varphi(\alpha)$ might not be available in closed form and finding it might require to use some kind of solver, with a non-negligible computational cost; the operation might thus be not computationally convenient, also taking into account that d_k might be a not-so-good direction not worth exploring with high precision.

(b) In the nonconvex case, we are not even able to check whether a stepsize is globally optimal or not for the search direction.

The above considerations motivate the interest in *inexact line search techniques*. This class fo methods for selecting the stepsize have the goal of identifying a value α_k providing a decrease in the objective function, i.e., $f(x^k + \alpha_k d_k) < f(x^k)$. Actually, as we have seen



(a) Exact line search

(b) Armijo line search

Figure 4: Exact and inexact line search.

in Example 4.2, the simple decrease of the objective function is not enough to guarantee convergence of the overall descent algorithm. Hence, the line search approaches aim to identify a stepsize that actually provides a *sufficient decrease* of the objective function, i.e., a decrease which is significant given the status of the current solution.

The arguably most popular sufficient decrease condition is *Armijo condition*, which requires

$$f(x^k + \alpha_k d_k) \leq f(x^k) + \gamma\alpha_k \nabla f(x^k)^T d_k. \tag{4}$$

Basically, Armijo condition asks to select a stepsize so that the objective function decreases at least like a linear model with a softer slope than the tangent model. This is illustrated in Figure 4b.

Indeed, let us look at the quantities involved in th condition; the element on the left side is actually $\phi(\alpha)$, i.e., the objective function along the considered direction; the term $f(x^k) = \phi(0)$ is the current objective value; if the value of γ were 0, we would have on the right side the equation of a horizontal line and the condition would be a simple decrease condition $f(x^{k+1}) \leq f(x^k)$. The term $\nabla f(x^k)^T d_k$, if we look carefully, is the derivative of $\varphi(\alpha)$ at 0 - by the chain rule we have

$$\varphi'(\alpha) = \nabla f(x^k + \alpha d_k)^T d_k, \qquad \varphi'(0) = \nabla f(x^k)^T d_k.$$

Therefore, for $\gamma = 1$, the rightmost part in the Armijo condition is the equation of the tangent line to the graph of $\varphi(\alpha)$ at $\alpha = 0$. More in general, we can rewrite the condition as

$$\varphi(\alpha) \leq \varphi(0) + \gamma\varphi'(0)\alpha.$$

Note that, as long as we select a descent direction, the quantity $\nabla f(x^k)^T d_k < 0$, which implies that both the function $\varphi(\alpha)$ and the line $\varphi(0) + \gamma\alpha\nabla f(x^k)^T d_k$ certainly have a downward trend initially. For values of $\gamma \in (0, 1)$, we obtain lines passing through $(0, \varphi(0))$ with a milder slope than the tangent line. Hence, for a given value of $\gamma \in (0, 1)$, Armijo condition asks us to find a stepsize such that the true function $\varphi(\alpha)$ lies below the resulting line (see Figure 4b). Note that, thanks to d_k being a descent direction, we are guaranteed that, at least, the condition will be satisfied by sufficiently small stepsizes - in other words, the first interval of stepsizes starting at 0 will satisfy the condition.

Now we might want to shift our focus on how we can operatively identify a stepsize satisfying the sufficient decrease condition. The algorithm to do so is very intuitive and is based on the concept of backtracking: assume we have an initial guess for the stepsize; we can check whether this value satisfies the Armijo condition or not; if it does, we are done; otherwise, we can shrink this value by a predefined factor (do a backtrack step) to obtain a new guess; the process can then be repeated until a satisfactory stepsize is obtained.

We can characterize this procedure by a simple pseudocode (Algorithm 1) or by the following formula:

$$\alpha_k = \max_{j=0,1,\dots} \{\delta^j \Delta_0 \mid \varphi(\delta^j \Delta_0) \leq \varphi(0) + \gamma \delta^j \Delta_0 \varphi'(0)\}$$

where $\delta \in (0,1)$ and $\Delta_0 > 0$.

Algorithm 1: `Armijo Line Search`

Input: $x^k \in \mathbb{R}^n$, $d_k \in \mathbb{R}^n : \nabla f(x^k)^T d_k < 0$, $\Delta_0 > 0$, $\gamma \in (0,1)$, $\delta \in (0,1)$.
Set $\alpha = \Delta_0$
while $f(x^k + \alpha d_k) > f(x^k) + \gamma \alpha \nabla f(x^k)^T d_k$ **do**
    Set $\alpha = \delta \alpha$;
Set $\alpha_k = \alpha$
return α_k

This simple algorithmic scheme enjoys the following theoretical properties.

Proposition 4.2. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function, $x^k \in \mathbb{R}^n$, $d_k \in \mathbb{R}^n$ such that $\nabla f(x^k)^T d_k < 0$. Let $\gamma \in (0,1)$, $\delta \in (0,1)$ and $\Delta_0 > 0$. Then, the Armijo line search procedure (Algorithm 1) terminates in a finite number of iterations, providing a stepsize α_k satisfying condition* (4)*. Moreover, one of the two following conditions hold:*

(a) $\alpha_k = \Delta_0$;

(b) $\alpha_k \leq \delta \Delta_0$ and $f\left(x^k + \frac{\alpha_k}{\delta} d_k\right) > f(x^k) + \gamma \frac{\alpha_k}{\delta} \nabla f(x^k)^T d_k$.

Proof. We first prove the finite termination property. Let us assume by contradiction that the algorithm never stops and goes on for an infinite number of iterations. Then, the stopping condition is not satisfied by the stepsize $\delta^j \Delta_0$ for all $j = 0, 1, \dots$, i.e., we have

$$f(x^k + \delta^j \Delta_0 d_k) > f(x^k) + \gamma \delta^j \Delta_0 \nabla f(x^k)^T d_k.$$

Rearranging and dividing both sides by the quantity $\delta^j \Delta_0$, we get

$$\frac{f(x^k + \delta^j \Delta_0 d_k) - f(x^k)}{\delta^j \Delta_0} > \gamma \nabla f(x^k)^T d_k.$$

Since $\delta \in (0,1)$, if we take the limits for $j \to \infty$ we get that $\delta^j \Delta_0 \to 0$ and then, on the left side of the inequality, we obtain the directional derivative of f along d_k at x^k. We hence have

$$\nabla f(x^k)^T d_k = \lim_{j \to \infty} \frac{f(x^k + \delta^j \Delta_0 d_k) - f(x^k)}{\delta^j \Delta_0} \geq \lim_{j \to \infty} \gamma \nabla f(x^k)^T d_k = \gamma \nabla f(x^k)^T d_k.$$

Rearranging once again, we get

$$(1 - \gamma) \nabla f(x^k)^T d_k \geq 0.$$

But $(1 - \gamma)$ is a positive quantity, whereas $f(x^k)^T d_k$ is negative by assumption. We thus got a contradiction, completing the first part of the proof.

As for the further property, it immediately follows by the instructions of the algorithm: indeed, if the first trial step Δ_0 satisfies the Armijo condition, we get $\alpha_k = \Delta_0$; otherwise, α_k is obtained carrying out at least one backtrack step, and the second last tried step was $\frac{\alpha_k}{\delta}$ and it did not satisfy the Armijo condition. □

The rationale of the first property is that, by repeatedly cutting down the stepsize by the same fraction, in the worst case we will at some point end up in the initial interval of stepsizes satisfying the condition, so that the line search actually stops in finite time. The second property trivially tells us that, if we do not accept the first stepsize, we obtain a stepsize by reducing a larger stepsize that does not satisfy the Armijo condition.

4.2.2 Gradient related directions

As we have seen from example 4.3 and maybe contrarily to the intuition, choosing at each iteration a descent direction is not sufficient to ensure its converge to stationarity.

If we again refer to Example 4.3, what happens there is that we asymptotically approach a non-stationary point using directions that are less and less of descent, tending to be orthogonal to the gradient of the limit point.

In order to avoid this kind of behavior, we shall thus ask for the direction to be linked to the gradient vector in some way. We formally give the following definition.

Definition 4.4. Let $\{x^k, d_k\}$ be the sequence generated by an iterative algorithm of the form (2). We say that the sequence $\{d_k\}$ of directions is *gradient related* to $\{x^k\}$ if there exists two constants $c_1, c_2 > 0$ such that, for all $k = 0, 1, \ldots$, the following conditions hold:

$$\|d_k\| \leq c_1 \|\nabla f(x^k)\|, \qquad \nabla f(x^k)^T d_k \leq -c_2 \|\nabla f(x^k)\|^2.$$

Let us now interpret these two conditions. The former one tells us that the size of the search direction can even be very large, as long as it is not infinitely larger than the size of the gradient; at the same time, the direction has to shrink to zero when the gradient goes to zero. The second condition, on the other hand, instead asks us to select a descent direction whose directional derivative is not negligible when compared to the directional derivative along the gradient direction and shall go to zero only when we are close to a stationary point.

Now let us denote by $\theta(v_1, v_2)$ the angle between two vectors v_1 and v_2. Combining the two conditions from Definition 4.4, we get

$$\cos\left(\theta\left(d_k, -\nabla f(x^k)\right)\right) = \frac{-\nabla f(x^k)^T d_k}{\|d_k\|\|\nabla f(x^k)\|} \geq \frac{-\nabla f(x^k)^T d_k}{c_1 \|\nabla f(x^k)\|^2} \geq \frac{c_2 \|\nabla f(x^k)\|^2}{c_1 \|\nabla f(x^k)\|^2} = \frac{c_2}{c_1} > 0.$$

Forcing these two properties, therefore, we are guaranteeing that the angle between the search direction and the negative gradients remains less or equal than $90°$ and bounded away from zero, thus avoiding the problem of the two directions becoming asymptotically orthogonal like in Example 4.3.

The following question shall concern what classes of search directions are actually gradient related. Of course, the negative gradient direction $-\nabla f(x^k)$ satisfies the two conditions with $c_1 = c_2 = 1$.

The gradient-related property holds, for example, for directions of the form $d_k = -H_k \nabla f(x^k)$ where H_k is a symmetric matrix with suitable properties; we shall note that d_k defined in this way is certainly a direction of descent if H_k is positive definite; indeed:

$$\nabla f(x^k)^T d_k = -\nabla f(x^k) H_k \nabla f(x^k) < 0.$$

However, positive definiteness of the entire sequence of matrices $\{H_k\}$ is not enough to guarantee gradient relatedness. The property can be ensured if we guarantee that the sequence $\{H_k\}$ satisfies the *bounded eigenvalues condition*, i.e., if there exists constants c_1, c_2 with $0 < c_2 < c_1 < \infty$ such that, for all $k = 0, 1, \ldots$, we have

$$c_2 \leq \lambda_{\min}(H_k) < \lambda_{\max}(H_k) \leq c_1.$$

Indeed, under this assumption, we have for any k

$$\|d_k\| = \|H_k \nabla f(x^k)\| \leq \lambda_{\max}(H_k) \|\nabla f(x^k)\| \leq c_1 \|\nabla f(x^k)\|$$

and

$$\nabla f(x^k)^T d_k = -\nabla f(x^k)^T H_k \nabla f(x^k) \leq -\lambda_{\min}(H_k) \|\nabla f(x^k)\|^2 \leq c_2 \|\nabla f(x^k)\|^2.$$

We shall keep in mind this condition, as it will be particularly valuable when we will discuss some important classes of algorithms.

4.2.3 Convergence results

In this section we provide the general convergence results for line search based descent methods. We start with the global convergence result.

Proposition 4.3. *Let f be a continuously differentiable function and let $\{x^k, d_k\}$ be the sequence generated by an iterative algorithm of the form (2), assuming the stepsize α_k is obtained by the Armijo line search (Algorithm 1) for all k and that the sequence of directions $\{d_k\}$ is gradient-related to $\{x^k\}$. Moreover, assume the level set $\mathcal{L}_0 = \{x \in \mathbb{R}^n \mid f(x) \leq f(x^0)\}$ is compact. Then, the sequence $\{x^k\}$ admits accumulation points, and each accumulation point $\bar{x}$ is a stationary point, i.e., $\nabla f(\bar{x}) = 0$.*

Proof. For all $k = 0, 1, \ldots$, since the Armijo condition holds and recalling the gradient related condition, we have

$$f(x^{k+1}) = f(x^k + \alpha_k d_k) \leq f(x^k) + \gamma \alpha_k \nabla f(x^k)^T d_k \leq f(x^k) - \gamma c_2 \alpha_k \|\nabla f(x^k)\|^2 < f(x^k). \tag{5}$$

The sequence $\{f(x^k)\}$ is thus monotone decreasing and, by the compactness of $\mathcal{L}_0$ and recalling Proposition 4.1, the sequence $\{x^k\}$ thus admits limit points each one belonging to $\mathcal{L}_0$. Moreover, $\{f(x^k)\}$ converges to some finite value f^* and we have

$$\lim_{k \to \infty} f(x^{k+1}) - f(x^k) = 0 \tag{6}$$

Now, let $\bar{x}$ be any of these limit points, i.e., there exists $K \subseteq \{0, 1, \ldots\}$ such that

$$\lim_{\substack{k \in K \\ k \to \infty}} x^k = \bar{x}.$$

Let us assume by contradiction that $\bar{x}$ is not stationary, i.e., $\|\nabla f(\bar{x})\| = \nu$ for some $\nu > 0$. Rearranging equation (5), we can write

$$f(x^{k+1}) - f(x^k) \leq \gamma \alpha_k \nabla f(x^k)^T d_k \leq 0.$$

Taking the limits for $k \in K$, $k \to \infty$ and recalling (6), we get

$$0 \leq \lim_{\substack{k \in K \\ k \to \infty}} \gamma \alpha_k \nabla f(x^k)^T d_k \leq 0,$$

and thus

$$\lim_{\substack{k \in K \\ k \to \infty}} \alpha_k \nabla f(x^k)^T d_k = 0.$$

By the gradient-related condition, we have that $\nabla f(x^k)^T d_k \leq -c_2 \|\nabla f(x^k)\|^2$ and thus

$$0 = \lim_{\substack{k \in K \\ k \to \infty}} \alpha_k \nabla f(x^k)^T d_k \leq \lim_{\substack{k \in K \\ k \to \infty}} -c_2 \alpha_k \|\nabla f(x^k)\|^2,$$

which implies

$$\lim_{\substack{k \in K \\ k \to \infty}} \alpha_k \|\nabla f(x^k)\|^2 = 0.$$

By the continuity of $\nabla f(x)$ we have for $k \in K$, $k \to \infty$, that $\|\nabla f(x^k)\|^2 \to \|\nabla f(\bar{x})\|^2 = \nu^2 > 0$. Hence, we necessarily have $\lim_{\substack{k \in K \\ k \to \infty}} \alpha_k = 0$.

Now, $\alpha_k \to 0$ along subsequence K implies that there exists $\bar{k} \in K$ such that $\alpha_k < \Delta_0$ for all $k \in K$, $k \geq \bar{k}$. By Proposition 4.2, we then have

$$f\left(x^k + \frac{\alpha_k}{\delta} d_k\right) > f(x^k) + \gamma \frac{\alpha_k}{\delta} \nabla f(x^k)^T d_k \tag{7}$$

for all $k \in K$, $k \geq \bar{k}$. On the other hand, by the mean value theorem, we can write for all k

$$f\left(x^k + \frac{\alpha_k}{\delta} d_k\right) = f(x^k) + \frac{\alpha_k}{\delta} \nabla f\left(x^k + \theta_k \frac{\alpha_k}{\delta} d_k\right)^T d_k$$

with $\theta_k \in (0,1)$. Combining this equality with (7), we get for all $k \in K$

$$\frac{\alpha_k}{\delta} \nabla f\left(x^k + \theta_k \frac{\alpha_k}{\delta} d_k\right)^T d_k > \gamma \frac{\alpha_k}{\delta} \nabla f(x^k)^T d_k,$$

and thus

$$\nabla f\left(x^k + \theta_k \frac{\alpha_k}{\delta} d_k\right)^T d_k > \gamma \nabla f(x^k)^T d_k. \tag{8}$$

We shall also note that, by the continuity of $\nabla f(x)$, we have that the sequence $\{\nabla f(x^k)\}_K$ converges to $\nabla f(\bar{x})$ and is thus bounded, i.e., there exists M such that $\|\nabla f(x^k)\| \leq M$ for all $k \in K$. By the gradient-related condition, we thus have

$$\|d_k\| \leq c_1 \|\nabla f(x^k)\| \leq c_1 M \qquad \forall k \in K;$$

the sequence $\{d_k\}_K$ is thus bounded and there exists a further subsequence $K_2 \subseteq K$ such that $\lim_{\substack{k \in K \\ k \to \infty}} d_k = \bar{d}$.

Taking the limits in (8) for $k \in K_2$, $k \to \infty$, recalling that $\theta_k \in (0,1)$ for all k and that that $\alpha_k \to 0$, we finally get

$$\nabla f(\bar{x})^T \bar{d} \geq \gamma \nabla f(\bar{x})^T \bar{d},$$

and thus

$$(1-\gamma) \nabla f(\bar{x})^T \bar{d} \geq 0,$$

which implies $\nabla f(\bar{x})^T \bar{d} \geq 0$. However, from the gradient-related condition and the assumption about $\bar{x}$ being not stationary, we know that

$$\nabla f(\bar{x})^T \bar{d} = \lim_{\substack{k \in K_2 \\ k \to \infty}} \nabla f(x^k)^T d_k \leq \lim_{\substack{k \in K_2 \\ k \to \infty}} -c_2 \|\nabla f(x^k)\|^2 = -\nu^2 < 0.$$

We finally got a contradiction, completing the proof.

□

Next, we provide some further insights on the Armijo condition and backtracking line search algorithm, in the special case where it is used in conjunction with gradient related directions and the objective function has Lipschitz-continuous gradients. The first result identifies a full interval of sufficiently small stepsizes guaranteed to satisfy the Armijo condition at all iterations.

Proposition 4.4. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be an L-smooth function. Let $\{x^k\}$ be the sequence generated by an iterative algorithm of the form (2), assuming the sequence of search directions $\{d_k\}$ is gradient-related. Then, for all k, the Armijo condition (4) is satisfied for all $\alpha \in [0, \Delta_{low}]$, with $\Delta_{low} = \frac{2c_2(1-\gamma)}{c_1^2 L}$.*

Proof. Let us assume that a step α does not satisfy the Armijo condition, i.e.,

$$f(x^k + \alpha d_k) > f(x^k) + \gamma \alpha \nabla f(x^k)^T d_k.$$

By the L-smoothness of f, we also have (see Proposition 1.8) that

$$f(x^k + \alpha d_k) \leq f(x^k) + \alpha \nabla f(x^k)^T d_k + \frac{\alpha^2 L}{2} \|d_k\|^2.$$

Combining the two inequalities, we get

$$\gamma \alpha \nabla f(x^k)^T d_k < \alpha \nabla f(x^k)^T d_k + \frac{\alpha^2 L}{2} \|d_k\|^2,$$

i.e.,

$$\alpha(1-\gamma)\nabla f(x^k)^T d_k + \frac{\alpha^2 L}{2}\|d_k\|^2 > 0.$$

Bounding the quantities $\nabla f(x^k)^T d_k$ and $\|d_k\|$ according to the gradient-related assumption, we furthermore get

$$-c_2\alpha(1-\gamma)\|\nabla f(x^k)\|^2 + \frac{c_1^2\alpha^2 L}{2}\|\nabla f(x^k)\|^2 > 0.$$

Dividing by $\alpha\|\nabla f(x^k)\|^2$ and rearranging, we finally get

$$\alpha > \frac{2c_2(1-\gamma)}{c_1^2 L} = \Delta_{\text{low}},$$

which completes the proof. □

Next, we can exploit the above result to set a bound on the number of backtrack steps required at each iteration to get a suitable stepsize.

