Mathematical Foundations of Data Sciences

Gabriel Peyré
CNRS & DMA
École Normale Supérieure
gabriel.peyre@ens.fr
https://mathematical-tours.github.io
www.numerical-tours.com

March 22, 2020

Chapter 1

Gradient Descent Methods

This chapter studies first order method for smooth unconstrained optimization, which are the most standard methods for machine learning.

1.1 Motivation in Machine Learning

Unconstraint optimization In most part of this Chapter, we consider unconstrained convex optimization problems of the form

$$\inf_{x \in \mathbb{R}^p} f(x), \tag{1.1}$$

and try to devise "cheap" algorithms with a low computational cost per iteration to approximate a minimizer when it exists. The class of algorithms considered are first order, i.e. they make use of gradient information. In the following, we denote

$$\operatorname*{argmin}_x f(x) \stackrel{\text{def.}}{=} \{x \in \mathbb{R}^p \;;\; f(x) = \inf f\},$$

to indicate the set of points (it is not necessarily a singleton since the minimizer might be non-unique) that achieve the minimum of the function f. One might have $\operatorname{argmin} f = \emptyset$ (this situation is discussed below), but in case a minimizer exists, we denote the optimization problem as

$$\min_{x \in \mathbb{R}^p} f(x). \tag{1.2}$$

In typical learning scenario, $f(x)$ is the empirical risk for regression or classification, and p is the number of parameter. For instance, in the simplest case of linear models, we denote $(a_i, y_i)_{i=1}^n$ where $a_i \in \mathbb{R}^p$ are the features. In the following, we denote $A \in \mathbb{R}^{n \times p}$ the matrix whose rows are the a_i.

Example 1 (Regression). For regression, $y_i \in \mathbb{R}$, in which case

$$f(x) = \frac{1}{2} \sum_{i=1}^n (y_i - \langle x, a_i \rangle)^2 = \frac{1}{2} \|Ax - y\|^2, \tag{1.3}$$

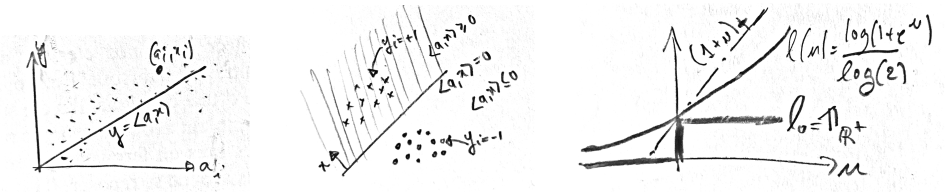


Figure 1.1: Left: linear regression, middle: linear classifier, right: loss function for classification.

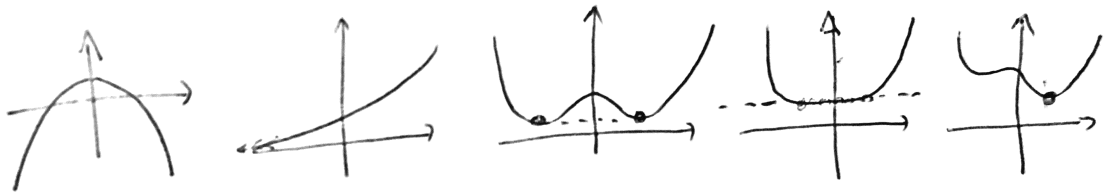


Figure 1.2: Left: non-existence of minimizer, middle: multiple minimizers, right: uniqueness.

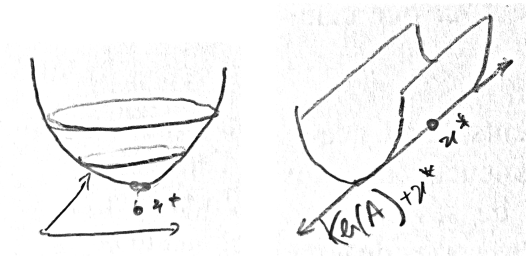


Figure 1.3: Coercivity condition for least squares.

is the least square quadratic risk function (see Fig. 1.2). Here $\langle u,\, v\rangle = \sum_{i=1}^p u_i v_i$ is the canonical inner product in $\mathbb{R}^p$ and $\|\cdot\|^2 = \langle \cdot,\, \cdot\rangle$.