Proposition 4.5. *Let the same assumptions as in Proposition 4.4 hold. Further assume that, for all k, the stepsize α_k is obtained by the Armijo procedure using $\Delta_0 > 0$ as the initial tentative stepsize. Then, at each iteration k, the number of backtrack steps j_k (and thus of additional evaluations of f) is bounded above by*

$$j_k \leq j^* = \max\left\{0, \left\lceil \log_{1/\delta} \frac{\Delta_0}{\Delta_{low}} \right\rceil\right\},$$

where $\Delta_{low} = \frac{2c_2(1-\gamma)}{c_1^2 L}$, and the step size α_k is bounded by

$$\alpha_k \geq \min\{\Delta_0, \delta\Delta_{low}\}.$$

Proof. Let j^* be the smallest integer such that

$$\delta^{j^*}\Delta_0 \leq \Delta_{\text{low}}.$$

By Proposition 4.4, the step $\delta^{j^*}\Delta_0$ is thus guaranteed to satisfy the Armijo-type condition for any k. Therefore, by the Armijo line search, we have $j_k \leq j^*$. By the definition of j^* we also have that

$$\delta^{j^*} \leq \frac{\Delta_{\text{low}}}{\Delta_0},$$

i.e.,

$$j^* \geq \log_\delta \frac{\Delta_{\text{low}}}{\Delta_0} = \log_{1/\delta} \frac{\Delta_0}{\Delta_{\text{low}}}.$$

Actually, j^* is the smallest positive integer such that the above inequality holds, so

$$j^* = \max\left\{0, \left\lceil \log_{1/\delta} \frac{\Delta_0}{\Delta_{\text{low}}} \right\rceil\right\}.$$

Now, if $\Delta_0 < \Delta_{\text{low}}$, there is certainly no need of backtracking, $j_k = 0$ and $\alpha_k = \Delta_0$. Otherwise, if $\Delta_0 > \Delta_{\text{low}}$, we have

$$j_k \leq j^* = \left\lceil \log_{1/\delta} \frac{\Delta_0}{\Delta_{\text{low}}} \right\rceil \leq \log_{1/\delta} \frac{\Delta_0}{\Delta_{\text{low}}} + 1,$$

and then

$$\frac{1}{\delta^{j_k-1}} \leq \frac{\Delta_0}{\Delta_{\text{low}}}, \qquad \text{i.e.,} \qquad \delta^{j_k-1} \geq \frac{\Delta_{\text{low}}}{\Delta_0},$$

and finally $\delta\Delta_{\text{low}} \leq \delta^{j_k}\Delta_0 = \alpha_k$. The proof is thus complete. □

A very interesting consequence of the above results is that, if we knew the quantities L, c_1, and c_2, and thus Δ_{low}, we could directly set $\alpha_k = \Delta_{\text{low}}$ and be ensured that the Armijo condition would always be satisfied and we have the convergence result from Proposition 4.3 without the need of backtracking. The forthcoming complexity result would also be guaranteed.

This considerations also allow us to justify the fact that sometimes descent algorithm work with a constant stepsize; if we are lucky enough to set this constant value lower than Δ_{low}, we get global convergence. Of course, too small stepsizes slow down the practical efficiency of the algorithms, so using line searches is in practice much more convenient, as we are allowed to try more aggressive steps without losing convergence guarantees.

We are finally ready to give the last result of this section, concerning the worst case complexity bound in the nonconvex case.

Proposition 4.6. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be an L-smooth function. Let $\{x^k\}$ be the sequence generated by an iterative algorithm of the form (2), assuming the sequence of search directions $\{d_k\}$ is gradient-related and the stepsize α_k is obtained by the Armijo procedure using $\Delta_0 > \delta\Delta_{low}$ as the initial tentative stepsize[2]. Also assume that f is bounded below by some value f^*. Then, for any $\epsilon > 0$, at most k_{max} iterations are needed to produce an iterate x^k such that $\|\nabla f(x^k)\| \leq \epsilon$, where*

$$k_{max} \leq \frac{f(x^0) - f^*}{\gamma c_2 \delta \Delta_{low} \epsilon^2} = \mathcal{O}(\epsilon^{-2}).$$

The same worst case complexity bound of $\mathcal{O}(\epsilon^{-2})$ also holds for the number of function and gradient evaluations.

Proof. Since the Armijo condition is satisfied at each iteration and by the gradient-related condition, we have for all k

$$\begin{aligned} f(x^{k+1}) - f(x^k) &= f(x^k + \alpha_k d_k) - f(x^k) \\ &\leq \gamma \alpha_k \nabla f(x^k)^T d_k \\ &\leq -\gamma c_2 \alpha_k \|\nabla f(x^k)\|^2 \\ &\leq -\gamma c_2 \delta \Delta_{\text{low}} \|\nabla f(x^k)\|^2. \end{aligned}$$

where the last inequality follows from Proposition 4.5.

We now assume that for the first k_ϵ iterations we have $\|\nabla f(x^k)\| > \epsilon$. Recalling that f^* is the lower bound of f, we can then write

$$\begin{aligned} f^* - f(x^0) &\leq f(x^{k_\epsilon}) - f(x^0) \\ &= \sum_{k=0}^{k_\epsilon - 1} f(x^{k+1}) - f(x^k) \\ &\leq \sum_{k=0}^{k_\epsilon - 1} -\gamma c_2 \delta \Delta_{\text{low}} \|\nabla f(x^k)\|^2 \\ &= \sum_{k=0}^{k_\epsilon - 1} -\gamma c_2 \delta \Delta_{\text{low}} \epsilon^2 \\ &= -k_\epsilon \gamma c_2 \delta \Delta_{\text{low}} \epsilon^2. \end{aligned}$$

Rearranging, we finally obtain

$$k_\epsilon \leq \frac{f(x^0) - f^*}{\gamma c_2 \delta \Delta_{\text{low}} \epsilon^2}.$$

Since we only evaluate the gradient once per iteration and, recalling Proposition 4.5, the upper bound on the number of backtracks is constant through iterations, we have that the same worst case complexity bound also holds for function and gradient evaluations. □

[2] we make the assumption $\Delta_0 > \delta\Delta_{\text{low}}$ only for the sake of simplicity

We finally proved that descent methods based on Armijo type line searches and gradient-related directions have, on nonconvex L-smooth functions, a worst case iteration complexity bound of $\mathcal{O}(\frac{1}{\epsilon^2})$ (equivalently, $\mathcal{O}(\frac{1}{\sqrt{k}})$). This result is in fact nice, as the bound has been proven to be tight for first order methods.

Proposition 4.7. *No first-order algorithm can be designed with a better complexity bound than $\mathcal{O}(\frac{1}{\epsilon^2})$ to find and ϵ-approximate stationary point of an L-smooth function.*

It is instead possible to prove better complexity bounds under stronger assumptions, in particular if the objective function is convex or, even better, strongly convex. To obtain these results, however, proofs usually need to exploit the specific update rule definition and the mechanisms of the algorithm at hand; for this reason, these particular cases will be analyzed later.

4.3 Gradient Descent Method

At this point in the discussion, it is trivial to introduce the archetypal nonlinear optimization algorithm: the *gradient descent* method. This famous algorithm is simply a line search based method of the form (2), where:

- the search direction is given by the negative gradient (which is, of course, a gradient-related direction with $c_1 = c_2 = 1$).
- the stepsize α_k is selected by means of the Armijo rule.

The pseudocode of the resulting method is reported in Algorithm 2.

Algorithm 2: `Gradient Descent`

```
Input: x^0 ∈ R^n.
Set k = 0
while ‖∇f(x^k)‖ ≠ 0 do
    Set d_k = −∇f(x^k)
    Compute α_k along d_k by the Armijo line search (Algorithm 1)
    Set x^{k+1} = x^k + α_k d_k
    Set k = k + 1
```

Since the Gradient Descent algorithm meets all the criteria characterizing the class of methods analyzed in Sections 4.2.1-4.2.2, we can immediately invoke Propositions 4.3 and 4.6 to state global convergence properties in the nonconvex setting and the worst-case iteration complexity bound of $\mathcal{O}(\frac{1}{\epsilon^2})$ under L-smoothness - optimal for first order methods. We shall also note that, since we only carry out one gradient computation per iteration in Algorithm 2, the same complexity bound also holds for the number of gradient evaluations. Moreover, since by Proposition 4.5 we know that the number of backtrack steps at every iteration is bounded by a constant quantity, the iteration complexity result also translates into a $\mathcal{O}(\frac{1}{\epsilon^2})$ function evaluations worst case complexity bound.

We shall now analyze what happens, instead, in the convex case. To simplify our analysis, we will assume that a constant stepsize $\alpha_k = \frac{1}{L}$ is employed at every iteration: we know, by Proposition 4.4 and setting $\gamma = 0.5$, that this step is actually always a valid choice to satisfy Armijo condition and lead to global convergence results.

Proposition 4.8. *Let f be an L-smooth convex function and let $\{x^k\}$ be the sequence produced by the gradient descent algorithm with constant stepsize $\alpha_k = \frac{1}{L}$. Assume $f^\star$ is the optimal value of f and $x^\star$ is a minimizer of f, i.e., $f(x^\star) = f^\star$. Then,*

$$f(x^k) - f^\star \leq \frac{L\|x^0 - x^\star\|^2}{2k},$$

i.e., the algorithm has an iteration error of $\mathcal{O}(\frac{1}{k})$ and an iteration complexity of $\mathcal{O}(\frac{1}{\epsilon})$.

Proof. From the Lipschitz continuity of $\nabla f(x)$ and by Proposition 1.8, being $x^{k+1} = x^k + \alpha_k d_k$, we have that

$$\begin{aligned} f(x^{k+1}) &\leq f(x^k) + \alpha_k \nabla f(x^k)^T d_k + \frac{1}{2}\alpha_k^2 L \|d_k\|^2 \\ &= f(x^k) - \frac{1}{L}\nabla f(x^k)^T \nabla f(x^k) + \frac{L}{2L^2}\left\|-\nabla f(x^k)\right\|^2 \\ &= f(x^k) - \frac{1}{2L}\|\nabla f(x^k)\|^2, \end{aligned} \tag{9}$$

where we have exploited the fact that $\alpha_k = \frac{1}{L}$ and $d_k = -\nabla f(x^k)$.
On the other hand, by the convexity of f, we can write

$$f(x^\star) \geq f(x^k) + \nabla f(x^k)^T (x^\star - x^k)$$

and rearranging

$$f(x^k) \leq f(x^\star) + \nabla f(x^k)^T (x^k - x^\star).$$

Combining this last result with (9), we get

$$f(x^{k+1}) \leq f(x^\star) + \nabla f(x^k)^T (x^k - x^\star) - \frac{1}{2L}\|\nabla f(x^k)\|^2,$$

and hence

$$\begin{aligned} f(x^{k+1}) - f(x^\star) &\leq \frac{L}{2}\left(\frac{2}{L}\nabla f(x^k)^T (x^k - x^\star) - \frac{1}{L^2}\|\nabla f(x^k)\|^2\right) \\ &= \frac{L}{2}\left(\frac{2}{L}\nabla f(x^k)^T (x^k - x^\star) - \frac{1}{L^2}\|\nabla f(x^k)\|^2 - \|x^k - x^\star\|^2 + \|x^k - x^\star\|^2\right) \\ &= \frac{L}{2}\left(-\|x^k - x^\star - \frac{1}{L}\nabla f(x^k)\|^2 + \|x^k - x^\star\|^2\right) \\ &= \frac{L}{2}\left(-\|x^{k+1} - x^\star\|^2 + \|x^k - x^\star\|^2\right), \end{aligned}$$

where we have used $\|a\|^2 + \|b\|^2 - 2a^T b = \|a - b\|^2$. For each k, we thus have

$$f(x^{k+1}) - f(x^\star) \leq \frac{L}{2}(\|x^k - x^\star\|^2 - \|x^{k+1} - x^\star\|^2).$$

Therefore

$$\begin{aligned} \sum_{t=0}^{k} f(x^{t+1}) - f(x^\star) &\leq \sum_{t=0}^{k} \frac{L}{2}(\|x^t - x^\star\|^2 - \|x^{t+1} - x^\star\|^2) \\ &= \frac{L}{2}(\|x^0 - x^\star\|^2 - \|x^{k+1} - x^\star\|^2) \\ &\leq \frac{L}{2}\|x^0 - x^\star\|^2. \end{aligned}$$

Recalling that $\{f(x^k)\}$ is decreasing, we have $f(x^{t+1}) \geq f(x^{k+1})$ for all $t = 0, \ldots, k$, hence

$$\sum_{t=0}^{k} f(x^{t+1}) - f(x^\star) \geq (k+1)(f(x^{k+1}) - f(x^\star)).$$

Putting everything together, we get

$$(k+1)(f(x^{k+1}) - f(x^\star)) \leq \frac{L}{2}\|x^0 - x^\star\|^2,$$

i.e.,

$$f(x^{k+1}) - f(x^\star) \leq \frac{L}{2(k+1)}\|x^0 - x^\star\|^2.$$

□

We have thus obtained that, in the convex case, the convergence rate of gradient descent remains sublinear, but the complexity bound improves from $\mathcal{O}(\frac{1}{\epsilon^2})$ to $\mathcal{O}(\frac{1}{\epsilon})$. This rate is not optimal for first-order methods in the convex case, we will discuss this aspect later.

Finally, we have a result for the strongly convex case.

Proposition 4.9. *Let f be an L-smooth strongly convex function. For any $x^0 \in \mathbb{R}^n$, the entire sequence $\{x^k\}$ produced by the gradient descent method with $\alpha_k = \frac{1}{L}$ for all k converges to the unique global minimizer x^* of f with a linear convergence rate.*

We can thus see that there is a significant gap of performance when we turn to strongly convex problems, as here the type of convergence rate is totally superior.

4.4 Gradient Methods with Momentum

The simple gradient descent method is not particularly efficient in theory and is not efficient at all in practice. For this reason, we start focusing on approaches that try to improve the speed of gradient descent method. We begin this journey focusing on first order descent methods that exploit information from the preceding iteration to determine the search direction and the stepsize at the current one. We will hence refer to *gradient methods with momentum*, i.e., to algorithms defined by an iteration of the generic form

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k) + \beta_k (x_k - x_{k-1}), \tag{10}$$

where $\alpha_k > 0$ is the stepsize, and $\beta_k > 0$ is the momentum weight. The idea of the above update is that the direction of last iteration will arguably be a good search direction even at the current one. Partially repeating the previous step has the effect of controlling oscillation and providing acceleration in low curvature regions. All of this can, in principle, be achieved only exploiting already available information: no additional function evaluations are required to be carried out. This feature makes the addition of momentum terms appealing in large-scale settings.

The best-known and most important gradient methods with momentum arguably are:

- *Heavy-ball* method;
- *Conjugate gradient* methods.

4.4.1 Heavy-ball method

The heavy-ball algorithm is directly described by an iteration of the form (10), i.e., by the following two step update rule:

$$\begin{cases} y^k = x^k - \alpha_k \nabla f(x^k), \\ x^{k+1} = y^k + \beta_k (x^k - x^{k-1}), \end{cases}$$

where α_k and β_k typically are fixed positive; in principle, suitable constants should be chosen depending on properties of the objective function (e.g., using Lipschitz constant of the gradient or the constant of strong convexity). In practice, however, this information is often not accessible and thus α_k can be chosen by a line search while the momentum parameter β_k is usually blindly set to some (more or less) reasonable value. Local convergence results for the heavy-ball method have been proven in the convex case. Moreover, in the case of strictly convex quadratic functions, the optimal values for the parameters α_k and β_k can be computed in closed form. The choice of this optimal parameters allows to obtain a local linear convergence rate with better constants than standard gradient descent. This result can be can be generalized under hypotheses of twice continuous differentiability, Lipschitz-continuous gradient and strong convexity.

However, the heavy-ball method was shown to be potentially unable to converge even under very strong regularity assumptions, and the convergence of the method in the non-convex case is still an open problem (note that the structure of the algorithm does not follow the framework from Section 4.2.2).

The heavy-ball method is often explained by a (not so accurate) physical analogy: the vector of variables can be seen as a particle, moving in the Euclidean space, which naturally tends to preserve its current trajectory, but at the same time is deflected by the acceleration (the gradient) produced by some force. In a similar vision, the momentum update can be equivalently defined by the following pair of equations:

$$\begin{aligned} v^{k+1} &= \beta_k v^k - \alpha_k \nabla f(x^k), \\ x^{k+1} &= x^k + v^{k+1}. \end{aligned}$$

The update vector v^k can be interpreted as a speed term, which is computed as an exponentially decaying mean of past negative gradients; the gradients are thus used to modify the speed, rather than the position of the particle. The movement is then performed according to the speed computed at the current positions.

Nesterov Accelerated Gradient

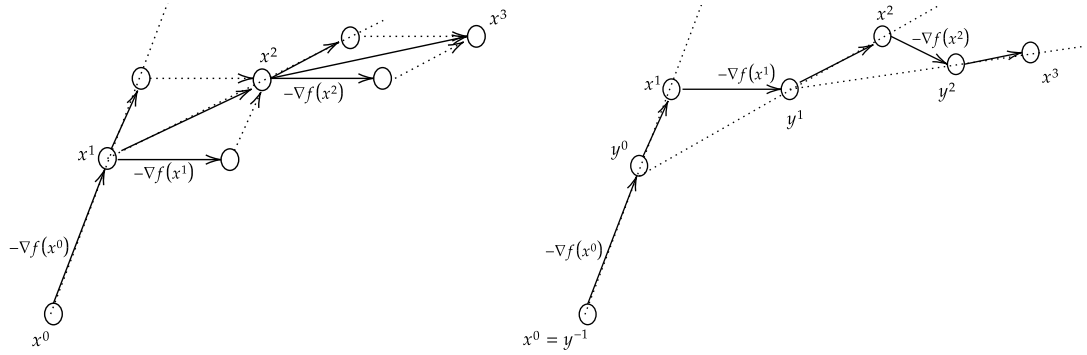
Nestereov Accelerated Gradient (NAG) method can be seen as a variant of heavy-ball. The update rule of Nesterov algorithm is given by the following pair of equation:

$$\begin{aligned} v^{k+1} &= \beta_k v^k - \alpha_k \nabla f(x^k + \beta_k v^k), \\ x^{k+1} &= x^k + v^{k+1}. \end{aligned}$$

As we can see, the only difference w.r.t. heavy-ball lies in the fact that gradient is computed at the point that would be obtained if the last move was repeated, rather than at the current point; iterations are essentially made of two steps: first a pure momentum step (with no gradient influence) and then a pure gradient descent step. The update rule can indeed be equivalently rewritten as:

$$\begin{cases} y^k = x^k - \alpha_k \nabla f(x^k) \\ x^{k+1} = y^k + \beta_k (y^k - y^{k-1}) \end{cases} \tag{11}$$

The stepsize α_k can be computed by an Armijo line-search, whereas β_k follows a suitable schedule. A visual comparison of the two algorithms can bee observed in Figure 5.



(a) Heavy-Ball (b) Nesterov AG

Figure 5: Visualization of the iterations of Momentum and Accelerated gradient algorithms.

The following proposition, concerning NAG method is very relevant in convex optimization.

Proposition 4.10. *Let f be an L-smooth convex function and let $\{x^k\}$ be the sequence produced by the Nesterov accelerated gradient algorithm[3]. Assume $f^\star$ is the optimal value of f. Then,*

$$f(x^k) - f^\star = \mathcal{O}\left(\frac{1}{k^2}\right),$$

i.e., the algorithm has an iteration error of $\mathcal{O}(\frac{1}{k^2})$ and an iteration complexity of $\mathcal{O}(\frac{1}{\sqrt{\epsilon}})$. These complexity bounds are tight for first order methods on convex functions.

It has been proved that the the accelerated gradient algorithm achieves an iteration complexity of $\mathcal{O}(\frac{1}{\sqrt{\epsilon}})$, i.e., $\mathcal{O}(\frac{1}{k^2})$; this result is significant, since this complexity bound has also been shown to be optimal for first-order methods: using only gradients information there is no way of doing better. Note that, however, such a complexity still leads to a sublinear asymptotic convergence rate.

As for the strongly convex case, the accelerated gradient method achieves the same iteration complexity and convergence rate as gradient descent, $\mathcal{O}(\rho^k)$, but with a smaller value of ρ, i.e., it has a faster linear convergence rate.

4.4.2 Conjugate gradient methods

Conjugate gradient methods can be described by the updates

$$x_{k+1} = x_k + \alpha_k d_k, \qquad d_{k+1} = -\nabla f(x_{k+1}) + \beta_{k+1} d_k, \tag{12}$$

where α_k is computed by means of a line search, whereas β_k is obtained according to one of the following rules:

$$\beta_{k+1} = \frac{\|\nabla f(x^{k+1})\|^2}{\|\nabla f(x^k)\|^2} \qquad \text{(Fletcher-Reeves)}$$

$$\beta_{k+1} = \frac{\nabla f(x^{k+1})^T(\nabla f(x^{k+1}) - \nabla f(x^k))}{\|\nabla f(x^k)\|^2} \qquad \text{(Polyak-Polak-Ribiére)}$$

$$\beta_{k+1} = \frac{\nabla f(x^{k+1})^T(\nabla f(x^{k+1}) - \nabla f(x^k))}{d_k^T(\nabla f(x^{k+1}) - \nabla f(x^k))} \qquad \text{(Hestenes-Stiefeld)}$$

$$\beta_{k+1} = \frac{\|\nabla f(x^{k+1})\|^2}{-d_k^T \nabla f(x^k)} \qquad \text{(Fletcher)}$$

$$\beta_{k+1} = \frac{\nabla f(x^{k+1})^T(\nabla f(x^{k+1}) - \nabla f(x^k))}{-d_k^T \nabla f(x^k)} \qquad \text{(Liu-Storey)}$$

$$\beta_{k+1} = \frac{\|\nabla f(x^{k+1})\|^2}{d_k^T(\nabla f(x^{k+1}) - \nabla f(x^k))} \qquad \text{(Dai-Yuan)}$$

We shall note that all the above rules only require to do some simple computations based on quantities that would be computed anyway even in the basic gradient descent method. Thus, the additional cost for the construction of direction d_{k+1} is really negligible in practice.