Example 2 (Classification). For classification, $y_i \in \{-1, 1\}$, in which case

$$f(x) = \sum_{i=1}^n \ell(-y_i \langle x,\, a_i\rangle) = L(-\operatorname{diag}(y)Ax) \tag{1.4}$$

where ℓ is a smooth approximation of the 0-1 loss $1_{\mathbb{R}^+}$. For instance $\ell(u) = \log(1 + \exp(u))$, and $\operatorname{diag}(y) \in \mathbb{R}^{n\times n}$ is the diagonal matrix with y_i along the diagonal (see Fig. 1.2, right). Here the separable loss function $L = \mathbb{R}^n \to \mathbb{R}$ is, for $z \in \mathbb{R}^n$, $L(z) = \sum_i \ell(z_i)$.

Coercivity. In general, there might be no solution to the optimization (1.1). This is of course the case if f is unbounded by bellow, for instance $f(x) = -x^2$ in which case the value of the minimum is $-\infty$. But this might also happen if f does not grow at infinity, for instance $f(x) = e^{-x}$, for which $\min f = 0$ but there is no minimizer.

In order to show existence of a minimizer, and that the set of minimizer is bounded (otherwise one can have problems with optimization algorithm that could escape to infinity), one needs to show that one can replace the whole space $\mathbb{R}^p$ by a compact sub-set $\Omega \subset \mathbb{R}^p$ (i.e. Ω is bounded and close) and that f is continuous on Ω (one can replace this by a weaker condition, that f is lower-semi-continuous, but we ignore this here). A way to show that one can consider only a bounded set is to show that $f(x) \to +\infty$ when $x \to +\infty$. Such a function is called coercive. In this case, one can choose any $x_0 \in \mathbb{R}^p$ and consider its associated lower-level set

$$\Omega = \{x \in \mathbb{R}^p \;;\; f(x) \leqslant f(x_0)\}$$

which is bounded because of coercivity, and closed because f is continuous. One can actually show that for convex function, having a bounded set of minimizer is equivalent to the function being coercive (this is not the case for non-convex function, for instance $f(x) = \min(1, x^2)$ has a single minimum but is not coercive).

Example 3 (Least squares). For instance, for the quadratic loss function $f(x) = \frac{1}{2}\|Ax - y\|^2$, coercivity holds if and only if $\ker(A) = \{0\}$ (this corresponds to the overdetermined setting). Indeed, if $\ker(A) \neq \{0\}$ if $x^\star$ is a solution, then $x^\star + u$ is also solution for any $u \in \ker(A)$, so that the set of minimizer is unbounded. On contrary, if $\ker(A) = \{0\}$, we will show later that the set of minimizer is unique, see Fig. 1.3. If ℓ is strictly convex, the same conclusion holds in the case of classification.

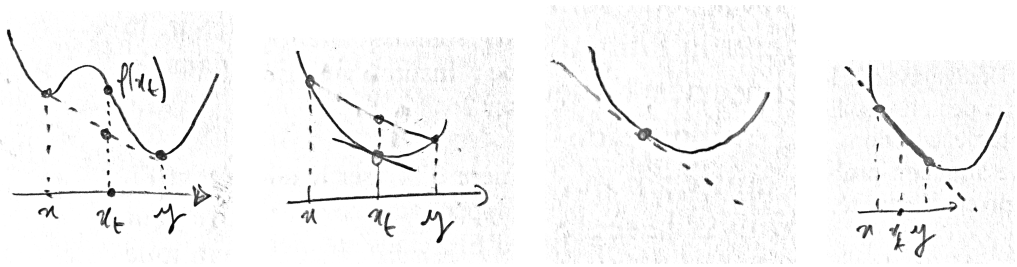


Figure 1.4: Convex vs. non-convex functions ; Strictly convex vs. non strictly convex functions.