The update of conjugate gradient methods can be rewritten as

$$\begin{aligned} x_{k+1} &= x_k + \alpha_k(-\nabla f(x_k) + \beta_k d_{k-1}) \\ &= x_k + \alpha_k\left(-\nabla f(x_k) + \frac{\beta_k}{\alpha_{k-1}}(x_k - x_{k-1})\right) \\ &= x_k - \alpha_k \nabla f(x_k) + \hat{\beta}_k(x_k - x_{k-1}), \end{aligned}$$

so that it can be viewed as a gradient method with momentum according to definition (10).

[3] α_k chosen by Armijo line search, β_k a suitable predefined sequence

Yet, differently than the heavy-ball method, since the update rule has the form $x^{k+1} = x^k + \alpha_k d_k$, where α_k is computed by a line search along direction d_k, we might see this algorithm as an instance of the class described in Section 4.2.2 and we might expect to immediately derive some convergence result. Unfortunately, this is not the case. Indeed, we have no guarantee in general that direction d_k, obtained according to (12), is a descent direction, let alone a gradient-related direction.

In order to recover convergence guarantees, a simple strategy consist of the introduction, within the algorithmic scheme, of a safeguard: given constants c_1 and c_2, if direction d_k satisfies the gradient-related conditions, then the conjugate gradient iteration (with Armijo line search) can be carried out. Otherwise, we "restart" the algorithm, switching to a pure gradient descent iteration. This way, we would immediately get the convergence and complexity results from Proposition 4.3 and 4.6.

A different path to ensure d_k is a descent direction and to obtain convergence guarantees consists in the employment of a stronger line search than the Armijo one: *Wolfe line search*.

To ensure that the direction $d_{k+1} = -\nabla f(x^{k+1}) + \beta_{k+1}\nabla f(x^k)$ will be a descent direction, the condition

$$d_{k+1}^T \nabla f(x^{k+1}) = -\|\nabla f(x^{k+1})\|^2 + \beta_{k+1} d_k^T \nabla f(x^{k+1}) < 0$$

must be guaranteed. Note that both $\nabla f(x^{k+1}) = \nabla f(x^k + \alpha_k d_k)$ and (considering for example the Fletcher-Reeves method) $\beta_{k+1} = \frac{\|\nabla f(x^k+\alpha_k d_k)\|^2}{\|g_k\|^2}$ are indeed functions of α_k and can be controlled by a line search.

The Wolfe line search aims at finding a stepsize that satisfies the *weak Wolfe conditions*

$$f(x^k + \alpha_k d_k) \leq f(x^k) + \gamma\alpha_k \nabla f(x^k)^T d_k, \qquad \nabla f(x^k + \alpha_k d_k)^T d_k \geq \sigma f(x^k)^T d_k$$

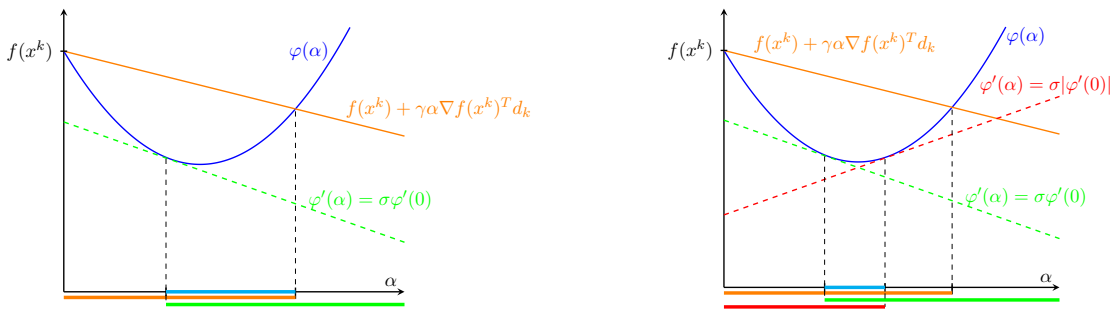
or the *strong Wolfe conditions*

$$f(x^k + \alpha_k d_k) \leq f(x^k) + \gamma\alpha_k \nabla f(x^k)^T d_k, \qquad \nabla f(x^k + \alpha_k d_k)^T d_k \leq |\sigma f(x^k)^T d_k|$$

where $\gamma \in (0, 1)$ and $\sigma \in (\gamma, 1)$.

Both conditions require the Armijo condition to hold, together with an additional property related to the directional derivative of the current direction at the new solution: at the next iteration, we want the current direction to be "much less" of descent than it was at this iteration; intuitively, we want to exploit the current direction to the bone, so that we will not be required to take a second descent step along a substantially unaltered direction. Somehow, we are restricting the interval of stepsizes to values that are approximately stationary for $\varphi(\alpha)$.

The graphical representation of the conditions is given in Figure 6.



(a) Weak Wolfe conditions

(b) Strong Wolfe conditions

Figure 6: Wolfe conditions.

A stepsize satisfying the weak or the strong Wolfe conditions can be found by a strategy similar to that of Armijo (except the trial stepsizes are allowed to increase if needed).

Computationally, the employment of this type of line search is not for free, as checking the Wolfe condition requires to compute not only the objective function, but also the gradient, for each trial stepsize.

For instance, if we employ the Wolfe line search within the Fletcher-Reeves conjugate gradient method, we are guaranteed that the next search direction will be of descent, i.e., $\nabla f(x^k + \alpha_k d_k)^T d_{k+1} < 0$, and the overall algorithm enjoys global convergence properties.

This class of methods is known to work pretty well both in the convex and the nonconvex case (even if complexity results are not available). The empirical reduction of iterations largely compensates in most cases the increased number of gradient evaluations, as long as the parameters of the line search are set properly and the trial stepsize is frequently accepted straightaway by the line search.

The quadratic case The *conjugate gradient method* appears to be particularly efficient at solving quadratic optimization problems with strictly convex objective functions (and, equivalently, linear systems of equations). In fact, the name of the method comes from this particular case, and the nonlinear algorithm can be seen as a generalization of the method for quadratic problems. Actually, all the rules for defining β_k reported for the nonlinear case collapse to the same quantity in this particular scenario, so that here we do talk about the congjugate gradient method (singular, without the s!).

The name conjugate gradient comes from a property of directions in quadratic problems of the form (3), with Q being a symmetric positive definite matrix.

Definition 4.5. Directions $d_0, d_1, \ldots, d_{m-1} \in \mathbb{R}^n$ are *mutually conjugate* w.r.t. a symmetric positive definite matrix Q if $d_i^T Q d_j = 0 \, \forall \, i, j = 0, \ldots, m-1$, $i \neq j$.

Being mutually conjugate can be proved to be a stronger condition than being linearly independent; from an optimization perspective, we would like to have a set of n conjugate directions w.r.t. Q, as the following proposition holds.

Proposition 4.11 (Finite Convergence of the Conjugate Directions Method in the Quadratic Case)**.** *Let $d_0, \ldots, d_{n-1}$ be a set of mutually conjugate directions w.r.t. Q. Let $x^0 \in \mathbb{R}^n$ and $x^{k+1} = x^k + \alpha_k d_k$, with $\alpha_k = \frac{-\nabla f(x^k)^T d_k}{d_k^T Q d_k}$. Then, there exists $m \leq n-1$ such that $x^{m+1} = x^\star$ such that $Qx^\star + c = 0$,i.e., $x^\star$ is the global minimizer.*

Note that the above property is very strong: if we have a quadratic problem and we do exact line search steps along mutually conjugate directions, we exactly get the global minimizer in finite time!

Now, the algorithmic issue with the method of conjugate directions is how to compute the set of directions. The *conjugate gradient* method comes here into play; let $g_k = \nabla f(x^k)$; let $d_0 = -g_0$ and then:

$$\alpha_k = \frac{-g_k^T d_k}{d_k^T Q d_k} = \frac{\|g_k\|^2}{d_k^T Q d_k}, \qquad \beta_{k+1} = \frac{\|g_{k+1}\|^2}{\|g_k\|^2}, \qquad d_{k+1} = -g_{k+1} + \beta_{k+1} d_k;$$

it can be shown that, at each iteration, the new direction d_{k+1} is mutually conjugate w.r.t. $d_0, \ldots, d_k$.

The scheme of the conjugate gradient method for quadratic problems is reported in Algorithm 3. Clearly, recalling Proposition 4.11, the method is exact, but in high dimensional cases a stopping criterion $\|g_k\| \leq \varepsilon$ can be introduced, making it an iterative method, to reduce the computational time. It is often observed that highly accurate solutions can be reached within very few iterations.

4.5 Newton's Method

We have seen that, with gradient descent and first-order algorithms in general, there are limits to what we can hope to achieve in terms of speed of convergence in the nonconvex,

Algorithm 3: `Conjugate Gradient Method for Quadratic Problems`

Input: $x^0 \in \mathbb{R}^n$, $d_0 = -g_0$.
$k = 0$;
while $\|g_k\| \neq 0$ **do**
  $\alpha_k = \frac{\|g_k\|^2}{d_k^T Q d_k}$;
  $x^{k+1} = x^k + \alpha_k d_k$;
  $g_{k+1} = g_k + \alpha_k Q d_k$;
  $\beta_{k+1} = \frac{\|g_{k+1}\|^2}{\|g_k\|^2}$;
  $d_{k+1} = -g_{k+1} + \beta_{k+1} d_k$;
  $k = k + 1$
Output: x^k

convex and strongly convex case. We are thus tempted to exploit, when the objective function allows it, second order information and see if we can obtain something better. Therefore, we assume here that $f \in \mathcal{C}^2(\mathbb{R}^n)$ and we now use the information provided by $\nabla^2 f(x)$ to construct a second-order method.

Let us assume we are at some solution x^k and also let $\nabla^2 f(x^k)$ be a positive definite matrix. We can then build Taylor's expansion of f to the second order obtaining

$$m_k(x) = f(x^k) + \nabla f(x^k)^T (x - x^k) + \frac{1}{2}(x - x^k)^T \nabla^2 f(x^k)(x - x^k).$$

The obtained approximation of the objective function is a quadratic strictly convex function (as the Hessian matrix is positive definite), we can thus find the global minimizer of m_k in closed form setting to zero its gradient, i.e.,

$$0 = \nabla m_k(x) = \nabla f(x^k) + \nabla^2 f(x^k)(x - x^k).$$

Rearranging, we obtain

$$x = x^k - [\nabla^2 f(x^k)]^{-1} \nabla f(x^k).$$

The idea of *Newton's method*, the prototypical second-order algorithm, is precisely that of defining an iterative scheme where the new iterate is obtained taking the minimizer of the quadratic model built around the current solution, i.e., using the update formula

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

This kind of update rule can be interpreted, to some extent, as the update rule of a descent method of the form (2), where $d_k = [\nabla^2 f(x^k)]^{-1} \nabla f(x^k)$ and α_k is constantly set to 1 throughout the entire process.

Now, we observe in the following example what can happen by adopting this kind of scheme on a simple univariate problem.

Example 4.4. Consider the problem

$$\min_{x \in \mathbb{R}} f(x) = \sqrt{1 + x^2}.$$

The problem has a quite regular objective function: it is strictly convex, coercive, twice continuously differentiable with

$$f'(x) = \frac{x}{\sqrt{1 + x^2}}, \qquad f''(x) = \frac{1}{(1 + x^2)^{3/2}}.$$

The iterations of the Newton's method applied to this problem take the form:

$$\begin{aligned} x^{k+1} &= x^k - \frac{f'(x^k)}{f''(x^k)} = x^k - \frac{x^k}{(1 + (x^k)^2)^{1/2}} (1 + (x^k)^2)^{3/2} \\ &= x^k - x^k (1 + (x^k)^2) = -(x^k)^3. \end{aligned}$$

If we take $x^0 = 1$ or $x^0 = -1$, iterates bounce back and forth from 1 to -1, defining a sequence with two accumulation points, 1 and -1, neither being stationary; if on the other hand we take x^0 such that $|x^0| > 1$, it is easy to observe that $x^k = (-1)^k (x^0)^{3^k}$ is a divergent sequence.

On the other hand, if we choose $|x^0| < 1$, for example $x^0 = \frac{1}{2}$, we get the following sequence of solutions:

$$x^0 = 0.5, \quad x^1 = -\frac{1}{2^3} = -0.125, \quad x^2 = \frac{1}{2^9} \approx 0.00195, \quad x^3 = -\frac{1}{2^{27}} \approx -0.000000007, \ \ldots$$

which we can see converges to 0, the unique minimizer, pretty fast: the distance from the optimum at each iteration is orders of magnitude smaller than it was at the preceding iteration - it seems a superlinear rate.

Hence, we can observe that Newton's method does not globally converge to the minimum point, as convergence depends on the starting solution, but it appears to be fast when it does converge.

The example basically tells us all the story about Newtons' method in its purest form: we are not sure it will converge, but if it does, it will be very fast. In formal terms, the algorithm does not possess global convergence guarantees (which is somewhat delusional, since it's worse than first order methods), but has fast local convergence guarantees.

We now fully characterize these properties in the following proposition.

Proposition 4.12. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a twice continuously differentiable function. Assume $x^\star \in \mathbb{R}^n$ be a point such that:*

- $\nabla f(x^\star) = 0$*;*
- $\nabla^2 f(x^\star)$ *is nonsingular.*

Then, there exists $\epsilon > 0$ such that, for all $x^0 \in \mathcal{B}(x^\star, \epsilon)$, the sequence $\{x^k\}$ generated by updates of the form $x^{k+1} = x^k - [\nabla^2 f(x^k)]^{-1} \nabla f(x^k)$ is well defined and satisfies:

i. $\{x^k\} \subseteq \mathcal{B}(x^\star, \epsilon)$*;*

ii. $\lim_{k\to\infty} x^k = x^\star$*;*

iii. the rate of convergence is superlinear.

Proof. Since the Hessian is continuous and nonsingular at $x^\star$, there exist $\epsilon_1 > 0$ and $\mu > 0$ such that, for all $x \in \mathcal{B}(x^\star, \epsilon_1)$, $\nabla^2 f(x)$ is invertible and[4]

$$\|(\nabla^2 f(x))^{-1}\| \leq \mu.$$

Moreover, for any $\sigma \in (0, 1)$, by the continuity of $\nabla^2 f$, it is always possible to find $\epsilon \leq \epsilon_1$ such that

$$\|\nabla^2 f(x) - \nabla^2 f(y)\| \leq \frac{\sigma}{\mu} \quad \forall\, x, y \in \mathcal{B}(x^\star, \epsilon). \tag{13}$$

Now, let us proceed by induction; we know by the assumptions that $x^0 \in \mathcal{B}(x^\star, \epsilon)$; let us now assume that, for an arbitrary k, $x^k \in \mathcal{B}(x^\star, \epsilon)$, i.e., $\|x^k - x^\star\| \leq \epsilon$. By Newton's update rule, we have

$$\begin{aligned}
x^{k+1} - x^\star &= x^k - [\nabla^2 f(x^k)]^{-1} \nabla f(x^k) - x^\star \\
&= -[\nabla^2 f(x^k)]^{-1} \nabla f(x^k) + [\nabla^2 f(x^k)]^{-1} \nabla^2 f(x^k)(x^k - x^\star) \\
&= -[\nabla^2 f(x^k)]^{-1} \left(\nabla f(x^k) - \nabla^2 f(x^k)(x^k - x^\star) \right) \\
&= -[\nabla^2 f(x^k)]^{-1} \left(\nabla f(x^k) - \nabla f(x^\star) - \nabla^2 f(x^k)(x^k - x^\star) \right)
\end{aligned}$$

[4] by the inverse function theroem, the inverse $(\nabla^2 f(x))^{-1}$ is continuous in a neighborhood of $x^\star$; then, by Weierstrass, the continuous function $\|(\nabla^2 f(x))^{-1}\|$ is bounded in that neighborhood

where we have used $[\nabla^2 f(x^k)]^{-1}\nabla^2 f(x^k) = I$ and then $\nabla f(x^\star) = 0$. We therefore have

$$\begin{aligned}\|x^{k+1} - x^\star\| &= \left\|[\nabla^2 f(x^k)]^{-1}\left(\nabla f(x^k) - \nabla f(x^\star) - \nabla^2 f(x^k)(x^k - x^\star)\right)\right\| \\ &\leq \left\|[\nabla^2 f(x^k)]^{-1}\right\| \left\|\nabla f(x^k) - \nabla f(x^\star) - \nabla^2 f(x^k)(x^k - x^\star)\right\| \\ &\leq \mu \left\|\nabla f(x^k) - \nabla f(x^\star) - \nabla^2 f(x^k)(x^k - x^\star)\right\|.\end{aligned}$$

Now, we can apply the mean value theorem for integrals (Proposition 1.7) to set

$$\nabla f(x^k) - \nabla f(x^\star) = \int_0^1 \nabla^2 f(x^* + t(x^k - x^\star))(x^k - x^\star)dt$$

and get

$$\begin{aligned}\|x^{k+1} - x^\star\| &\leq \mu \left\|\int_0^1 \nabla^2 f(x^* + t(x^k - x^\star))(x^k - x^\star)dt - \nabla^2 f(x^k)(x^k - x^\star)\right\| \\ &= \mu \left\|\int_0^1 (\nabla^2 f(x^* + t(x^k - x^\star)) - \nabla^2 f(x^k))(x^k - x^\star)dt\right\| \\ &\leq \mu \int_0^1 \left\|(\nabla^2 f(x^* + t(x^k - x^\star)) - \nabla^2 f(x^k))(x^k - x^\star)\right\| dt \\ &\leq \mu \int_0^1 \left\|\nabla^2 f(x^* + t(x^k - x^\star)) - \nabla^2 f(x^k)\right\| \left\|x^k - x^\star\right\| dt.\end{aligned} \tag{14}$$

We shall now note that $\mathcal{B}(x^\star, \epsilon)$ is a convex set and that both $x^\star$ and x^k belong to $\mathcal{B}(x^\star, \epsilon)$. Therefore, the point $x(t) = x^\star + t(x^k - x^\star) = (1-t)x^\star + tx^k$ also belongs to $\mathcal{B}(x^\star, \epsilon)$ for all $t \in [0, 1]$, i.e., for all values in the integral interval. Hence, recalling (13), we have $\|\nabla^2 f(x(t)) - \nabla^2 f(x^k)\| \leq \frac{\sigma}{\mu}$ for all $t \in [0, 1]$. We can thus write

$$\begin{aligned}\|x^{k+1} - x^\star\| &\leq \mu \int_0^1 \frac{\sigma}{\mu} \left\|x^k - x^\star\right\| dt \\ &= \sigma\|x^k - x^\star\| \\ &\leq \|x^k - x^\star\| \leq \epsilon,\end{aligned}$$

where the last two inequalities follow from $\sigma \in (0, 1)$ and $x^k \in \mathcal{B}(x^\star, \epsilon)$.

We have thus shown that $x^{k+1} \in \mathcal{B}(x^\star, \epsilon)$ if $x^k \in \mathcal{B}(x^\star, \epsilon)$. We can therefore conclude by induction that the whole sequence $\{x^k\} \subseteq \mathcal{B}(x^\star, \epsilon)$.

Now, iterating the first inequality in (4.5), we see that

$$0 \leq \|x^{k+1} - x^\star\| \leq \sigma\|x^k - x^\star\| \leq \sigma^2\|x^{k-1} - x^\star\| \leq \ldots \leq \sigma^{k+1}\|x^0 - x^\star\|.$$

Taking the limits for $k \to \infty$ and recalling again that $\sigma \in (0, 1)$, we get that

$$0 \leq \lim_{k\to\infty} \|x^k - x^\star\| \leq \lim_{k\to\infty} \sigma^k\|x^0 - x^\star\| = 0,$$

i.e.,

$$\lim_{k\to\infty} x^k = x^\star.$$

Moreover, we can go back to (14) and divide both sides of the inequality by $\|x^k - x^\star\|$, getting

$$0 \leq \frac{\|x^{k+1} - x^\star\|}{\|x^k - x^\star\|} \leq \mu \int_0^1 \left\|\nabla^2 f(x^\star + t(x^k - x^\star)) - \nabla^2 f(x^k)\right\| dt.$$

Taking the limits for $k \to \infty$, recalling that $x^k \to x^\star$, we finally get

$$\begin{aligned} 0 \leq \lim_{k\to\infty} \frac{\|x^{k+1} - x^\star\|}{\|x^k - x^\star\|} &\leq \lim_{k\to\infty} \mu \int_0^1 \left\|\nabla^2 f(x^\star + t(x^k - x^\star)) - \nabla^2 f(x^k)\right\| dt \\ &= \mu \int_0^1 \|\nabla^2 f(x^\star + tx^\star - tx^\star) - \nabla^2 f(x^\star)\| dt \\ &= \mu \int_0^1 \|\nabla^2 f(x^\star) - \nabla^2 f(x^\star)\| = 0, \end{aligned}$$

i.e., the convergence rate is superlinear. □

We have thus proved that pure Newton's method enjoys local superlinear convergence guarantees. This result of course is encouraging towards the employment of second order information in nonlinear optimization algorithms, but we shall take into account that the computation of the Hessian matrix is an expensive operation (we can roughly consider it to be n times more costly than computing ∇f and n^2 times more costly than f); moreover, to compute $d_k = -[\nabla^2 f(x^k)]^{-1}\nabla f(x^k)$, either we have to invert the $n \times n$ matrix and then carry out a matrix-vector product, or we shall solve the linear system (cheaper and more numerically robust); either way, the cost of this operation is $\mathcal{O}(n^3)$. Thus, while the local convergence rate in terms of iterations is very good, the significantly increased cost is hidden within this result. Thus, it is not always the case that the reduction in the number of iterations is mirrored by a decrease in the total cost of the algorithm.

Moreover, we have to deal with some issues:

- while presenting the Newton's method, we systematically made assumptions on the positive-definiteness of $\nabla^2 f(x^k)$, but in practice it will not always be the case:
  - if the objective function is convex but not strongly, we might end up at a point x^k such that the Hessian is not invertible and the algorithm is then not well-defined;
  - with general nonconvex functions, the Hessian not only might be not invertible as in the point above, but it also might even have negative eigenvalues and the direction might be of ascent.
- the absence of global convergence guarantees is a relevant weakness of the method.

All the above problems can be overcome by a suitable *globalization strategy*; in practice, we might want to:

(a) look at the Hessian's eigenvalues; if some eigenvalues are outside a predefined interval $[c_2, c_1]$ with $0 < c_2 < c_1 < \infty$, then we shall modify the Newton matrix so that the bounded eigenvalues condition holds;

(b) carry out an Armijo type line search along the obtained direction, with an initial stepsize of 1.

With these modifications to the procedure, we end up with a descent method using the Armijo line search along a gradient-related direction, thus getting global convergence guarantees.

Interestingly, it can be shown under reasonable assumptions that when the globalized Newton's method enters a neighborhood of a stationary point satisfying the assumptions of Proposition 4.12, there is no more need of neither modifying the Hessian or doing backtracks: the pure Newton's method is carried out from that point onward, thus recovering the fast local convergence result. When a method has this kind of behavior, we call it a *globally convergent Newton's method*.

4.6 Quasi-Newton Methods

We have seen that Newton's method possesses very interesting fast convergence properties, but we have also highlighted how the per-iteration cost might also become very high as the number of variables grows, up to the point where the advantage in terms of iterations is fully overwhelmed by weight of the required operations and the algorithm becomes basically unemployable. It is thus natural to ask ourselves if it would be possible to come up with something analogous to Newton's method but avoiding the computation of the Hessian matrix, only using first order information, and without the need of inverting an $n \times n$ matrix (or solving an analogous linear system).

Quasi-Newton methods have been proposed precisely following this idea: the Hessian matrix can be replaced by an approximation built using first-order information and knowledge from previous iterations. The general form of Quasi-Newton methods thus follows the general update rule

$$x^{k+1} = x^k + \alpha_k d_k \quad \text{with} \quad d_k = -B_k^{-1} d_k,$$

where B_k approximates the true hessian matrix at x^k. Recall that a direction defined in this way will certainly be a descent direction if B_k is positive definite.

Now, the key issue here is how to compute B_k and what properties this approximate matrix shall possess. To answer this question, we take inspiration once more from the quadratic case. For a quadratic function $f(x) = \frac{1}{2}x^T Q x + c^T x$, with $Q \succ 0$, we have for any $x, z \in \mathbb{R}^n$

$$\nabla f(x) - \nabla f(z) = Qx + c - Qz - c = Q(x - z).$$

If we plug two consecutive iterates x^k and x^{k+1} produced by some algorithm, and we define $y_k = \nabla f(x^{k+1}) - \nabla f(x^k)$ and $s_k = x^{k+1} - x^k$, we get

$$y_k = \nabla f(x^{k+1}) - \nabla f(x^k) = Q(x^{k+1} - x^k) = \nabla^2 f(x^{k+1}) s_k.$$

The former equality, which holds for the true Hessian in the quadratic case, is called *Quasi-Newton equation*. Equivalently, we could write

$$[\nabla^2 f(x^{k+1})]^{-1} y_k = s_k.$$

We can thus think of translating this property to the general nonlinear case, where we shall employ an approximation of the Hessian matrix that satisfies the Quasi-Newton equation, i.e., it satisfies a property typical of the true Hessian for quadratic objectives.

General schemes for Quasi-Newton approaches can be described with updates either in *direct* or *inverse form*:

$$x^{k+1} = x^k - \alpha_k B_k^{-1} \nabla f(x^k), \qquad B_k \approx \nabla^2 f(x^k) \qquad B_{k+1} s_k = y_k \qquad \text{(direct update)}$$

$$x^{k+1} = x^k - \alpha_k H_k \nabla f(x^k), \qquad H_k \approx [\nabla^2 f(x^k)]^{-1}, \qquad H_{k+1} y_k = s_k \qquad \text{(inverse update)}$$

Inverse updates are often more convenient from a computational point of view: since we approximate the inverse of the Hessian, rather than the Hessian itself, we can do not have the computational burden of inverting a large matrix or solving a large linear system to compute the search direction.

Clearly, we have yet to see how we can obtain approximate matrices that actually satisfy the Quasi-Newton property. The idea is, given the current approximation B_k (or H_k), to update it to get B_{k+1} (H_{k+1}); in other words, we update the matrices according to rules of the form

$$B_{k+1} = B_k + \Delta B_k, \qquad H_{k+1} = H_k + \Delta H_k.$$

The generic algorithmic scheme of a QN method is shown in Algorithm 4; for simplicity, we only report the case with inverse type updates - the case with direct updates is analogous.

Algorithm 4: `Quasi-Newton Method with Inverse Updates`

Input: $x^0 \in \mathbb{R}^n$, $H_0 \in \mathcal{S}_n, H_0 \succ 0$.
$k = 0$;
while $\|g_k\| \neq 0$ **do**
  set $d_k = -H_k \nabla f(x^k)$
  compute α_k along d_k by some line search
  $x^{k+1} = x^k + \alpha_k d_k$
  compute ΔH_k such that $(H_k + \Delta H_k)(\nabla f(x^{k+1}) - \nabla f(x^k)) = x^{k+1} - x^k$
  compute $H_{k+1} = H_k + \Delta H_k$
  $k = k + 1$

Algorithm 4 is a general scheme; realizations of this scheme are characterized by two elements: the type of line search employed and the specific update rule chosen to define ΔH_k. We immediately realize that if we employ an Armijo line search and we add a safeguard to guarantee H_k satisfies the bounded eigenvalues condition (similarly as what we discussed in the case of Newton's method), we immediately retrieve the global convergence and $\mathcal{O}(\frac{1}{\epsilon^2})$ complexity bounds results seen in Section 4.2.3.

Update rules for the Hessian approximations are usually based on rank-1 and rank-2 perturbations of the current matrix. In particular:

- **rank-1 updates** take the form

$$\Delta B_k = \rho_k u_k v_k^T \qquad \text{or} \qquad \Delta H_k = \rho_k u_k v_k^T,$$

  where $\rho_k \in \mathbb{R}_+$ and $u_k, v_k \in \mathbb{R}^n$; the update matrix obtained by this rule is indeed a rank-1 matrix; the effectiveness of the update depends on the specific choice of ρ_k, u_k, v_k. A popular rank-1 update is Broyden's rule: $B_{k+1} = B_k + \frac{(y_k - B_k s_k) s_k^T}{s_k^T s_k}$. The disadvantage of rank-1 updates is that B_{k+1} might lose symmetry, if suitable corrections are not adopted, and positive definiteness cannot guaranteed regardless.

- **rank-2 updates** take the form

$$\Delta B_k = a_k u_k u_k^T + b_k v_k v_k^T \qquad \text{or} \qquad \Delta H_k = a_k u_k u_k^T + b_k v_k v_k^T,$$

  with $a_k, b_k \in \mathbb{R}_+$ and $u_k, v_k \in \mathbb{R}^n$. Again, the effectiveness depends on the specific choice of the scalars and the vectors defining the update. This kind of updates is more stable than rank-1, as they allow to provably maintain the symmetry and positive definiteness properties of the matrices.

4.6.1 BFGS

The BFGS algorithm is a Quasi-Newton method that operates a rank-2 update of the approximation of the inverse of the Hessian matrix.

The BFGS update of the Quasi-Newton matrix, in the direct form, is given by

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k},$$

whereas in the inverse case it is

$$H_{k+1} = H_k + \left(1 + \frac{y_k^T H_k y_k}{s_k^T y_k}\right) \frac{s_k s_k^T}{s_k^T y_k} - \frac{s_k y_k^T H_k + H_k y_k s_k^T}{s_k^T y_k}.$$

Note that these update rules actually satisfy the Quasi-newton property; indeed:

$$\begin{aligned} B_{k+1}s_k &= B_k s_k + \frac{y_k y_k^T}{y_k^T s_k} s_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} s_k \\ &= B_k s_k + y_k \frac{y_k^T s_k}{y_k^T s_k} - B_k s_k \frac{s_k^T B_k s_k}{s_k^T B_k s_k} \\ &= B_k s_k + y_k - B_k s_k = y_k, \end{aligned}$$

and similar calculations can be done for the inverse update rule.

We also see how the BFGS update rule can enforce the preservation of positive definiteness.

Proposition 4.13. *Let H_k be positive definite and let H_{k+1} computed by the BFGS rule. Then H_{k+1} is positive definite if and only if $s_k^T y_k > 0$.*

The above proposition ensures that, if H_0 is positive definite and at each iteration $s_k^T y_k > 0$, the positive definiteness condition will continue to hold at each step. The condition $s_k^T y_k > 0$ can be imposed by properly setting the stepsize α_k; in fact, we have

$$s_k^T y_k = (\nabla f(x^{k+1}) - \nabla f(x^k))^T (x^{k+1} - x^k) > 0 \iff (\nabla f(x^{k+1}) - \nabla f(x^k))^T d_k > 0,$$

or in other words

$$\nabla f(x^{k+1})^T d_k > \nabla f(x^k)^T d_k,$$

which can be obtained using a Wolfe-type line search to determine α_k in Algorithm 4.

For the pure BFGS algorithm, with no safeguards, results exist of global convergence under convexity hypotheses on the objective function f; moreover, if strong convexity and twice differentiability are assumed, the method can be proved to superlinearly converge to the global minimizer, always accepting the unit stepsize for large k. Therefore, the BFGS algorithm is able to match the fast local convergence rate of Newton's method without employing second-order information.

4.6.2 L-BFGS

The L-BFGS algorithm is nothing but a BFGS modification specifically designed for the large scale setting (the L stands for Limited memory), but which actually shows strong performance in general settings and is widely considered the preferred choice for solving unconstrained nonlinear optimization problems.

The basic idea of the method is that, with high dimensional problems, storing the matrix H_k may be computationally prohibitive. In addition, the cost of computing the product $d_k = -H_k \nabla f(x^k)$ becomes unsustainable.

The matrix H_k can be obtained by a recursive rule requiring the only knowledge of vectors y_k and s_k:

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T$$

where

$$\rho_k = \frac{1}{y_k^T s_k}, \qquad V_k = I - \rho_k y_k s_k^T.$$

The first key cog in L-BFGS consists in approximating H_{k+1} by stopping the recursive formula after m steps: only the last m pairs (y_k, s_k) are required together with a starting approximation of H_{k-m}. The approximation usually employed is $H_{k-m} \approx \gamma I$, which can be easily stored in memory. Experimental experience suggests that relatively small values of m (2 up to 20) are enough to obtain satisfactory approximations.

This formula can then be exploited to reduce the computation of $H_k \nabla f(x^k)$. The so-called HG recursive procedure allows to compute $d_k = -H_k \nabla f(x^k)$ by only performing $4mn$ multiplications, with the sole knowledge of (y_{k-i}, s_{k-i}), $i = 0, \ldots, m$, and the approximation of H_{k-m}.

As already discussed, the L-BFGS algorithm is really efficient in practice at solving unconstrained optimization problems. Unfortunately, the experimental strength is not coupled with strong theoretical properties: global convergence results to stationary points are not known, even under convexity assumptions - convergence can be guaranteed only by means of the usual safeguarding strategies. Yet, it is possible to prove linear convergence in the strongly convex case under suitable hypotheses on the sequence $\{H_k\}$.

As a final remark, we point out that an extension of the L-BFGS method, called L-BFGS-B, has been designed to address bound constrained optimization problems.

4.7 Decomposition Techniques

There are various situations where optimization problems are very complex and difficult to handle as a whole. Yet, in some of these cases, it is possible to effectively tackle the problem by splitting it into smaller, simpler parts. Cases where *decomposition techniques* are helpful include situations where one or more of the following aspects occur:

- the number n of variables is large ($\sim 10^4$) and the information on variables or the objective cannot entirely be stored in memory;
- fixing the value of some variables, we obtain a subproblem that can be split into independent parts be solved with parallel computation techniques;
- fixing the value of some variables, the remaining subproblem is easy to solve or possesses superior regularity properties than the original one.

Example 4.5. Here below, we list some example problems where decomposition strategies might be helpful:

- $f(x) = g(x_1) + \sum_{i=2}^{n} h_i(x_1)s_i(x_i)$

  The above function becomes simple to handle if the value of variable x_1 is fixed, as in that case we are left with $n-1$ *separable* terms, each one only depending on a single variable x_i, that can be minimized in parallel. We can thus alternate between a univariate optimization step in x_1 and a parallel optimization of all other variables.

- $f(x, y) = \|\Phi(y)x - b\|^2$

  The above *nonlinear least squares* problem becomes a simple linear least squares problem if the value of variables y is fixed.

- $f(x, y) = x^2y^2 + 10xy + x^2 + y^2$

  The above function is nonconvex; however, it is a convex parabola w.r.t. x for any value of y, and the same holds w.r.t. y for any value of x. Thus, minimizing w.r.t. only one of the two variables amounts to an easier convex optimization problem that can be solved up to global optimality.

Let thus now assume that we have a generic problem of the form

$$\min_{x \in X} f(x) \tag{15}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $X \subseteq \mathbb{R}^n$; the problem can be equivalently rewritten in a decomposed fashion as

$$\min_{x_1 \in X_1, \ldots, x_m \in X_m} f(x_1, \ldots, x_m) \tag{16}$$

where $f : \mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_m} \to \mathbb{R}$, $X_1 \subseteq \mathbb{R}^{n_1}, \ldots, X_m \subseteq \mathbb{R}^{n_m}$ and $\sum_{i=1}^{m} n_i = n$. In other words, the vector of variables has been split into separate *blocks* of variables that might be handled independently.

There are plenty of techniques to identify a way of splitting variables into blocks and to exploit this subdivision algorithmically. These algorithms can be grouped into two main classes: *sequential methods* and *parallel methods*. In both cases, subproblems are defined with respect to a single block of variables, keeping the values of variables in all other blocks fixed.

Sequential Decomposition Methods

Sequential methods solve subproblems one at a time and variables are updated at each step (ref. Figure 7).

$$\underset{f=3}{(x^k, y^k, z^k)} \longrightarrow \underset{f=2.3}{(x^{k+1}, y^k, z^k)} \longrightarrow \underset{f=2}{(x^{k+1}, y^{k+1}, z^k)} \longrightarrow \underset{f=1.6}{(x^{k+1}, z^{k+1}, z^{k+1})}$$

Figure 7: Parallel decomposition schemes

The simplest example of sequential method is *Gauss-Seidel* algorithm. Gauss-Seidel algorithm is defined by the following update rule:

$$x_i^{k+1} = \arg\min_{\xi_i \in X_i} f(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, \xi_i, x_{i+1}^k, \ldots, x_m^k).$$

The algorithm cyclically solves the subproblems, obtained fixing the value of all variables except those in a block. The solution of the subproblem is immediately used to update value of current block's variables.

Note that the algorithm requires to compute a global optimizer for each subproblem: this is a strong requirement, that can be satisfied only if the subproblem are simple enough.

Global convergence to stationary points can be guaranteed for the Gauss-Seidel method under compactness assumptions on the level set and, in addition, one of the following conditions:

- f is convex;
- f is block-wise strictly convex, i.e., the objective of all the subproblems is always strictly convex;
- $m = 2$ - with only two blocks of variables convexity of the objective is not necessary.

Parallel Decomposition Methods

Parallel methods solve independently and simultaneously all the subproblems generated by each different block. Then, the best one among these solutions is used for the variables update, i.e., at each iteration only one block of variables is updated (ref. Figure 8).

An example of parallel method is *Jacobi* algorithm, defined by the following update rule:

$$\hat{x}_i^{k+1} \in \arg\min_{x_i} f(x_1^k, \ldots, x_i, \ldots, x_m^k), \qquad x^{k+1} \in \underset{(x_1^k, \ldots, \hat{x}_i^{k+1}, \ldots, x_m^k)}{\arg\min} f(x_1^k, \ldots, \hat{x}_i^{k+1}, \ldots, x_m^k).$$

4.7.1 Decomposition Methods with Blocks Overlapping

Up to this point, we considered decomposition approaches where $x \in \mathbb{R}^n$ is partitioned into m blocks that do not vary with the iterations: $x^k = (x_1^k, \ldots, x_m^k)$.

We now present a more flexible scheme: we introduce the notion of *working set* $W_k \subset \{1, \ldots, n\}$. At each iteration we consider the subproblem

$$\min_{x_{W_k}} f(x_{W_k}, x_{\overline{W}^k}^k) \tag{17}$$

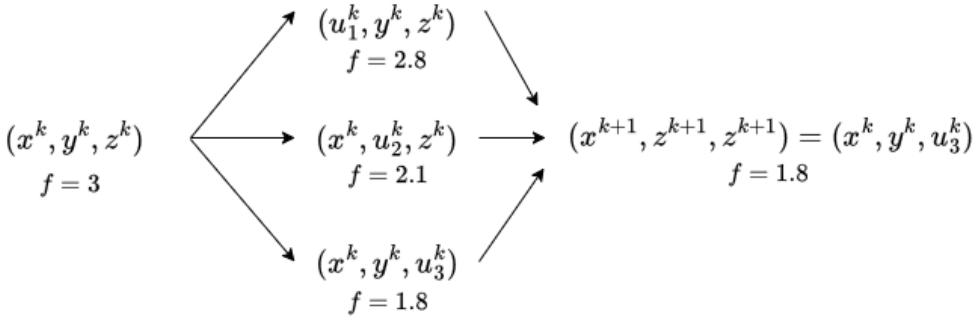


Figure 8: Parallel decomposition schemes

where $W^K \subset \{1, \ldots, n\}$, $\overline{W}^K = \{1, \ldots, n\} \setminus W_k$ and $x^k = (x^k_{W_k}, x^k_{\overline{W}^k})$. Thus, if $x^\star_{W_k}$ is the solution of problem (17) at the k-th iteration, we have the following update rule:

$$x_i^{k+1} = \begin{cases} x_i^\star & \text{if } i \in W_k, \\ x_i^k & \text{otherwise.} \end{cases} \tag{18}$$

Global convergence of this scheme depends on the *working set selection rule*, i.e., on how we choose the variables at each iteration. Possible selection rules ensuring global convergence to stationary points include:

- *Cyclic Rule*: $\exists\, M > 0$ s.t. $\forall\, i \in \{1, \ldots, n\}$, $\forall\, k\ \exists\, \ell(k) \leq M$ s.t. $i \in W^{k+\ell(k)}$.

  This rule requires that each variables appears in the working set at least every M iterations; this way, we prevent relevant variables to be optimized from being forgotten.

- *Gauss-Southwell Rule*: $\forall k$, $i(k) \in W_k$ if $|\nabla_{i(k)} f(x^k)| \geq |\nabla_i f(x^k)|\ \forall\, i \in \{1, \ldots, n\}$.

  This rule requires to include in the selection of variables, at each iteration, the most promising variable (that is, the one associated with the highest derivative). This strategy allows, on the limit, to drive to zero the largest partial derivative, i.e., it drives to zero the entire gradient.

4.8 Stochastic Gradient Method for Finite-Sum Problems

A particularly relevant class of unconstrained nonlinear optimization problems is that of *finite-sum problems*, i.e.,

$$\min_{x \in \mathbb{R}^n}\ f(x) = \frac{1}{N} \sum_{i=1}^{N} f_i(x).$$

When the number N of terms in the sum defining the objective function is large, dealing with this problem with standard descent methods might be prohibitive. Handling the entire objective function f at once might indeed become expensive in terms of memory or computational time.

Stochastic Gradient Descent (SGD) is the prototypical stochastic optimization method, dating back to 1950s, designed to tackle this class of problems in a specialized manner. The main idea of stochastic optimization of finite-sum problems is to update the variables at each iteration by descent steps, approximating the true gradient of the function $\nabla f(x)$ by the direction

$$d_k = -\nabla f_{i_k}(x^k),$$

where i_k is an index randomly drawn from $\{1, \ldots, N\}$ at iteration K. The approximation somehow alleviates the burden of derivatives computation, as we only need to compute one gradient ∇f_i at each iteration instead of the N terms needed to evaluate the true ∇f.