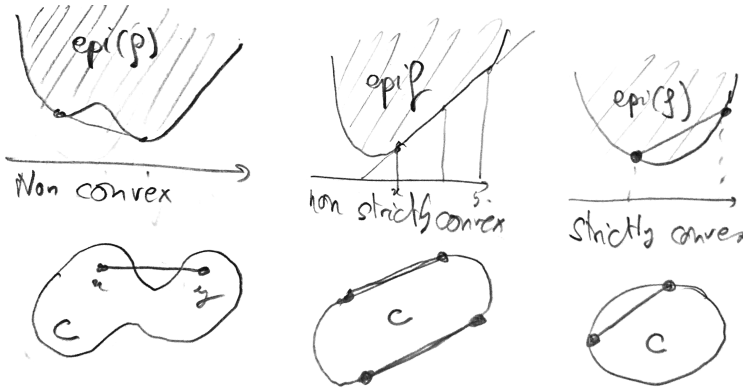


Figure 1.5: Comparison of convex functions $f : \mathbb{R}^p \to \mathbb{R}$ (for $p = 1$) and convex sets $C \subset \mathbb{R}^p$ (for $p = 2$).

Convexity. Convex functions define the main class of functions which are somehow "simple" to optimize, in the sense that all minimizers are global minimizers, and that there are often efficient methods to find these minimizers (at least for smooth convex functions). A convex function is such that for any pair of point $(x, y) \in (\mathbb{R}^p)^2$,

$$\forall\, t \in [0,1], \quad f((1-t)x + ty) \leqslant (1-t)f(x) + tf(y) \tag{1.5}$$

which means that the function is bellow its secant (and actually also above its tangent when this is well defined), see Fig. 1.4. If $x^\star$ is a local minimizer of a convex f, then $x^\star$ is a global minimizer, i.e. $x^\star \in \operatorname{argmin} f$.

Convex function are very convenient because they are stable under lots of transformation. In particular, if f, g are convex and a, b are positive, $af + bg$ is convex (the set of convex function is itself an infinite dimensional convex cone!) and so is $\max(f, g)$. If $g : \mathbb{R}^q \to \mathbb{R}$ is convex and $B \in \mathbb{R}^{q\times p}, b \in \mathbb{R}^q$ then $f(x) = g(Bx + b)$ is convex. This shows immediately that the square loss appearing in (1.3) is convex, since $\|\cdot\|^2/2$ is convex (as a sum of squares). Also, similarly, if ℓ and hence L is convex, then the classification loss function (1.4) is itself convex.

Remark 1 (Convexity of the set of minimizers). In general, minimizers $x^\star$ might be non-unique, as shown on Figure 1.3. When f is convex, the set $\operatorname{argmin}(f)$ of minimizers is itself a convex set. We recall that $\Omega \subset \mathbb{R}^p$ is said to be convex if for any $(x, y) \in \Omega^2$, $(1-t)x+ty \in \Omega$ for $t \in [0, 1]$. Indeed, if $x_1^\star$ and $x_2^\star$ are minimizers, so that in particular $f(x_1^\star) = f(x_2^\star) = \min(f)$, then $f((1-t)x_1^\star + tx_2^\star) \leqslant (1-t)f(x_1^\star) + tf(x_2^\star) = f(x_1^\star) = \min(f)$, so that $(1-t)x_1^\star + tx_2^\star$ is itself a minimizer. Figure 1.5 shows convex and non-convex sets. The connexion between convex function and convex sets is that a function f is convex if and only if its epigraph $\operatorname{epi}(f) \overset{\text{def.}}{=} \big\{(x, t) \in \mathbb{R}^{p+1} \;;\; t \geqslant f(x)\big\}$ is a convex set.

When f is convex, one can strengthen the condition (1.5) and impose that the inequality is strict for $t \in]0, 1[$ (see Fig. 1.4, right), i.e.

$$\forall\, t \in]0,1[, \quad f((1-t)x + ty) < (1-t)f(x) + tf(y). \tag{1.6}$$

In this case, if a minimum $x^\star$ exists, then it is unique. Indeed, if $x_1^\star \neq x_2^\star$ were two different minimizer, one would have by strict convexity $f(\frac{x_1^\star + x_2^\star}{2}) < f(x_1^\star)$ which is impossible.

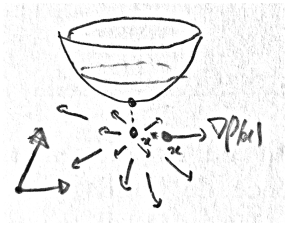
Example 4 (Least squares). For the quadratic loss function $f(x) = \frac{1}{2}\|Ax - y\|^2$, strict convexity is equivalent to $\ker(A) = \{0\}$. Indeed, we see later that its second derivative is $\partial^2 f(x) = A^\top A$ and that strict convexity is implied by the eigenvalues of $A^\top A$ being strictly positive. The eigenvalues of $A^\top A$ being positive, it is equivalent to $\ker(A^\top A) = \{0\}$ (no vanishing eigenvalue), and $A^\top A z = 0$ implies $\langle A^\top A z,\, z\rangle = \|Az\|^2 = 0$ i.e. $z \in \ker(A)$.

1.2 Derivative and gradient

Gradient. If f is differentiable along each axis, we denote

$$\nabla f(x) \stackrel{\text{def.}}{=} (\partial_{x_1} f(x), \ldots, \partial_{x_p} f(x))^\top \in \mathbb{R}^p$$

the gradient vector, so that $\nabla f : \mathbb{R}^p \to \mathbb{R}^p$ is a vector field.



Beware that $\nabla f(x)$ can exist without f being differentiable. Differentiability of f at each reads

$$f(x+\varepsilon) = f(x) + \langle \varepsilon,\, \nabla f(x)\rangle + o(\|\varepsilon\|). \tag{1.7}$$

Here $R(\varepsilon) = o(\|\varepsilon\|)$ denotes a quantity which decays faster than ε toward 0, i.e. $\frac{R(\varepsilon)}{\|\varepsilon\|} \to 0$ as $\varepsilon \to 0$. Existence of partial derivative corresponds to f being differentiable along the axes, while differentiability should hold for any converging sequence of $\varepsilon \to 0$ (i.e. not along along a fixed direction).

Also, $\nabla f(x)$ is the only vector such that the relation (1.7). This means that a possible strategy to both prove that f is differentiable and to obtain a formula for $\nabla f(x)$ is to show a relation of the form

$$f(x+\varepsilon) = f(x) + \langle \varepsilon,\, g\rangle + o(\|\varepsilon\|),$$

in which case one necessarily has $\nabla f(x) = g$.

The following proposition shows that convexity is equivalent to the graph of the function being above its tangents.

Proposition 1. *If f is differentiable, then*

$$f \text{ convex} \quad \Leftrightarrow \quad \forall (x, x'),\ f(x) \geqslant f(x') + \langle \nabla f(x'),\, x - x'\rangle.$$

Proof. One can write the convexity condition as

$$f((1-t)x + tx') \leqslant (1-t)f(x) + tf(x') \quad \Longrightarrow \quad \frac{f(x + t(x'-x)) - f(x)}{t} \leqslant f(x') - f(x)$$

hence, taking the limit $t \to 0$ one obtains

$$\langle \nabla f(x),\, x' - x\rangle \leqslant f(x') - f(x).$$

For the other implication, we apply the right condition replacing (x, x') by $(x, x_t \stackrel{\text{def.}}{=} (1-t)x + tx')$ and $(x', (1-t)x + tx')$

$$\begin{aligned} f(x) &\geqslant f(x_t) + \langle \nabla f(x_t),\, x - x_t\rangle = f(x_t) - t\langle \nabla f(x_t),\, x - x'\rangle \\ f(x') &\geqslant f(x_t) + \langle \nabla f(x_t),\, x' - x_t\rangle = f(x_t) + (1-t)\langle \nabla f(x_t),\, x - x'\rangle, \end{aligned}$$

multiplying these inequality by respectively $1-t$ and t, and summing them, gives

$$(1-t)f(x) + tf(x') \geqslant f(x_t).$$

□

Figure 1.6: Function with local maxima/minima (left), saddle point (middle) and global minimum (right).