We thus have updates of the form

$$x^{k+1} = x^k - \alpha_k \nabla f_{i_k}(x^k), \tag{19}$$

where the step size α_k is usually set to a constant, or at most follows a predefined sequence of values. Classical line searches are instead not generally suitable in this case, as the objective function changes at each iteration: on the one hand, we cannot guarantee a sufficient decrease of the true f; on the other hand, forcing a sufficient decrease on the current approximation may not provide us with actual benefits.

The rationale for this stochastic choice of the search direction is that, if we look at the expected value of this quantity, recalling that for a uniform distribution over the N values $\{1, \dots, N\}$ we have $p_i = \frac{1}{N}$, we get

$$\mathbb{E}[\nabla f_i(x^k)] = \sum_{i=1}^{N} p_i \nabla f_i(x^k) = \sum_{i=1}^{N} \frac{1}{N} \nabla f_i(x^k) = \frac{1}{N} \sum_{i=1}^{N} \nabla f_i(x^k) = \nabla f(x^k);$$

in other words, on average we expect to get the true gradient.

Formally, we say that a direction d_k such that $\mathbb{E}[d_k] = -\nabla f(x^k)$ is an *unbiased estimator* of the true gradient $\nabla f(x^k)$. In principle, we would like to employ a search direction, among many unbiased estimators of $-\nabla f(x^k)$, one with small variance. Indeed, reducing variance would imply a lower probability of making large errors in the estimation of the gradient direction.

Variance-reduction strategies have been proposed for SGD, leading to nice theoretical improvements in the convergence rate of the algorithm; however, these approaches require either large memory requirements or to compute the true gradient at some iterations; for this reason, these methods can only be used for particularly structured problems.

The most common way employed in practice to induce a variance-reduction effect with low cost and good performance is to estimate the gradients not based on a single term in the sum, but rather based on a subset of terms $B_k \subset \{1, \dots, N\}$, with $\|B_k\| = M \ll N$:

$$d_k = -\frac{1}{|B_k|} \sum_{i \in B_k} \nabla f_i(x^k),$$

Remark 4.1. In the context of machine and statistical learning, each term f_i corresponds to the loss associated with a sample (ref. to the following sections for more details); a subset B_k of examples is referred to as *minibatch*, opposed to the *full batch*, i.e., the entire data set[5]. Moreover, in the particular context of deep learning, the stepsize is commonly referred to as *learning rate*.

Mini-batching and stochastic gradient approaches in general are often combined with a *random reshuffling* strategy: instead of choosing the indices in B_k completely random at each iteration, macro-iterations (often called *epochs*), are carried out where all terms in the sum are used once and only once. The structure of minibatch GD with random reshuffling is shown in Algorithm **??**

Basically, within each epoch (indexed by k), $\frac{N}{M}$ iterations (indexed by t) of minibacth GD are carried out; each time, a different minibatch of functions is considered, so that there is no overlapping and within an epoch each term is considered once and only once. Note that the random split into minibatches is different for each epoch.

4.8.1 Theoretical Analysis of SGD

In this section, we report the convergence analysis for the basic SGD method, where a single, random sample is selected to approximate the gradient at each iteration; in other words, the algorithm we are going to analyze performs steps of the form (19).

[5] in learning context, classical descent methods exploiting the information about the entire objective function are referred to as *batch optimizers*

Algorithm 5: `Mini-bacth Gradient Descent with Random Reshuffling`

Input: $f_1, \dots, f_N$, x^0, $\{\alpha_k\}$.
$k = 0$;
while *stopping criterion not satisfied* **do**
  randomly split the index set $\{1, \dots, N\}$ into $\frac{N}{M}$ mini-batches $B_0^k, \dots, B_{\frac{N}{M}-1}$ of size M
  $x_0^k = x^k$
  for $t = 0, \dots, \frac{N}{M} - 1$ **do**
    $x_{t+1}^k = x_t^k - \alpha_k \frac{1}{M} \sum_{i \in B_{t^k}} \nabla f_i(x_t^k)$
  $x^{k+1} = x_{\frac{N}{M}}^k$
Output: x^k

Unlike gradient descent, SGD does not necessarily decrease the value of the objective function at each step. In order to rigorously study the algorithm, we first need some new assumption that characterizes how far the gradient samples can be from the true gradient. Assume that f is bounded below and that, for some constant $G > 0$, the magnitude of gradient samples are bounded, for all $x \in \mathbb{R}^n$, by

$$\|\nabla f_i(x)\| \leq G.$$

We will also assume that the objective function is L-smooth.

Proposition 4.14. *Let $\{x^k\}$ be the sequence generated by the SGD algorithm (**??**) with a stepsize sequence $\{\alpha_k\}$ satisfying*

$$\sum_{k=0}^{\infty} \alpha_k = \infty, \qquad \sum_{k=0}^{\infty} \alpha_k^2 < \infty.$$

Further assume that, at each iteration k, the algorithm would randomly output $z^k = x^\tau$ with probability

$$\mathbb{P}(\tau = t) = \frac{\alpha_t}{\sum_{i=0}^{k-1} \alpha_i}$$

for $t = 0, \dots, k-1$. Then,

$$\lim_{k \to \infty} E\left[\left\|\nabla f(z^k)\right\|^2\right] = 0.$$

Proof. Let k be any iteration. From Proposition 1.8 and recalling the boundedness assumption on functions ∇f_i, we get that

$$\begin{aligned} f(x^{k+1}) &= f(x^k - \alpha_k \nabla f_{i_k}(x^k)) \\ &\leq f(x^k) - \alpha_k \nabla f_{i_k}(x^k)^T \nabla f(x^k) + \frac{\alpha_k^2 L}{2} \|\nabla f_{i_k}(x^k)\|^2 \\ &\leq f(x^k) - \alpha_k \nabla f_{i_k}(x^k)^T \nabla f(x^k) + \frac{\alpha_k^2 L G^2}{2}. \end{aligned}$$

Now, the term $\alpha_k \nabla f_{i_k}(x^k)^T \nabla f(x^k)$ is not necessarily nonnegative, we are not necessarily making any progress in the objective function. We shall then see what happens in expectation:

$$\begin{aligned} E\left[f(x^{k+1})\right] &\leq E\left[f(x^k) - \alpha_k \nabla f_{i_k}(x^k)^T \nabla f(x^k) + \frac{\alpha_k^2 L G^2}{2}\right] \\ &= E\left[f(x^k)\right] - \alpha_k E\left[\nabla f_{i_k}(x^k)^T \nabla f(x^k)\right] + \frac{\alpha_k^2 L G^2}{2}. \end{aligned}$$

Now, the expected value of $\nabla f_{i_k}(x^k)$ given x^k is

$$E\left[\nabla f_{i_k}(x^k) \mid x^k\right] = \sum_{i=1}^{n} \nabla f_i(x^k) \cdot \mathbb{P}(i_k = i \mid x^k) = \sum_{i=1}^{n} \nabla f_i(x^k) \cdot \frac{1}{n} = \nabla f(x^k),$$

so [6]

$$E\left[f(x^{k+1})\right] \leq E\left[f(x^k)\right] - \alpha_k E\left[\left\|\nabla f(x^k)\right\|^2\right] + \frac{\alpha_k^2 L G^2}{2}.$$

Applying recursively the above inequality and noting that $E[f(x^0)] = f(x^0)$, we get

$$E\left[f(x^{k+1})\right] - f(x^0) \leq -\sum_{t=0}^{k} \alpha_t E\left[\left\|\nabla f(x^t)\right\|^2\right] + \frac{LG^2}{2} \sum_{t=0}^{k} \alpha_t^2,$$

i.e.,

$$\begin{aligned}\sum_{t=0}^{k} \alpha_t E\left[\left\|\nabla f(x^t)\right\|^2\right] &\leq f(x^0) - E\left[f(x^{k+1})\right] + \frac{LG^2}{2} \sum_{t=0}^{k} \alpha_t^2 \\ &\leq f(x^0) - f^\star + \frac{LG^2}{2} \sum_{t=0}^{k} \alpha_t^2,\end{aligned}$$

where $f^\star$ is the (finite) global optimum of f. Now, let us consider the expected value of the gradient at the "output" solution z^{k+1}:

$$\begin{aligned}E\left[\left\|\nabla f(z^{k+1})\right\|^2\right] &= \sum_{t=0}^{k} E\left[\left\|\nabla f(x^t)\right\|^2\right] \cdot \mathbb{P}(z^{k+1} = x^t) \\ &= \sum_{t=0}^{k} E\left[\left\|\nabla f(x^t)\right\|^2\right] \cdot \frac{\alpha_t}{\sum_{i=0}^{k} \alpha_i} \\ &= \frac{1}{\sum_{i=0}^{k} \alpha_i} \sum_{t=0}^{k} \alpha_t E\left[\left\|\nabla f(x^t)\right\|^2\right].\end{aligned}$$

We hence have

$$E\left[\left\|\nabla f(z^{k+1})\right\|^2\right] \leq \frac{1}{\sum_{i=0}^{k} \alpha_i} \left(f(x^0) - f^\star + \frac{LG^2}{2} \sum_{t=0}^{k} \alpha_t^2\right).$$

Taking the limits for $k \to \infty$, recalling that $\sum \alpha_t = \infty$ and $\sum \alpha_t^2 < \infty$, we get

$$\lim_{k \to \infty} E\left[\left\|\nabla f(z^{k+1})\right\|^2\right] = 0.$$

□

A direct result of the above proposition is the following one.

Proposition 4.15. *Let $\{x^k\}$ be the sequence generated by the SGD algorithm (**??**) with a stepsize sequence $\{\alpha_k\}$ satisfying*

$$\sum_{k=0}^{\infty} \alpha_k = \infty, \qquad \sum_{k=0}^{\infty} \alpha_k^2 < \infty.$$

Then,

$$\liminf_{k \to \infty} \|\nabla f(x^k)\| = 0.$$

[6] exploiting the law of iterated expextation: $E[X] = E_Y[E_X[X|Y]]$

The above proposition tells us that, if the same assumption on the stepsizes required for Proposition 4.14 holds, we can get a result of convergence to stationarity, in expectation, for sequence $\{x^k\}$.

A step size schedule that ensures convergence in expectation to stationary points for the SGD algorithm is, for example, given by the following rule:

$$\alpha_k = \frac{\alpha_0}{k+1}.$$

Basically, steps shall go to zero to converge, but "slowly" enough to allow the algorithm reach a stationary point.

As for the complexity of the algorithm, the speed of convergence is lower than that of full-batch methods, as we can observe in Table 3. The worst-case complexity bound is worse for SGD than for GD in the nonconvex, convex and strongly convex cases. In particular, in the strongly convex case we have linear vs. sublinear convergence rates for the two algorithms.

	nonconvex f	**convex f**	**strongly convex f**
GD	$\mathcal{O}(\frac{1}{\epsilon^2})$	$\mathcal{O}(\frac{1}{\epsilon})$	$\mathcal{O}(\log(\frac{1}{\epsilon}))$
SGD	$\mathcal{O}(\frac{1}{\epsilon^4})$	$\mathcal{O}(\frac{1}{\epsilon^2})$	$\mathcal{O}(\frac{1}{\epsilon})$

Table 3: Examples of complexity types. The values in the table should help visualizing trends; however recall that the bounds hold asymptotically, i.e., are more accurate for small values of ϵ.

Moreover, acceleration does not improve the rate of SGD. However, as opposed to batch GD, SGD does not hide within the time complexity constants the number N of the summation terms of the objective functions (i.e., the per-iteration cost is much smaller than that of batch GD). The advantage of GD w.r.t. SGD is thus only observed, ion practice, for very small values of ϵ, i.e., only when high accuracy is required for the solutions (see Figure 9. This is one of the main reasons why minibatch GD ($1 < |B| = M \ll N$), representing the middle way between batch and stochastic GD, is in practice the most effective approach in applications.

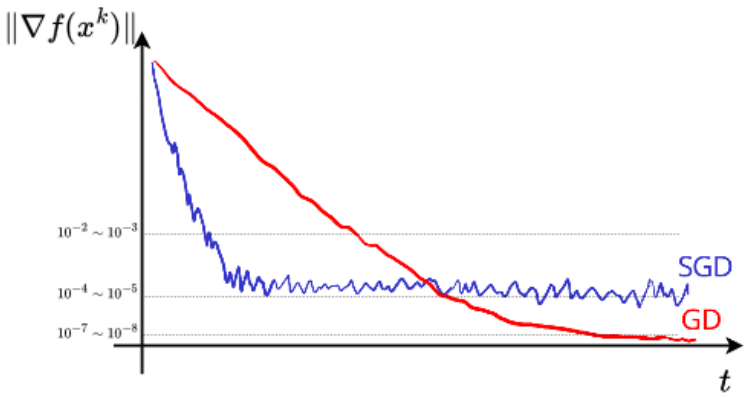


Figure 9: Typical decrease of the (true) objective function - SGD v. GD.

5 Constrained Optimization Algorithms

TBD

6 Optimization Problems in Machine Learning: Basics

6.1 Introduction

Machine learning is not optimization. Machine learning is a branch of artificial intelligence techniques, that proved to be highly successful on numerous tasks in recent years; its success is arguably attributable to the strong statistical properties possessed by learning models, that allow to properly make use of the large amount of data available nowadays; moreover, the immense ongoing advance of hardware and software tools played a crucial role in making learning systems actually, effectively employable.

Still, *mathematical optimization* is a core cog for this technology: the "engine" metaphor is often used. Indeed, the training process of a learning system consists, for the vast majority of models, in the solution of an optimization problem. Many machine learning libraries have become popular and widely employed in recent years; in each of them, hitting the "run button" does nothing else than staring an optimization algorithm.

For this reason, it is extremely important for machine learning experts and engineers to know in detail the mechanisms within these processes; this is especially true in order to be able to interpret bad behaviors and fix issues that frequently occur when designing and implementing learning systems.

Throughout the course, we will focus on the most relevant optimization algorithms employed with *supervised learning* tasks. Thus, we have a dataset

$$\mathcal{D} = \left\{ (x^{(i)}, y^{(i)}) \mid x^{(i)} \in \mathcal{X},\ y^{(i)} \in \mathcal{Y},\ i = 1, \dots, n \right\},$$

where $\mathcal{X} \subseteq \mathbb{R}^p$ and either $\mathcal{Y} = \mathbb{R}$ (*regression tasks*) or $\mathcal{Y} = \{0, 1\}$ (*binary classification tasks*). The dataset represents a sampling from some distribution, where some relation f exists between pairs (x, y), such that $f(x) = y$.

The aim in machine learning is to construct, based on the pairs in $\mathcal{D}$, a function $\hat{f}$ that captures the essence of f, being able to accurately provide values of $\hat{y} = \hat{f}(x)$ for point x that are not present in the training set $\mathcal{D}$.

Training is typically modeled as an optimization problem (*empirical risk minimization*), where a *loss function* has to be minimized w.r.t. the parameters w of the model f. The usual form of training optimization problems is

$$\min_{w}\ L(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(x^{(i)}; w), y^{(i)}), \tag{20}$$

i.e., we want to minimize the finite sum of terms that can have different forms, depending on the specific loss employed; examples of loss functions are

- **Square loss:** $\ell(u, v) = (u - v)^2$ (regression);
- **ℓ_1 loss:** $\ell(u, v) = |u - v|$ (regression);
- **Log loss:** $\ell(u, v) = -(u \log(v) - (1 - u) \log(1 - v))$ (binary classification);
- **0-1 loss:** $\ell(u, v) = 1 - \mathbb{1}\{u = v\}$ (binary classification);
- **Hinge loss:** $\ell(u, v) = \max\{0, 1 - uv\}$ (binary classification).

Now, being able to effectively solve the optimization problem (20) is not sufficient to guarantee that the resulting model will work well on out-of-sample data; two unfortunate situations often occur:

- if data quality is poor, or if the model is too simple compared to data distribution, a good approximation of the "true" f cannot be identified even if the optimization problem is accurately solved; this problem is referred to as *underfitting* and is not addressable with mathematical optimization tools alone;

- the second problem is somewhat the converse of the first one and occurs when the optimization problem is solved "too well"; in fact, a surrogate objective function is used in problem (20): the error on the training set is minimized, whereas we would like to minimize the error on the entire data distribution, including unseen data; if the learning model is sufficiently expressive (which is often the case with large *parametric models*) a very complicated prediction function might be obtained, perfectly tailored for data in $\mathcal{D}$, but absolutely incorrect with unseen data. This issue is referred to as *overfitting*.

In order to partially mitigate the latter problem, a regularization term is usually introduced in the training problem to enhance the generalization ability of the learning model. The resulting optimization problem is given by

$$\min_w \; L(w) + \Omega(w), \tag{21}$$

where the regularization term $\Omega(w)$ is usually set as one among:

- $\Omega(w) = \|w\|_2^2$ (quadratic regularization);
- $\Omega(w) = \|w\|_1$ (ℓ_1 regularization);
- $\Omega(w) = \|w\|_0$ (ℓ_0 regularization; $\|w\|_0 = |\{i : w_i \neq 0\}|$).

The quadratic regularizer is the most often employed for its simplicity and its regularity properties. The ℓ_1 and ℓ_0 regularizers are sparsity-inducing penalties, the former one having much stronger regularity properties than the latter one. Sparsity is often a valuable characteristic in predictive models.

From now on, we will not focus on the statistical details of learning models, but we will only consider the pure mathematical optimization point of view.

As a matter of fact, we shall note that adding a quadratic regularization term into the optimization problem not only has a statistical value, improving the generalization properties of the trained model, but it also turns convex objective functions into *strongly convex* functions. Keeping into account the discussion on computational complexity of optimization algorithms in Section 4.2.3, we can point out that regularization should also significantly speed up the optimization process. Moreover, the regularization term makes any bounded loss function coercive (even in absence of convexity properties), ensuring the existence of solutions to the underlying optimization problem.

6.2 Linear Regression

The simplest model for regression tasks is the *linear regression* one. Linear regressors are usually obtained by solving a (regularized) *least squares problem*:

$$\min_{w \in \mathbb{R}^p} \; \|Aw - b\|^2 + \lambda \|w\|^2 \tag{22}$$

where $A \in \mathbb{R}^{n \times p}$ and $b \in \mathbb{R}^n$. The problem is convex and is thus equivalent to finding a solution with zero gradients. Letting

$$f(w) = \|Aw - b\|^2 + \lambda \|w\|^2 = w^T A^T A w - 2 w^T A^T b + \|b\|^2 + \lambda w^T w$$

and

$$\nabla f(w) = 2A^T A w - 2A^T b + 2\lambda w,$$

the problem is hence equivalent to solving the following linear system of equations, usually referred to as *normal equations*:

$$(A^T A + \lambda I) w = A^T b. \tag{23}$$

Proposition 6.1. *Problem* (22) *admits a unique optimal solution.*

Proof. The objective function is coercive:

$$\lim_{\|w\|\to\infty} \|Aw-b\|^2 + \lambda\|w\|^2 \geq \lim_{\|w\|\to\infty} \lambda\|w\|^2 = +\infty.$$

Hence, by Weierstrass theorem, the problem admits solution; the Hessian matrix of the objective function is given by

$$\nabla^2 f(w) = 2A^TA + 2\lambda I,$$

which is positive definite; indeed, for any $w \neq 0$, we have

$$2w^TA^TAw + 2w^T(\lambda I)w = 2\|Aw\|^2 + 2\lambda\|w\|^2 \geq \lambda\|w\|^2 > 0.$$

Therefore, $f(w)$ is strictly convex and the (global) minimizer is unique. □

The system of equations (23) has a unique solution, that can be computed:

- in closed form, by matrix inversion: $w^\star = (A^TA + \lambda I)^{-1}A^Tb$; this approach can be used if p is relatively small (the cost of matrix inversion is $\mathcal{O}(p^3)$) and if A is not ill-conditioned;
- using an iterative method, such as gradient descent or Newton's method; in fact, the *conjugate gradient* method is often employed with linear systems.

6.2.1 Regularization-free case

The problem

$$\min_{w\in\mathbb{R}^p} \|Aw-b\|^2 \tag{24}$$

has similar properties as its ℓ_2-regularized counterpart and can be solved accordingly, but showing that the solution of the problem always exists is a little bit trickier; moreover, the solution is not always unique, if the rank of A is not guaranteed to be full; if A is not full-rank, the Hessian matrix A^TA is not strictly positive-definite and f is not necessarily coercive nor strictly convex.