First order condition. The main theoretical interest (we will see later that it also have algorithmic interest) of the gradient vector is that it is a necessarily condition for optimality, as stated bellow.

Proposition 2. *If $x^\star$ is a local minimum of the function f (i.e. that $f(x^\star) \leqslant f(x)$ for all x in some ball around $x^\star$) then*

$$\nabla f(x^\star) = 0.$$

Proof. One has for ε small enough and u fixed

$$f(x^\star) \leqslant f(x^\star + \varepsilon u) = f(x^\star) + \varepsilon \langle \nabla f(x^\star), u \rangle + o(\varepsilon) \quad \Longrightarrow \quad \langle \nabla f(x^\star), u \rangle \geqslant o(1) \quad \Longrightarrow \quad \langle \nabla f(x^\star), u \rangle \geqslant 0.$$

So applying this for u and $-u$ in the previous equation shows that $\langle \nabla f(x^\star), u \rangle = 0$ for all u, and hence $\nabla f(x^\star) = 0$. □

Note that the converse is not true in general, since one might have $\nabla f(x) = 0$ but x is not a local mininimum. For instance $x = 0$ for $f(x) = -x^2$ (here x is a maximizer) or $f(x) = x^3$ (here x is neither a maximizer or a minimizer, it is a saddle point), see Fig. 1.6. Note however that in practice, if $\nabla f(x^\star) = 0$ but x is not a local minimum, then $x^\star$ tends to be an unstable equilibrium. Thus most often a gradient-based algorithm will converge to points with $\nabla f(x^\star) = 0$ that are local minimizers. The following proposition shows that a much strong result holds if f is convex.

Proposition 3. *If f is convex and $x^\star$ a local minimum, then $x^\star$ is also a global minimum. If f is differentiable and convex,*

$$x^\star \in \operatorname*{argmin}_x f(x) \quad \Longleftrightarrow \quad \nabla f(x^\star) = 0.$$

Proof. For any x, there exist $0 < t < 1$ small enough such that $tx + (1-t)x^\star$ is close enough to $x^\star$, and so since it is a local minimizer

$$f(x^\star) \leqslant f(tx + (1-t)x^\star) \leqslant tf(x) + (1-t)f(x^\star) \quad \Longrightarrow \quad f(x^\star) \leqslant f(x)$$

and thus $x^\star$ is a global minimum.

For the second part, we already saw in (2) the $\Leftarrow$ part. We assume that $\nabla f(x^\star) = 0$. Since the graph of x is above its tangent by convexity (as stated in Proposition 1),

$$f(x) \geqslant f(x^\star) + \langle \nabla f(x^\star), x - x^\star \rangle = f(x^\star).$$

□

Thus in this case, optimizing a function is the same a solving an equation $\nabla f(x) = 0$ (actually p equations in p unknown). In most case it is impossible to solve this equation, but it often provides interesting information about solutions $x^\star$.

Example: least squares. The most important gradient formula is the one of the square loss (1.3), which can be obtained by expanding the norm

$$\begin{aligned} f(x+\varepsilon) &= \frac{1}{2}\|Ax-y+A\varepsilon\|^2 = \frac{1}{2}\|Ax-y\| + \langle Ax-y,\, A\varepsilon\rangle + \frac{1}{2}\|A\varepsilon\|^2 \\ &= f(x) + \langle \varepsilon,\, A^\top(Ax-y)\rangle + o(\|\varepsilon\|). \end{aligned}$$

Here, we have used the fact that $\|A\varepsilon\|^2 = o(\|\varepsilon\|)$ and use the transpose matrix $A^\top$. This matrix is obtained by exchanging the rows and the columns, i.e. $A^\top = (A_{j,i})_{i=1,\dots,n}^{j=1,\dots,p}$, but the way it should be remember and used is that it obeys the following swapping rule of the inner product,

$$\forall\, (u,v) \in \mathbb{R}^p \times \mathbb{R}^n, \quad \langle Au,\, v\rangle_{\mathbb{R}^n} = \langle u,\, A^\top v\rangle_{\mathbb{R}^p}.$$

Computing gradient for function involving linear operator will necessarily requires such a transposition step. This computation shows that

$$\nabla f(x) = A^\top(Ax-y).$$

This implies that solutions $x^\star$ minimizing $f(x)$ satisfies the linear system $(A^\top A)x^\star = A^\top y$. If $A^\star A \in \mathbb{R}^{p\times p}$ is invertible, then f has a single minimizer, namely

$$x^\star = (A^\top A)^{-1}A^\top y. \tag{1.8}$$

This shows that in this case, $x^\star$ depends linearly on the data y, and the corresponding linear operator $(A^\top A)^{-1}A^\star$ is often called the Moore-Penrose pseudo-inverse of A (which is not invertible in general, since typically $p \neq n$). The condition that $A^\top A$ is invertible is equivalent to $\ker(A) = \{0\}$, since

$$A^\top Ax = 0 \quad \Longrightarrow \quad \|Ax\|^2 = \langle A^\top Ax,\, x\rangle = 0 \quad \Longrightarrow \quad Ax = 0.$$

In particular, if $n < p$ (under-determined regime, there is too much parameter or too few data) this can never holds. If $n \geqslant p$ and the features x_i are "random" then $\ker(A) = \{0\}$ with probability one. In this overdetermined situation $n \geqslant p$, $\ker(A) = \{0\}$ only holds if the features $\{a_i\}_{i=1}^n$ spans a linear space $\mathrm{Im}(A^\top)$ of dimension strictly smaller than the ambient dimension p.

Link with PCA. Let us assume the $(a_i)_{i=1}^n$ are centered, i.e. $\sum_i a_i = 0$. If this is not the case, one needs to replace a_i by $a_i - m$ where $m \overset{\text{def.}}{=} \frac{1}{n}\sum_{i=1}^n a_i \in \mathbb{R}^p$ is the empirical mean. In this case, $\frac{C}{n} = A^\top A/n \in \mathbb{R}^{p\times p}$ is the empirical covariance of the point cloud $(a_i)_i$, it encodes the covariances between the coordinates of the points. Denoting $a_i = (a_{i,1},\dots,a_{i,p})^\top \in \mathbb{R}^p$ (so that $A = (a_{i,j})_{i,j}$) the coordinates, one has

$$\forall\, (k,\ell) \in \{1,\dots,p\}^2, \quad \frac{C_{k,\ell}}{n} = \frac{1}{n}\sum_{i=1}^n a_{i,k}a_{i,\ell}.$$

In particular, $C_{k,k}/n$ is the variance along the axis k. More generally, for any unit vector $u \in \mathbb{R}^p$, $\langle Cu,\, u\rangle/n \geqslant 0$ is the variance along the axis u.

For instance, in dimension $p = 2$,

$$\frac{C}{n} = \frac{1}{n}\begin{pmatrix} \sum_{i=1}^n a_{i,1}^2 & \sum_{i=1}^n a_{i,1}a_{i,2} \\ \sum_{i=1}^n a_{i,1}a_{i,2} & \sum_{i=1}^n a_{i,2}^2 \end{pmatrix}.$$