Proposition 6.2. *Problem* (24) *always admits solution.*

Proof. Let us consider the problem

$$\begin{aligned} \min_z \ & \frac{1}{2}\|b-z\|^2 \\ \text{s.t. } & A^Tz = 0. \end{aligned}$$

The objective function of the problem is coercive and the feasible set is closed, hence the problem admits a solution, that is unique being the objective function strictly convex; since the objective is quadratic and the constraints linear, KKTs are necessary and sufficient conditions of optimality: $\exists\mu^\star \in \mathbb{R}^p$ such that

$$\nabla_z L(z^\star,\mu^\star) = -(b-z^\star) + A\mu^\star = 0, \qquad \text{with } A^Tz^\star = 0.$$

Hence, $b = z^\star + A\mu^\star$, with $z^\star : A^Tz^\star = 0$ and $\mu^\star \in \mathbb{R}^p$. We have retrieved a basic result from geometry:

$$b = b_R + b_N,$$
$$b_R = A\mu^\star \in Im(A) \quad (\exists y : Ay = b_R), \qquad b_N = z^\star \in Ker(A^T) \quad (A^Tb_N = 0).$$

Now, we can observe that

$$A^Tb = (A^Tz^\star + A^TA\mu^\star) = A^TA\mu^\star,$$

i.e., $b_R = A\mu^\star$ is solution of the normal equations. □

6.3 Linear Classifiers and Logistic Regression

THIS SECTION WILL BE HEAVILY RESTRUCTURED IN THE NEAR FUTURE

In this section, we address the problem of fitting a logistic regression model; the optimization problem for this setting has the form

$$\min_{w\in\mathbb{R}^p} \mathcal{L}(w) + \lambda\Omega(w), \tag{25}$$

where $\mathcal{L}(w)$ is the negative log-likelihood function of the logistic model, which is a convex function, and $\Omega(w)$ is a convex regularizer. Note that this setting is conceptually equivalent to other training problems with convex loss functions, such as softmax regression or ARMA models fitting in time series.

For problems of this form, an iterative algorithm is required to train the model; methods such as gradient descent or Newton's method are all viable options; however, the L-BFGS algorithm is generally considered the most efficient method for unconstrained optimization, both in the convex case and in the nonconvex case when global optimality is not actually required, with problems up to a considerably large size. With huge datasets, SGD type method can also be considered.

Here we show how the gradients and the Hessian matrix for the logistic regression problem can be computed. We first recall that, letting $\mathcal{Y} = \{-1, 1\}$ the loss function has the form

$$\mathcal{L}(w; X, y) = \sum_{i=1}^{n} \log(1 + \exp(-y^{(i)} w^T x^{(i)})).$$

If we set $z = Xw$ (i.e., $z_i = w^T x^{(i)}$ for all i), we have

$$\mathcal{L}(w; X, y) = \phi(z; y) = \sum_{i=1}^{n} \log(1 + \exp(-y^{(i)} z_i)).$$

We want to compute $\nabla_w \mathcal{L}(w; X, y)$; by the multivariate chain rule, we have

$$\nabla_w \mathcal{L}(w; X, y)^T = \nabla_z \phi(Xw; y)^T \frac{\partial}{\partial w}(Xw).$$

It is also easy to observe that

$$\begin{aligned}
\frac{\partial}{\partial z_i}\phi(z; y) &= \frac{\partial}{\partial z_i}(\log(1 + \exp(-y^{(i)} z_i))) \\
&= \frac{1}{1 + \exp(-y^{(i)} z_i)} \exp(-y^{(i)} z_i)(-y^{(i)}) \\
&= -y^{(i)} \frac{1}{1 + \exp(y^{(i)} z_i)} = -y^{(i)} \sigma(-y^{(i)} z_i),
\end{aligned}$$

where $\sigma(\cdot)$ is the sigmoid function. Hence,

$$\nabla_z \phi(z; y) = (-y^{(1)} \sigma(-y^{(1)} z_1), \ldots, -y^{(n)} \sigma(-y^{(n)} z_n))^T.$$

On the other hand, we have

$$\frac{\partial}{\partial w}(Xw) = X,$$

therefore

$$\nabla_w \mathcal{L}(w; X, y) = (r^T X)^T = X^T r$$

with $r \in \mathbb{R}^n$, $r_i = -y^{(i)} \sigma(-y^{(i)} w^T x^{(i)})$ for all $i = 1, \ldots, n$.

By similar calculations we can also get

$$\nabla^2 \mathcal{L}(w; X, y) = X^T D X,$$

where D is a diagonal matrix with $d_{ii} = \sigma(y^{(i)} w^T x^{(i)}) \sigma(-y^{(i)} w^T x^{(i)})$.

7 Support Vector Machines

A (linear) Support Vector Machine (SVM) model is, in brief, the classification model obtained solving the empirical risk minimization problem with the hinge loss and the ℓ_2 regularizer, i.e.,

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \max\{0, 1 - y^{(i)}(w^T x^{(i)} + b)\}.$$

It is easy to realize that the optimal solutions of the above nonsmooth unconstrained optimization problem are also solution of the following smooth problem with linear constraints:

$$\begin{aligned} \min_{w,b,\xi} \ & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi^{(i)} \\ \text{s.t.} \ & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi^{(i)}, \\ & \xi^{(i)} \geq 0. \end{aligned} \tag{26}$$

It can be shown that, when $C = \infty$, the problem amounts to finding, among the hyperplanes perfectly separating the training data, the one maximizing the distance from the closest point (of both classes), see Figures 10 and 11. Of course, however, the problem has no feasible solution in that case if data are not linearly separable (Figure 12).

Hence, with a finite choice of C, we accept to "pay a cost" for all points that are classified incorrectly, and even for those classified correctly but with an insufficient confidence (Figure 13). Not only this allows to guarantee the existence of a solution, but it is also useful to avoid overfitting issues.

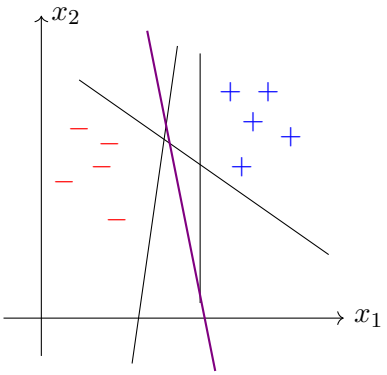


Figure 10: Classification through separating hyperplanes. Each one of the hyperplanes in figure solves correctly the classification task of training data. SVM selects amongst them the one which is equidistant from the two classes (the purple one).

Now, we turn to the study of the optimization problem (26). The problem is convex quadratic with linear constraints, thus it is in principle solvable by standard constrained solvers, and in particular the Frank-Wolfe algorithm. However, the number of constraints is proportional to the number of training data points, and might grow large quite fast. For this reason (and also others that we will address later), a different path shall be followed.

7.1 The dual problem

As aforementioned, problem (26) is convex quadratic with linear constraints. Thus, KKTs are necessary and sufficient conditions of optimality. Now, without delving deep into an important but vast and complex topic, we introduce a result from *duality theory* concerning constrained convex problems.

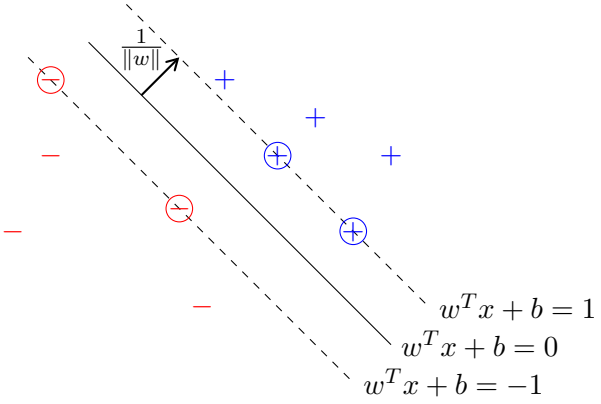


Figure 11: Maximum margin hyperplane. The circled points, that define the separation margin, are called support vectors.

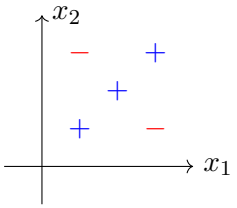


Figure 12: Example of a non linearly separable dataset.

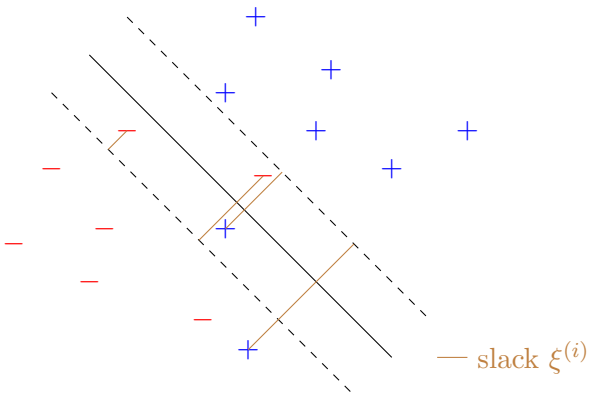


Figure 13: SVM classifier obtained allowing non null slacks.

Proposition 7.1. *Let us consider the optimization problem*

$$\begin{aligned} &\min_x f(x) \\ &s.t.\ g(x) \leq 0, \end{aligned}$$

being f and g continuously differentiable convex functions. Let $x^\star$ be an optimal solution for the problem and assume $\mu^\star$ is a vector of multipliers such that $(x^\star, \mu^\star)$ satisfies the KKT conditions. Then $(x^\star, \mu^\star)$ is an optimal solution of the so called Wolfe dual problem.

$$\begin{aligned} &\max_{x,\mu} \mathcal{L}(x,\mu) = f(x) + \mu^T g(x) \\ &s.t.\ \nabla_x \mathcal{L}(x,\mu) = 0, \\ &\quad \mu \geq 0. \end{aligned}$$

Proof. The pair $(x^\star, \mu^\star)$ satisfies KKT conditions, i.e.,

$$\nabla_x \mathcal{L}(x^\star, \mu^\star) = 0, \quad \mu^\star \geq 0, \quad g(x^\star) \leq 0, \quad \mu_i^\star g_i(x^\star) = 0 \ \forall\, i.$$

Thus, $(x^\star, \mu^\star)$ satisfies the constraints of the Wolfe dual problem. Moreover, by the complementarity condition, $\mathcal{L}(x^\star, \mu^\star) = f(x^\star) + \sum_i \mu_i^\star g_i(x^\star) = f(x^\star)$. Now, let (x, μ) be any feasible solution for the dual problem. Since $\mu \geq 0$ and $g(x^\star) \leq 0$, we get

$$\mathcal{L}(x^\star, \mu^\star) = f(x^\star) \geq f(x^\star) + \sum_i \mu_i g_i(x^\star) = \mathcal{L}(x^\star, \mu).$$

Then, by the convexity of $\mathcal{L}(x, \mu)$ w.r.t. x variables (it is the positive linear combination of the convex functions $f, g_1, \ldots, g_m$), we can also write

$$\mathcal{L}(x^\star, \mu) \geq \mathcal{L}(x, \mu) + \nabla_x \mathcal{L}(x, \mu)^T (x^\star - x).$$

Putting the pieces together and recalling that $\nabla_x \mathcal{L}(x, \mu) = 0$ being (x, μ) feasible for the dual problem, we obtain

$$\mathcal{L}(x^\star, \mu^\star) \geq \mathcal{L}(x^\star, \mu) \geq \mathcal{L}(x, \mu).$$

Being (x, μ) an arbitrary feasible solution of the dual problem, we get the thesis. □

We can now go back to the SVM problem (26). Being KKTs necessary and sufficient conditions for global optimality, for an optimal solution $(w^\star, b^\star, \xi^\star)$ there have to exist multipliers $(\alpha^\star, \mu^\star)$ such that $(w^\star, b^\star, \xi^\star, \alpha^\star, \mu^\star)$ is a KKT point and, by the above proposition, is the solution of the dual problem

$$\begin{aligned} &\max_{w,b,\xi,\alpha,\mu} \mathcal{L}(w,b,\xi,\alpha,\mu) = \frac{1}{2}\|w\|^2 + C\sum_i \xi_i + \sum_i -\mu_i \xi_i + \sum_i \alpha_i (1 - y^{(i)}(w^T x^{(i)} + b) - \xi_i) \\ &\text{s.t. } \alpha \geq 0, \quad \mu \geq 0, \\ &\quad \nabla_w \mathcal{L}(w,b,\xi,\alpha,\mu) = 0, \\ &\quad \nabla_b \mathcal{L}(w,b,\xi,\alpha,\mu) = 0, \\ &\quad \nabla_\xi \mathcal{L}(w,b,\xi,\alpha,\mu) = 0. \end{aligned}$$

We can then observe that the constraints imply:

$$0 = \nabla_w \mathcal{L}(w,b,\xi,\alpha,\mu) = w - \sum_i \alpha_i y^{(i)} x^{(i)}, \quad \text{i.e.,} \quad w = \sum_i \alpha_i y^{(i)} x^{(i)},$$

$$0 = \nabla_b \mathcal{L}(w,b,\xi,\alpha,\mu) = -\sum_i \alpha_i y^{(i)}, \quad \text{i.e.,} \quad \alpha^T y = 0$$

$$0 = \nabla_\xi \mathcal{L}(w,b,\xi,\alpha,\mu) = Ce - \sum_i \mu_i e_i - \sum_i \alpha_i e_i, \quad \text{i.e.,} \quad \alpha_i = C - \mu_i \leq C \ \forall\, i.$$

Manipulating the objective function, we can first obtain

$$\frac{1}{2}w^T w + C\sum_i \xi_i + \sum_i -\mu_i\xi_i + \sum_i \alpha_i + \\ -\sum_i \alpha_i y^{(i)} w^T x^{(i)} - b\sum_i \alpha_i y^{(i)} - \sum_i \alpha_i \xi_i,$$

then we shall recall that $\sum_i \alpha_i y^{(i)} = 0$ by the second condition above, whereas using the third one we can write $C\sum_i \xi_i - \sum_i \mu_i\xi_i = \sum_i \xi_i(C - \mu_i) = \sum_i \xi_i\alpha_i$. We therefore get

$$\frac{1}{2}w^T w + \sum_i \alpha_i\xi_i + \sum_i \alpha_i - \sum_i \alpha_i y^{(i)} w^T x^{(i)} - \sum_i \alpha_i\xi_i,$$

i.e.,

$$w^T\left(\frac{1}{2}w - \sum_i \alpha_i y^{(i)} x^{(i)}\right) + \sum_i \alpha_i.$$

Now, we can substitute $w = \sum_i \alpha_i y^{(i)} x^{(i)}$ to finally obtain that

$$\mathcal{L}(w, b, \xi, \alpha, \mu) = -\frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y^{(i)} y^{(j)} {x^{(i)}}^T x^{(j)} + \sum_i \alpha_i$$

for any feasible solution of the Wolfe dual problem. In other words, the objective is only function of multipliers α, that are solely constrained by $y^T\alpha = 0$, $\alpha \geq 0$ and $\alpha \leq C$.

Swapping the signs and turning to matrix notation, after defining the matrix Q such that $Q_{ij} = y^{(i)} y^{(j)} {x^{(i)}}^T x^{(j)}$, we end up with the dual problem

$$\begin{aligned} &\min_\alpha \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \\ &\text{s.t. } \alpha^T y = 0, \\ &\qquad 0 \leq \alpha_i \leq C \quad \forall\, i. \end{aligned} \tag{27}$$

Once problem (27) has been solved, we can use the solution $\alpha^\star$ to retrieve the other variables:

$$w^\star = \sum_i \alpha_i^\star y^{(i)} x^{(i)}, \quad \mu_i^\star = C - \alpha_i^\star \;\forall\, i,$$

$$b^\star = \frac{1}{y^{(i)}} - {w^\star}^T x^{(i)} \text{ for any } i \text{ s.t. } \alpha_i^\star \in (0, C), \quad \xi_i^\star = \max\{0, 1 - y^{(i)}({w^\star}^T x^{(i)} + b^\star)\},$$

where the third equation (for $b^\star$) follows from imposing the complementarity slackness conditions form KKTs.

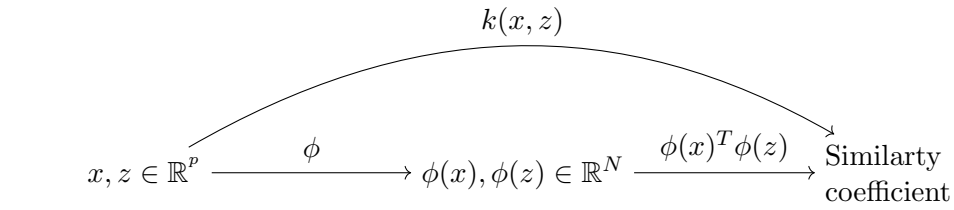
We shall note that, by the complementarity conditions, we have

$$\alpha_i^\star(1 - y^{(i)}({w^\star}^T x^{(i)} + b^\star) - \xi_i^\star) = 0 \text{ and } 0 = \mu_i^\star\xi_i^\star = (C - \alpha_i^\star)\xi_i^\star.$$

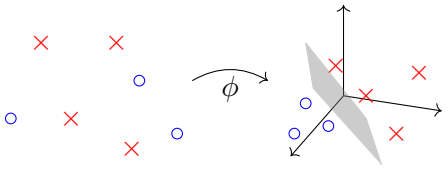
Thus, if $\alpha_i^\star \in (0, C)$, we have $\xi_i^\star = 0$ and $y^{(i)}({w^\star}^T x^{(i)} + b^\star) = 1$, i.e., the i-th data sample exactly lies on the border of the separation margin; on the other hand, if $\alpha_i^\star < C$, then $\xi_i^\star = 0$: the samples for which we pay a penalty are only those associated with multipliers $\alpha_i^\star$ with value at the upper bound. All the points associated with nonzero multipliers $\alpha_i^\star$ are called support vectors; we can observe that, since $w^\star = \sum_i \alpha_i^\star y^{(i)} x^{(i)}$, the resulting classifier is only based on support vectors.

When classifying a new data point x, what we actually do is computing

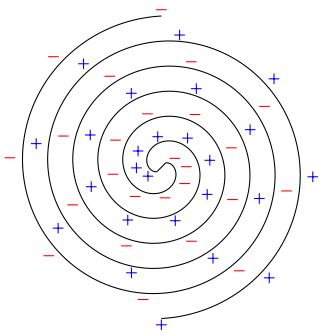
$${w^\star}^T x = \sum_i \alpha_i y^{(i)} x^T x^{(i)};$$



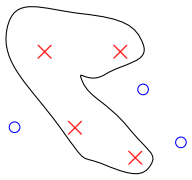
(a) Kernels concept



(b) Kernels and feature maps



(c) Example of separation surface obtainable with kernels



(d) Example of separation surface obtainable with kernels

Figure 14: Kernels

in other words, the outcome of the decision function is a weighted sum over training data, where only support vectors are actually taken into account; for each support vector, we carry out the dot product with the point to be classified: the higher the dot product is, the higher is the similarity between $x^{(i)}$ and x, the larger will be the contribution of that support vector towards assigning class $y^{(i)}$ to the new data point.

The dot product is not the only "similarity" measure available for comparing two data points; in fact, we might think of substituting the terms $x^T x^{(i)}$ in the decision function with *kernel* functions, $k(\cdot, \cdot) : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ (see Figure 14); in this case, the elements of matrix Q in problem (27) shall be defined as $Q_{ij} = y^{(i)} y^{(j)} k(x^{(i)}, x^{(j)})$. Without delving deep into kernel theory, we shall just recall that a function k is a valid kernel to be used if and only if:

- the matrix Q is positive semi-definite for any possible dataset $\mathcal{D}$ (we are guaranteed that the dual is actually a convex problem);
- the kernel function represents a dot product between the data points mapped to some possibly higher dimensional space.

In these cases, of course, the decision function becomes

$$h(x) = \sum_i \alpha_i^\star y^{(i)} k(x, x^{(i)}),$$

which is in general not a linear function. From the one hand, it is no more possible to express the classifier in terms of weights w; on the other hand, the classifier is nonlinear, so that more powerful classifiers can be constructed.

To sum up, the two major reasons to consider the dual SVM problem are:

- by the kernel trick, it is possible to obtain nonlinear classifiers;
- the problem is convex quadratic with linear constraints, like the primal, but constraints are simpler (the only "difficult" constraint is $\alpha^T y = 0$).

7.2 Solving the Dual Problem

In this section we show how the dual problem for nonlinear SVM training can be efficiently solved. We first recall the formulation with some notation changes to make it clearer from an optimization perspective:

$$\begin{aligned} \min\ & \frac{1}{2}x^T Q x - e^T x \\ \text{s.t. } & a^T x = 0, \\ & 0 \le x \le C, \end{aligned} \tag{28}$$

where $Q \in \mathbb{R}^{n\times n}$ is a positive semi-definite matrix (n is training set dimension) and $e = (1 \dots 1)^T$. Typically n is large and Q is dense; these facts contribute to problem hardness. The other source of complexity is the linear constraint $a^T x = 0$. Box constraints are easy to treat, since they are component-wise separable.

This problem is efficiently solved through a decomposition strategy: at each iteration a working set $W \subset \{1, \dots, n\}$ is selected; $\overline{W} = \{1, \dots, n\} \setminus W$ denotes its complementary. The resulting subproblem is therefore

$$\begin{aligned} \min_{x_W}\ & f(x_W, x^k_{\overline{W}}) = \frac{1}{2}x_W^T Q_{WW} x_W - (e_W - Q_{W\overline{W}} x^k_{\overline{W}})^T x_W \\ \text{s.t. } & a_W^T x_W = -a_{\overline{W}}^T x^k_{\overline{W}}, \\ & 0 \le x_W \le C, \end{aligned} \tag{29}$$

whose solution is $x^\star_W$. Thus, variables are updated as follows:

$$x_i^{k+1} = \begin{cases} x_i^\star & \text{if } i \in W, \\ x_i^k & \text{if } i \notin W. \end{cases}$$

At this point, two questions arise. The first one is the minimum cardinality of W: it must be $|W| \ge 2$; if it were $W = \{i\}$ it would always be $x^{k+1} = x^k$, since both x^k and x^{k+1} are feasible and satisfy $a_i x_i = -a_{\overline{W}}^T x^k_{\overline{W}}$.

Decomposition algorithms are classified according to the working set cardinality: if $|W| = 2$, we talk about *Sequential Minimal Optimization* (SMO) algorithms, that do not require a solver for the subproblem; otherwise, if $|W| > 2$, the algorithm needs a solver to find a solution to (29).

The second issue is the selection of W; we will deepen this aspect in the rest of this section.

7.2.1 Sequential Minimal Optimization

In the case $|W| = 2$, subproblem (29) becomes

$$\begin{aligned} \min\ & \frac{1}{2}\begin{bmatrix} x_i & x_j \end{bmatrix}\begin{bmatrix} q_{ii} & q_{ji} \\ q_{ij} & q_{jj} \end{bmatrix}\begin{bmatrix} x_i \\ x_j \end{bmatrix} - x_i - x_j + p_{\overline{W}}^T \begin{bmatrix} x_i \\ x_j \end{bmatrix} \\ \text{s.t. } & a_i x_i + a_j x_j = b, \\ & 0 \le x_i, x_j \le C, \end{aligned} \tag{30}$$

which is a quadratic convex problem in the two variables x_i, x_j. The solution to this problem can be computed analytically: not needing to employ any solver is the main benefit of SMO algorithms.

Now, we focus on the selection strategy of subproblem variables; we want

$$x^{k+1} = (x_1^k, \dots, x_i^{k+1}, \dots, x_j^{k+1}, \dots, x_n^k)$$

to be feasible and

$$f(x^{k+1}) < f(x^k).$$

To achieve these goals, we have to identify at each step a feasible descent direction with two and only two non-zero components.

Proposition 7.2. *The set of feasible directions for Problem* (28) *at point* $\bar{x}$ *is given by*

$$\mathcal{D}(\bar{x}) = \left\{ d \in \mathbb{R}^n \mid a^T d = 0,\ d_i \geq 0\ \forall\, i \in L(\bar{x}),\ d_i \leq 0\ \forall\, i \in U(\bar{x}) \right\}$$

where

$$L(\bar{x}) = \{ i \in \{1, \ldots, n\} \mid \bar{x}_i = 0 \}\,;$$
$$U(\bar{x}) = \{ i \in \{1, \ldots, n\} \mid \bar{x}_i = C \}\,.$$

Proof. Let's denote with $\mathcal{S}$ the *feasible set*

$$\mathcal{S} = \{ x \in \mathbb{R}^n \mid a^T x = 0,\ 0 \leq x \leq C \}$$

Let $\bar{x} \in \mathcal{S}$ and let $\mathcal{D}(\bar{x})$ be the set of feasible directions at $\bar{x}$; if $d \in \mathcal{D}(\bar{x})$, then $\bar{x} + td \in \mathcal{S}\ \forall\, t \in [0, \bar{t}\,]$ for $\bar{t}$ sufficiently small. Thus $a^T(\bar{x} + td) = 0$, i.e., $a^T\bar{x} + ta^T d = 0$, which implies

$$a^T d = 0. \tag{31}$$

Furthermore $0 \leq \bar{x} + td \leq C$, which implies

$$\begin{cases} d_i \leq 0 & \text{if } \bar{x}_i = C \\ d_i \geq 0 & \text{if } \bar{x}_i = 0. \end{cases} \tag{32}$$

Putting conditions (31) and (32) together we get the thesis. □

We are looking for directions in $\mathcal{D}(\bar{x})$ with two non-zero components:

$$d^{i,j} = \begin{pmatrix} 0 & \ldots & d_i & \ldots & d_j & \ldots & 0 \end{pmatrix}^T. \tag{33}$$

Since $a^T d^{i,j} = 0$, we have $a_i d_i + a_j d_j = 0$: we can choose

$$d_i = \frac{1}{a_i}, \qquad d_j = -\frac{1}{a_j}. \tag{34}$$

Moreover, assume $i \in L(\bar{x})$, i.e., $\bar{x}_i = 0$. Then we need $d_i \geq 0$ and thus we can only consider variables corresponding to $a_i > 0$. On the contrary, if $i \in U(\bar{x})$, then d_i must be non-positive and we can consider the i-th component only if $a_i < 0$. Similarly, if $j \in L(\bar{x})$ we need $a_j < 0$ and if $j \in U(\bar{x})$ it has to be $a_j > 0$ (Table 4).

index	**variable value**	**feasible direction**	**coeff. constraint**
$i \in L(\bar{x})$	$\bar{x}_i = 0$	$d_i \geq 0$	$a_i > 0$
$i \in U(\bar{x})$	$\bar{x}_i = C$	$d_i \leq 0$	$a_i < 0$
$j \in L(\bar{x})$	$\bar{x}_j = 0$	$d_j \geq 0$	$a_j < 0$
$j \in U(\bar{x})$	$\bar{x}_j = C$	$d_j \leq 0$	$a_j > 0$

Table 4: Constraints on the selection of non-zero components of directions $d^{i,j}$, based on the sign of coefficients a.

We note that if $0 < \bar{x}_i < C$ there is no constraint on the sign of a_i; the same observation holds for a_j. We can subsequently partition sets U and L as follows:

$$L(\bar{x}) = L^+(\bar{x}) \cup L^-(\bar{x}) \qquad U(\bar{x}) = U^+(\bar{x}) \cup U^-(\bar{x})$$

where

$$L^+(\bar{x}) = \{ h \in L(\bar{x}) \mid a_h > 0 \} \qquad L^-(\bar{x}) = \{ h \in L(\bar{x}) \mid a_h < 0 \}$$
$$U^+(\bar{x}) = \{ h \in U(\bar{x}) \mid a_h > 0 \} \qquad U^-(\bar{x}) = \{ h \in U(\bar{x}) \mid a_h < 0 \}.$$

In the end, we define the sets

$$R(\bar{x}) = L^+(\bar{x}) \cup U^-(\bar{x}) \cup \{ i \mid 0 < \bar{x}_i < C \}$$

and

$$S(\bar{x}) = L^-(\bar{x}) \cup U^+(\bar{x}) \cup \{ i \mid 0 < \bar{x}_i < C \}.$$

Proposition 7.3. *Direction $d^{i,j}$ defined as* (33)-(34) *is feasible at $\bar{x}$ for Problem* (28) *if and only if $i \in R(\bar{x})$ and $j \in S(\bar{x})$.*

Proof. Let $d^{i,j}$ be feasible and assume by contradiction that $j \notin S(\bar{x})$. Then, either $j \in L^+(\bar{x})$ or $j \in U^-(\bar{x})$, but in the first case $d_j = -1/a_j < 0$, while in the second one $d_j = -1/a_j > 0$: both cases violate the feasibility assumption. The case $i \notin R(\bar{x})$ is similar.
On the other hand, let $i \in R(\bar{x})$ and $j \in S(\bar{x})$; in particular, assume $i \in U^-(\bar{x})$ and $j \in U^+(\bar{x})$ (the other combinations can be treated similarly). Then $a^T d^{i,j} = a_i \frac{1}{a_i} - a_j \frac{1}{a_j} = 0$; moreover, since $i \in U^-(\bar{x})$, $a_i < 0$ and therefore $d_i = 1/a_i < 0$; similarly, $j \in U^+(\bar{x})$ implies $a_j > 0$ and thus $d_j = -1/a_j < 0$. This completes the proof. □

Proposition 7.4. *Direction $d^{i,j}$ defined as* (33)-(34) *is a descent direction for Problem* (28) *if and only if*

$$\frac{\nabla_i f(\bar{x})}{a_i} < \frac{\nabla_j f(\bar{x})}{a_j}. \tag{35}$$

Proof. Since f is a quadratic convex function, d is a descent direction at $\bar{x}$ if and only if we have

$$\nabla f(\bar{x})^T d^{i,j} = \frac{1}{a_i} \nabla_i f(\bar{x}) - \frac{1}{a_j} \nabla_j f(\bar{x}) < 0,$$

i.e., (35), is a necessary and sufficient condition of descent. □

We are now able to write the general scheme of an SMO algorithm (Algorithm 6).

Algorithm 6: `Sequential Minimal Optimization`

Input: Q, a, C.
$x^0 = 0$;
$k = 0$;
$\nabla f(x^0) = -e$;
while *stopping criterion not satisfied* **do**
  select $i \in R(x^k)$, $j \in S(x^k)$ such that $\frac{\nabla_i f(\bar{x})}{a_i} - \frac{\nabla_j f(\bar{x})}{a_j} < 0$
  $W = \{i, j\}$;
  solve analytically
    $$\min_{x_W} f(x_W, x^k_{\overline{W}}) \text{ s.t. } a_W^T x_W = -a_{\overline{W}}^T x^k_{\overline{W}}, \;\; 0 \le x_W \le C;$$
  let $x^\star_W$ be the solution found at the previous step;
    $$\text{set } x_h^{k+1} = \begin{cases} x_j^\star & \text{if } h = j, \\ x_i^\star & \text{if } h = i, \\ x_h^k & \text{otherwise;} \end{cases}$$
  $\nabla f(x^{k+1}) = \nabla f(x^k) + Q_i(x_i^{k+1} - x_i^k) + Q_j(x_j^{k+1} - x_j^k)$;
  $k = k + 1$;
Output: x^k

As we know from Propositions 7.3 and 7.4, this choice of i and j guarantees x^{k+1} feasibility and f decrease. The gradient of f at x^{k+1} is given by

$$\begin{aligned} \nabla f(x^{k+1}) &= Qx^{k+1} - e = Q(x^{k+1} - x^k) + Qx^k - e \\ &= Q_i(x_i^{k+1} - x_i^k) + Q_j(x_j^{k+1} - x_j^k) + \nabla f(x^k), \end{aligned}$$

since Qx^{k+1} and Qx^k only differ by the i-th and j-th components. This expression shows that only two columns of Q are needed to update the gradients, resulting in a saving of time. Moreover, $x^0 = 0$ and thus $\nabla f(x^0) = -e$: there is no need to compute the gradient

at the starting point. Matrix Q is thus never required in its entirety to get the gradient: this is an advantage also in terms of I/O time.

Feasibility and strict decrease are not sufficient to ensure global convergence; in the next section we will address convergence issues more deeply.

7.2.2 SMO Convergence Properties with First-order Selection Rule

The choice of the working set $W = \{i, j\}$ is the key to obtain convergence properties for SMO.

Proposition 7.5. *A point $x^\star$ is a global minimizer for problem (28) if and only if*

$$\max_{h \in R(x^\star)} \left\{ -\frac{\nabla_h f(x^\star)}{a_h} \right\} \leq \min_{h \in S(x^\star)} \left\{ -\frac{\nabla_h f(x^\star)}{a_h} \right\} \tag{36}$$

Proof. Since problem (28) is a convex problem with linear constraints, KKTs are necessary and sufficient optimality conditions:

$$\begin{aligned} \nabla_i \mathcal{L}(x, \lambda, \xi, \hat{\xi}) = \nabla_i f(x) + \lambda a_i - \xi_i + \hat{\xi}_i = 0 \quad & \forall i = 1, \ldots, n, \\ \xi_i x_i = 0 \quad & \forall i = 1, \ldots, n, \\ \hat{\xi}_i (x_i - C) = 0 \quad & \forall i = 1, \ldots, n, \\ \xi, \hat{\xi} \geq 0 \quad & \forall i = 1, \ldots, n. \end{aligned}$$

Therefore, for optimal $x^\star, \lambda^\star, \xi^\star, \hat{\xi}^\star$, we have

$$\nabla_i f(x^\star) + \lambda^\star a_i \begin{cases} \geq 0 & \text{if } i \in L(x^\star), \\ \leq 0 & \text{if } i \in U(x^\star), \\ = 0 & \text{if } 0 < x_i^\star < C, \end{cases}$$

and then

$$\frac{\nabla_i f(x^\star)}{a_i} + \lambda^\star \begin{cases} \geq 0 & \text{if } i \in L^+(x^\star) \cup U^-(x^\star), \\ \leq 0 & \text{if } i \in L^-(x^\star) \cup U^+(x^\star), \\ = 0 & \text{otherwise,} \end{cases}$$

from which follows

$$\begin{cases} \lambda^\star \leq -\frac{\nabla_i f(x^\star)}{a_i} & \forall i \in L^-(x^\star) \cup U^+(x^\star), \\ \lambda^\star \geq -\frac{\nabla_i f(x^\star)}{a_i} & \forall i \in L^+(x^\star) \cup U^-(x^\star), \\ \lambda^\star = -\frac{\nabla_i f(x^\star)}{a_i} & \forall i : 0 < x_i^\star < C. \end{cases}$$

Recalling

$$\begin{aligned} R(x^\star) &= L^+(x^\star) \cup U^-(x^\star) \cup \{i \mid 0 < x_i^\star < C\}, \\ S(x^\star) &= L^-(x^\star) \cup U^+(x^\star) \cup \{i \mid 0 < x_i^\star < C\}, \end{aligned}$$

we finally obtain (36). □

Corollary 7.6. *Let x^k be the current (feasible) solution and assume it is not optimal. Then there exist $i \in R(x^k)$ and $j \in S(x^k)$ such that*

$$-\frac{\nabla_i f(x^k)}{a_i} > -\frac{\nabla_j f(x^k)}{a_j}.$$

Proof. This Corollary is a direct consequence of Proposition 7.5. □

Definition 7.1. The *most violating pair* at the feasible, non optimal solution x^k of (28) is the pair of indices $(i^\star, j^\star)$ defined as

$$i^\star \in \arg \max_{h \in R(x^k)} \left\{ -\frac{\nabla_h f(x^k)}{a_h} \right\}, \qquad j^\star \in \arg \min_{h \in S(x^k)} \left\{ -\frac{\nabla_h f(x^k)}{a_h} \right\}. \tag{37}$$

We can think of selecting in the SMO scheme a Most Violating Pair, i.e., the pair $(i^\star, j^\star)$ that violates the most the optimality condition (36). We thus obtain the famous SVM$^{\text{light}}$ Algorithm 7, which is a SMO decomposition algorithm with convergence properties historically employed in software libraries to solve non-linear SVM training problem.

Algorithm 7: SVM$^{\texttt{light}}$

Input: Q, a, C.
$x^0 = 0$;
$k = 0$;
$\nabla f(x^0) = -e$;
while *stopping criterion not satisfied* **do**
  identify the most violating pair $(i^\star, j^\star)$
  $W = \{i^\star, j^\star\}$;
  solve analytically
    $$\min_{x_W} f(x_W, x^k_{\overline{W}}) \text{ s.t. } a_W^T x_W = -a_{\overline{W}}^T x^k_{\overline{W}}, \;\; 0 \le x_W \le C;$$
  let $x^\star_W$ be the solution found at the previous step; set
    $$x_h^{k+1} = \begin{cases} x_h^\star & \text{if } h \in W, \\ x_h^k & \text{otherwise;} \end{cases}$$
  $\nabla f(x^{k+1}) = \nabla f(x^k) + Q_{i^\star}(x^{k+1}_{i^\star} - x^k_{i^\star}) + Q_{j^\star}(x^{k+1}_{j^\star} - x^k_{j^\star})$;
  $k = k + 1$;
Output: x^k

By a quite complex proof, the following proposition can be stated.

Proposition 7.7. *The sequence $\{x_k\}$ generated by algorithm SVM^{light} has limit points, each one being a solution of problem (28).*

Last, we have to discuss about the stopping criterion. The optimality condition is (36); a reasonable stopping criterion is based on the quantities

$$m(x) = \max_{h \in R(x)} \left\{ -\frac{\nabla_h f(x^k)}{a_h} \right\}, \qquad M(x) = \min_{h \in S(x)} \left\{ -\frac{\nabla_h f(x^k)}{a_h} \right\}. \tag{38}$$

From Proposition 36, $x^\star$ is an optimal solution if and only if $m(x^\star) \le M(x^\star)$. Then we can think of defining a stopping criterion based on the following proposition.

Proposition 7.8. *Let $m(x)$ and $M(x)$ be defined as in* (38)*. Then, for any $\epsilon > 0$, algorithm SVM^{light} produces a solution x^k satisfying*

$$m(x^k) \le M(x^k) + \epsilon \tag{39}$$

within a finite number of iterations.

The proof of the above property again requires a quite complicated reasoning, mainly because of the fact the functions $m(x)$ and $M(x)$ are not continuous.

7.2.3 Working Set Selection using Second Order Information

The choice of the most violating pair as working set is directly related to the first order approximation of f. $W = \{i^\star, j^\star\}$ indeed can be proven to solve following problem:

$$\begin{aligned} \min_{W:|W|=2} \ & \min_{d_W} \nabla_W f(x^k)^T d_W \\ & \text{s.t. } a_W^T d_W = 0, \\ & \quad d_t \geq 0 \text{ if } x_t^k = 0,\ t \in W, \\ & \quad d_t \leq 0 \text{ if } x_t^k = C,\ t \in W, \\ & \quad -1 \leq d_t \leq 1,\ t \in W. \end{aligned} \tag{40}$$

Recalling our definition of d, the first order approximation of f at x^k is

$$f(x^k + d) \approx f(x^k) + \nabla f(x^k)^T d = f(x^k) + \nabla_W f(x^k)^T d_W$$

which is exactly the objective of (40). The linear constraint comes from $a^T(x^k + d) = 0$ and $a^T x^k = 0$. The second and third constraints come from $0 < x^k + d < C$. The box constraint prevents the linear objective to go to $-\infty$. Looking for the maximal violating pair is an efficient way to solve this problem (time required is $\mathcal{O}(n)$).

We might think of extending this technique to exploit second order information. In fact, this is somehow what is implemented to select the working set in the current version of LIBSVM. Since f is quadratic, we have the reduction of the objective value can be exactly expressed as

$$\begin{aligned} f(x^k + d) - f(x^k) &= \nabla f(x^k)^T d + \frac{1}{2} d^T \nabla^2 f(x^k) d \\ &= \nabla_W f(x^k)^T d_W + \frac{1}{2} d_W^T \nabla^2_{WW} f(x^k) d_W. \end{aligned} \tag{41}$$

Substituting the objective function in (40) with this quantity and removing the box constraint (which is no more necessary since the new objective function is bounded below), we get the following problem:

$$\begin{aligned} \min_{W:|W|=2} \ & \min_{d_W} \nabla_W f(x^k)^T d_W + \frac{1}{2} d_W^T \nabla^2_{WW} f(x^k) d_W \\ & \text{s.t. } a_W^T d_W = 0, \\ & \quad d_t \geq 0 \text{ if } x_t^k = 0,\ t \in W, \\ & \quad d_t \leq 0 \text{ if } x_t^k = C,\ t \in W. \end{aligned} \tag{42}$$

However, solving this problem requires $\mathcal{O}(n^2)$ operations (all $n(n-1)/2$ possible working sets should be checked). Then, the way of proceeding is checking only several Ws heuristically: one component of W is selected as before (e.g. $i^\star \in \arg\max_{h \in R(x^k)} \{-\nabla_h f(x^k)/a_h\}$); now, only $\mathcal{O}(n)$ Ws are left to be checked to decide $j^\star$.

It has been shown that $j^\star$ solving (42) fixed $i^\star$ is such that

$$j^\star \in \arg\min_t \left\{ -\frac{b_{it}^2}{a_{it}} \middle| \, t \in S(x^k), -\nabla_t f(x^k)/a_t < -\nabla_i f(x^k)/a_i, \right\}$$

where $a_{it} = Q_{ii} + Q_{tt} - 2Q_{it}$ and $b_{it} = \nabla_t f(x^k)/a_t - \nabla_i f(x^k)/a_i > 0$.

In fact, this formula holds assuming the kernel matrix is positive definite; in the positive semi-definite case some corrections would be required, but we will not address them here.

It is important to remark that with this working set selection rule the theoretical convergence properties of `SVM`$^{\texttt{light}}$ continue to hold.

7.3 Algorithms for Linear SVMs

With particularly large datasets, linear classifiers are often the preferable solution, for two main reasons; first, data dimensionality may be so high that mapping points to higher dimensional spaces by the kernel trick may not be necessary; secondly, if the linear kernel is used, special features of the problem can be exploited to make the training process much faster and thus allowing to carry out training with larger amounts of data.

Problem (26) can be equivalently rewritten as

$$\min_w \ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\max\{0, 1 - y^{(i)}w^T x^{(i)}\}; \tag{43}$$

here, the bias term b is not explicitly introduced in the model (we talk about *unbiased formulations*); this is not a restriction from a statistical perspective: bias can be implicitly set into the model by adding a constant feature across all examples in the dataset. However, we will see that this simple change has interesting consequences from an optimization point of view.

A similar problem that can be addressed is the following one

$$\min_w \ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\max\{0, 1 - y^{(i)}w^T x^{(i)}\}^2, \tag{44}$$

where the hinge loss terms are squared.

These two problems share the dual formulation:

$$\begin{aligned} &\min_\alpha \ \frac{1}{2}\alpha^T\bar{Q}\alpha - e^T\alpha \\ &\text{s.t. } 0 \le \alpha \le U, \end{aligned} \tag{45}$$

where $\bar{Q} = Q + D$ and D is a diagonal matrix; for problem (43) we have $U = C$ and $D_{ii} = 0\,\forall i$, while for problem (44) we have $U = \infty$ and $D_{ii} = \frac{1}{2C}$. We can observe that the removal of the bias term leads to a bound constrained dual, without the linear equality constraint.

A *dual coordinate descent* (DCD) algorithm has been proposed to efficiently solve problem (45) when the linear kernel is employed. The algorithm has the following characteristics:

- it is a decomposition method where the variables are updated one at a time:

$$\alpha_i^{k+1} = \arg\min_{\alpha_i \in [0,U]} f(\alpha_1^{k+1}, \ldots, \alpha_{i-1}^{k+1}, \alpha_i, \alpha_{i+1}^{k} \ldots, \alpha_n^{k}),$$

  which can be done in closed form if $\nabla_i f(\alpha_1^{k+1}, \ldots, \alpha_{i-1}^{k+1}, \alpha_i^{k}, \alpha_{i+1}^{k} \ldots, \alpha_n^{k})$ is available;

- variables are updated in cyclic order;

- randomly selecting the variables to be updated experimentally proves to be more efficient;

- solving the subproblems requires the knowledge of $\nabla_i f(\alpha^k)$, which in turn requires the i-th column of $\bar{Q}$;

In Table 5, we report the cost of the main operations of the DCD and the SMO algorithms, in order to carry out a comparison. If we assume the cost of computing a kernel matrix element to be $\mathcal{O}(p)$, we have that SMO spends $\mathcal{O}(np)$ to keep the gradients updated, but then it can smartly select the variables and then solve the subproblems with basically no additional cost; on the other hand, the DCD requires to compute a kernel column at each update; the single iteration of both algorithms approximately has

the same cost, but since SMO selects the variables in a much more effective manner, exploiting first-order information, the number of total iterations is usually much lower for this latter algorithm.

However, in the special case of linear kernels, the relation $w = \sum_{i=1}^{n} y^{(i)} \alpha_i x^{(i)}$ can be exploited; in fact, the following equality holds

$$\nabla_i f(\alpha^k) = y^{(i)} w^T x^{(i)} - 1 + D_{ii} \alpha_i^k,$$

making the variables update cost $\mathcal{O}(p)$, i.e., the iteration cost is much lower than SMO and does not depend on the number of the training samples.

	SMO	**DCD**	**DCD-linear**
variables update	$\mathcal{O}(1)$	$\mathcal{O}(np)$	$\mathcal{O}(p)$
gradients update	$\mathcal{O}(np)$	NA	NA

Table 5: Cost of main operations in SMO and DCD.

The DCD algorithm is thus well suited for large scale linear classification. However, as the size of the problems keeps increasing, this approach may also become computationally unsustainable.

For these cases, a specialized Newton type method has been proposed to directly tackle the continuously differentiable primal problem (44).

These approaches have been implemented in the popular LIBLINEAR library for large scale linear classification.

8 Large Scale Optimization for Deep Models

One of the most relevant optimization problems nowadays is that of *training artificial neural networks*. ANNs are a powerful machine learning model that has been used in a number of applications with impressive results.

The *empirical risk minimization* task of a network with weights $w \in \mathbb{R}^n$ given the data set (X, Y) of N samples takes the form of problem (21), but possesses a number of very peculiar features w.r.t. classical unconstrained nonlinear optimization problem. More in detail:

1. As with other machine learning problems, we are not actually interested in minimizing the empirical risk, which is a surrogate objective function, but rather in finding an effective model in terms of generalization capabilities. In fact, the highly nonconvex training problem typically has plenty of suboptimal stationary points, many of which however with a loss value pretty close to the global optimizer; still, the out-of-sample performance of these solution may dramatically vary from one to another. There is thus no point in seeking the global optimum; rather, good out-of-sample solutions often correspond to local optima that can be reached even with low-precision solvers.

2. Thanks to the *backpropagation* algorithm, the gradients of the loss function $\nabla \mathcal{L}(w)$ can be computed efficiently, to the point that the cost of gradients computation is only twice the cost of evaluating the loss function itself. More details on backpropagation and automatic differentiation are reported in Section 8.2.

3. On the other hand, computing the value of $\mathcal{L}(w)$ is computationally expensive, as the entire data set has to be used to determine all the elements of the sum defining the function.

SGD, and more in general stochastic optimization algorithms, appear to be well suited for ANN training problems for many reasons, which we try to summarize hereafter:

1. Data is often redundant, so using all the available information at every iteration is in fact inefficient.

2. The computational experience gathered by the machine learning community through the years teaches that, if a proper step size α is selected, stochastic methods are indeed much faster than batch ones, especially at the early stages of the optimization process.

3. The rate of convergence to ϵ-optimality of SGD is slower than that of batch GD (ref. Table 3), but is in fact independent from the training set size, which is typically huge in applications. Thus the asymptotic lower cost of GD starts to be observed only for very low values of ϵ. Moreover, recent works showed that in most deep learning problems the loss function satisfy two strong properties:

   - *the interpolation property:* a global optimizer $w^* \in \arg\min_w \mathcal{L}(w)$ of the entire loss is also an optimizer for each individual term, i.e., $w^* \in \arg\min_w \ell_i(w)$ for all $i = 1, \dots, N$. This means that the model is so expressive that it can perfectly fit all data in the dataset;
   - *the PL-condition*: $\mathcal{L}(w) - \mathcal{L}^* \leq \frac{1}{2\mu}\|\mathcal{L}(w)\|^2$ for all $w \in \mathbb{R}^n$; this condition implies that every stationary point is a global optimizer of the problem.

   Under these two assumptions, it is possible to prove that SGD actually has a linear convergence rate, just like full batch GD.

4. Since SGD carries out imprecise descent steps, it is less likely to end up in so-called "sharp minima", i.e., minimizers placed at the bottom of strict holes. Instead, SGD is likely to end up in "flat minimizers". This property results in an intrinsic regularization operation, leading to benefits in terms of generalization properties. See Figure 15 for an intuition.

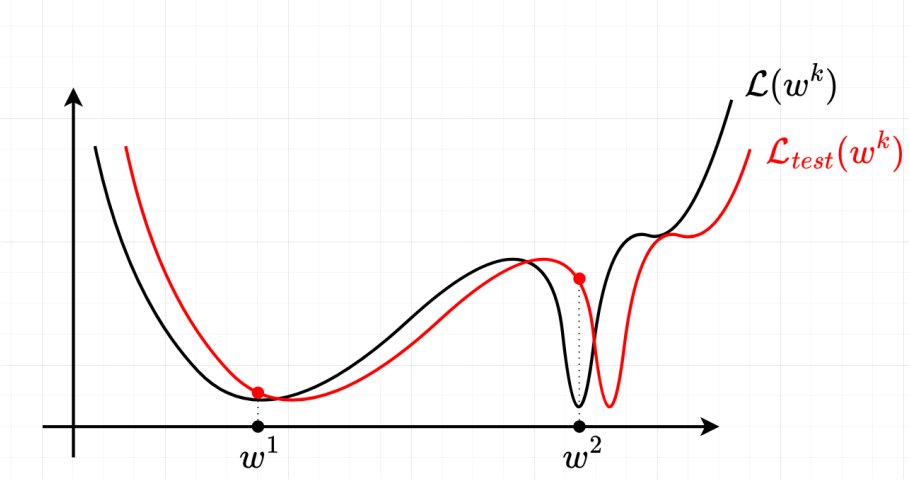


Figure 15: In machine learning, we minimize the training loss as a surrogate of the test loss, which is the real quantity of interest. The test loss will likely be similar, yet not identical, to the training loss. "Sharp" minimizers like w^2 are much more sensitive to the shift from training to test loss than "flat" minimizers like w^1.

Note that, however, batch approaches possess some intrinsic advantages. In particular:

- the use of full gradient information at each iteration opens the door for many deterministic gradient-based optimization methods;

- due to the sum structure of the empirical risk, a batch method, as opposed to SG, can easily benefit from parallelization since the bulk of the computation lies in evaluations of the objective function and gradients;

- if we were to consider a larger number of epochs, then one would see the batch approach eventually overtake the stochastic method and yield a lower training error.

The above comments, both of theoretical and empirical nature, should be convincing about minibatch SGD being the most appropriate approach to solving deep neural nets problem. Choosing $1 < M \ll N$ is a middle way that allows to take advantage of the best features of both stochastic and full batch methods.

8.1 Improvements to SGD for Deep Networks Training

In this section, we describe some modifications to the SGD algorithm that have proven to be practically useful for solving NNs training problems.

8.1.1 Acceleration

Momentum or Nesterov acceleration terms, discussed in Section **??** for the standard unconstrained optimization case, are particularly impactful with stochastic optimization algorithms for deep learning. There are two main reasons:

- Momentum terms are based on averages of the previous steps; these averages somehow filter out strong oscillations due to the stochastic nature of the considered "descent" directions; thus, adding momentum or acceleration terms allows to heavily alleviate the zigzagging behavior that is typical of SGD-type algorithms.
- In the deep learning case, the computation of both the objective function and its gradient is expensive; on the other hand, the information provided by terms $(x^k - x^{k-1})$ is obtainable very cheaply; thus, acceleration can be seen as a way to improve the effectiveness of each iteration without basically any additional cost.

8.1.2 Adaptive stepsizes

Momentum and Nesterov AG exploit information about past iterations to change and improve the quality of search directions. The other road that has been followed (successfully) for the last years with deep learning problems in the field of nonlinear nonconvex stochastic optimization is that of using adaptive learning rates: the stepsize changes at each iteration, based on the progress of the process; moreover, since weights gradients have different dynamics at different layers, each variable is associated with a different step size.

Several strategies to update the learning rate have been proposed; in the following we report the most relevant ones. For the sake of simplicity, we will ignore the fact that in stochastic optimization gradients are computed using only a portion of the objective function. We will describe the upcoming methods with "batch optimization" notation, but they are stochastic methods indeed.

AdaGrad One of the first proposed methods following this idea is *AdaGrad*. In AdaGrad, the running sums s^k of the squares of directional derivatives is stored:

$$s_i^{k+1} = s_i^k + (\nabla_i f(x^k))^2.$$

These quantities are used to scale the gradients, leading to the following update formula

$$x^{k+1} = x^k - \alpha(\mathrm{diag}(s^{k+1}) + \epsilon I)^{-1/2} \nabla f(x^k), \tag{46}$$

where ϵ is a is a smoothing term that avoids division by zero; the update rule can be equivalently written in a component-wise form which is easier to interpret

$$x_i^{k+1} = x_i^k - \frac{\alpha}{\sqrt{s_i^{k+1} + \epsilon}} \nabla_i f(x^k).$$

Note that the components of s are increasing through the iterations. This is ok in principle, as SG theory suggests to diminish learning rates, scaling them with $\mathcal{O}(1/k)$. However, the behavior of AdaGrad is often too aggressive in practice. For this reason, variants have been proposed in the literature, having great impact in the deep learning field.

RMSprop The *RMSprop* (Root Mean Square Propagation) method tries to overcome the limitations of AdaGrad replacing the running sum with an exponentially decaying average of past gradients:

$$s_i^{k+1} = \rho s_i^k + (1-\rho)(\nabla_i f(x^k))^2.$$

Then, parameters are updated by equation (46), exactly as with AdaGrad. With this simple modification, the most recent steps have the largest influence. Also, the sequence s^k is no more divergent, with an advantage in terms of numerical stability.

AdaDelta *AdaDelta* is another variant of AdaGrad, trying again to reduce the aggressive behavior of the latter algorithm. In fact, even though AdaDelta has been developed independently from RMSprop, it can be seen as an extension of the latter. Along with the exponential average of the squared gradients, AdaDelta requires to store the running average of the squared displacements:

$$r^{k+1} = \gamma r^k + (1-\gamma)(x^k - x^{k-1})^2.$$

The update rule of AdaDelta is then

$$x_i^{k+1} = x_i^k - \frac{\sqrt{r_i^{k+1} + \epsilon}}{\sqrt{s_i^{k+1} + \epsilon}} \nabla_i f(x^k).$$

We can note there is no more need to set the base learning rate α.

Adam *Adaptive Moment Estimation*, or simply *Adam*, is another adaptive learning rate algorithm. It can be seen as an evolution of RMSprop and AdaDelta and is at present the most widely employed optimization algorithm for deep networks training.

In addition to the exponentially decaying average of past squared gradients, Adam also keeps, similarly as Momentum, the exponentially decaying average of gradients:

$$\begin{aligned} m_i^{k+1} &= \beta_1 m_i^k + (1-\beta_1)\nabla_i f(x^k), \\ v_i^{k+1} &= \beta_2 v_i^k + (1-\beta_2)(\nabla_i f(x^k))^2 \end{aligned}$$

Vectors m^k and v^k can be seen as estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients, hence the name of the method. These vectors are initialized to 0, generating a bias towards zero which is particularly strong at initial iterations and when the decay rates are small. Indeed:

$$m_i^{k+1} = (1-\beta_1) \sum_{i=0}^{k} \beta_1^{k-i} \nabla f(x^{k-i})$$

and thus

$$E[m_i^{k+1}] = (1-\beta_1)E[\nabla f(x)] \sum_{i=0}^{k} \beta_1^{k-i} = (1-\beta_1)E[\nabla f(x)] \frac{1-\beta_1^k}{1-\beta_1} = E[\nabla f(x)](1-\beta_1^k).$$

This bias can be countered by computing bias-corrected first and second moment estimates:

$$\hat{m}^k = \frac{m^k}{1-(\beta_1)^k} \tag{47}$$

$$\hat{v}^k = \frac{v^k}{1-(\beta_2)^k} \tag{48}$$

Then, the variables are updated by the following update rule:

$$x^{k+1} = x_i^k - \frac{\alpha}{\sqrt{\hat{v}_i^k} + \epsilon}\hat{m}_i^k$$

Some variants of the Adam algorithm have been proposed in recent years. One of them is *AdaMax*, which was proposed along with Adam. AdaMax replaces the update formula (48) for the second momentum with the (empirically) more stable

$$u^{k+1} = \max\{\beta_2 u^k, |\nabla f(x^k)|\},$$

which also doesn't need correction for the initialization bias.

Another popular variant of Adam is *Nadam*, which stands for Nesterov-accelerated Adaptive Moment Estimation. The core idea of Nadam is that of incorporating Nesterov's acceleration into Adam.

8.2 Automatic Differentiation and Backpropagation Algorithm

One of the key issues arising to train neural networks concerns the computation of the gradients of the loss function. Indeed, the objective of the optimization problem is (ignoring the regularization term) the finite sum of functions of (typically) millions of variables, each one being the cascaded composition of elementary functions.

Therefore, numerical differentiation techniques are out of question: the cost is too high (a large number of function evaluations is required to obtain gradient approximation) and they also suffer from significant approximation errors; thus, finite difference approximation is generally used only to check gradients implementation correctness, at low accuracy.

On the other hand, deriving explicit expressions for the gradients is too complex; even the employment of symbolic differentiation tools, i.e., software to manipulate expressions for obtaining expressions of the derivatives, leads to very long and complicated formulae and consequently to massive computations.

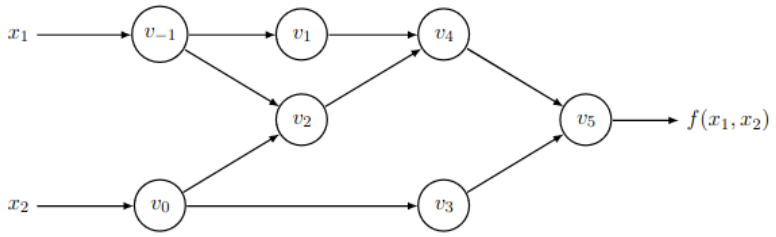
Neural networks training was actually made possible by the development of *Automatic Differentiation* (AD) techniques. By automatic differentiation, we refer to a set of approaches that smartly exploit the multivariate chain rule:

$$f = f(g(x)) \qquad \frac{\partial f}{\partial x_j} = \sum_i \frac{\partial f}{\partial g_i}\frac{\partial g_i}{\partial x_j}.$$

Since evaluating a mathematical function, no matter how complicated, consists in computing out a sequence of elementary operations and functions (exp, log, sin, cos, etc.), by repeatedly applying the chain rule to these operations, partial derivatives can be obtained automatically.

In the context of deep learning, in particular, the AD algorithm typically employed is the famous *backpropagation* algorithm, which is, in technical terms, automatic differentiation in a so called *reverse mode*.

We will not delve deep into the technical details of AD and the backpropagation algorithm. We will just get an idea of its mechanisms by means of a simple example, see Figure 16.



(a) Computation Graph

Forward Primal Trace			Reverse Adjoint (Derivative) Trace		
$v_{-1} = x_1$		$= 2$	$\bar{\boldsymbol{x}}_1 = \bar{\boldsymbol{v}}_{-1}$		$\mathbf{= 5.5}$
$v_0 = x_2$		$= 5$	$\bar{\boldsymbol{x}}_2 = \bar{\boldsymbol{v}}_0$		$\mathbf{= 1.716}$
$v_1 = \ln v_{-1}$		$= \ln 2$	$\bar{v}_{-1} = \bar{v}_{-1} + \bar{v}_1 \frac{\partial v_1}{\partial v_{-1}}$	$= \bar{v}_{-1} + \bar{v}_1 / v_{-1}$	$= 5.5$
$v_2 = v_{-1} \times v_0$		$= 2 \times 5$	$\bar{v}_0 = \bar{v}_0 + \bar{v}_2 \frac{\partial v_2}{\partial v_0}$	$= \bar{v}_0 + \bar{v}_2 \times v_{-1}$	$= 1.716$
			$\bar{v}_{-1} = \bar{v}_2 \frac{\partial v_2}{\partial v_{-1}}$	$= \bar{v}_2 \times v_0$	$= 5$
$v_3 = \sin v_0$		$= \sin 5$	$\bar{v}_0 = \bar{v}_3 \frac{\partial v_3}{\partial v_0}$	$= \bar{v}_3 \times \cos v_0$	$= -0.284$
$v_4 = v_1 + v_2$		$= 0.693 + 10$	$\bar{v}_2 = \bar{v}_4 \frac{\partial v_4}{\partial v_2}$	$= \bar{v}_4 \times 1$	$= 1$
			$\bar{v}_1 = \bar{v}_4 \frac{\partial v_4}{\partial v_1}$	$= \bar{v}_4 \times 1$	$= 1$
$v_5 = v_4 - v_3$		$= 10.693 + 0.959$	$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3}$	$= \bar{v}_5 \times (-1)$	$= -1$
			$\bar{v}_4 = \bar{v}_5 \frac{\partial v_5}{\partial v_4}$	$= \bar{v}_5 \times 1$	$= 1$
$y = v_5$		$= 11.652$	$\bar{v}_5 = \bar{y}$	$= 1$	

(b) Reverse AD

Figure 16: Automatic differentiation (reverse mode) example; $f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$. We use the notation $\bar{v}_t = \partial y / \partial v_t$. Credits: `https://www.jmlr.org/papers/volume18/17-468/17-468.pdf`

First, all elementary operations are mapped into a *Computation Graph* (Figure 16a), which is a structure that allows to link quantities that depend one from the other. This way, computed quantities that will be needed to compute other terms can be stored in memory, avoiding duplicate computations.

The actual computation first proceeds by feeding the inputs to the function and by computing intermediate products up until the function output; then, the graph is traversed backward, to compute all the partial derivatives. We shall note that, in order to compute the gradients, we obtain as an intermediate side product the function value; thus, in a gradient descent iteration, we can obtain the function value at the current point, by carrying out a forward pass through the computation graph, and then the gradients, after a backward pass through the graph.

As aforementioned, some terms appear multiple times throughout computation; in order to avoid duplicate calculations, we can store these values to reuse them when needed; of course, it shall not be hard to imagine that in deep and complex networks terms repeat in a much more massive manner than in the simple example at hand.