Since C is a symmetric, it diagonalizes in an ortho-basis $U = (u_1,\dots,u_p) \in \mathbb{R}^{p\times p}$. Here, the vectors $u_k \in \mathbb{R}^p$ are stored in the columns of the matrix U. The diagonalization means that there exist scalars (the eigenvalues) $(\lambda_1,\dots,\lambda_p)$ so that $(\frac{1}{n}C)u_k = \lambda_k u_k$. Since the matrix is orthogononal, $UU^\top = U^\top U = \mathrm{Id}_p$, and equivalently $U^{-1} = U^\top$. The diagonalization property can be conveniently written as $\frac{1}{n}C = U\,\mathrm{diag}(\lambda_k)U^\top$. One can thus re-write the covariance quadratic form in the basis U as being a separable sum of p squares

$$\frac{1}{n}\langle Cx,\, x\rangle = \langle U\,\mathrm{diag}(\lambda_k)U^\top x,\, x\rangle = \langle \mathrm{diag}(\lambda_k)(U^\top x),\, (U^\top x)\rangle = \sum_{k=1}^p \lambda_k \langle x,\, u_k\rangle^2. \tag{1.9}$$

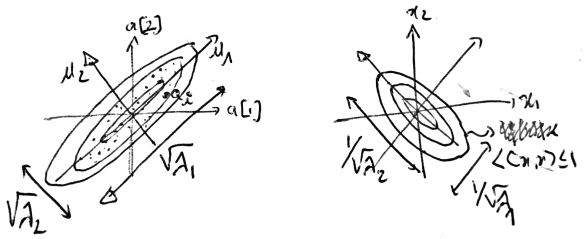


Figure 1.7: Left: point clouds $(a_i)_i$ with associated PCA directions, right: quadratic part of $f(x)$.

Here $(U^\top x)_k = \langle x, u_k \rangle$ is the coordinate k of x in the basis U. Since $\langle Cx, x \rangle = \|Ax\|^2$, this shows that all the eigenvalues $\lambda_k \geqslant 0$ are positive.

If one assumes that the eigenvalues are ordered $\lambda_1 \geqslant \lambda_2 \geqslant \ldots \geqslant \lambda_p$, then projecting the points a_i on the first m eigenvectors can be shown to be in some sense the best linear dimensionality reduction possible, and it is called Principal Component Analysis (PCA). It is useful to perform compression or dimensionality reduction, but in practice, it is mostly used for data visualization in 2-D ($m = 2$) and 3-D ($m = 3$).

The matrix C/n encodes the covariance, so one can approximate the point cloud by an ellipsoid whose main axes are the $(u_k)_k$ and the width along each axis is $\propto \sqrt{\lambda_k}$ (the standard deviations). If the data are approximately drawn from a Gaussian distribution, whose density is proportional to $\exp(\frac{-1}{2}\langle C^{-1}a, a \rangle)$, then the fit is good. This should be contrasted with the shape of quadratic part $\frac{1}{2}\langle Cx, x \rangle$ of $f(x)$, since the ellipsoid $\left\{x \;;\; \frac{1}{n}\langle Cx, x \rangle \leqslant 1\right\}$ has the same main axes, but the widths are the inverse $1/\sqrt{\lambda_k}$. Figure 1.7 shows this in dimension $p = 2$.

Example: Classification We can do a similar computation for the gradient of the classification loss (1.4). Assuming that L is differentiable, and using the Taylor expansion (1.7) at point $-\operatorname{diag}(y)Ax$, one has

$$\begin{aligned} f(x+\varepsilon) &= L(-\operatorname{diag}(y)Ax - \operatorname{diag}(y)A\varepsilon) \\ &= L(-\operatorname{diag}(y)Ax) + \langle \nabla L(-\operatorname{diag}(y)Ax), -\operatorname{diag}(y)A\varepsilon \rangle + o(\|\operatorname{diag}(y)A\varepsilon\|) \end{aligned}$$

Using the fact that $o(\|\operatorname{diag}(y)A\varepsilon\|) = o(\|\varepsilon\|)$, one obtains

$$\begin{aligned} f(x+\varepsilon) &= f(x) + \langle \nabla L(-\operatorname{diag}(y)Ax), -\operatorname{diag}(y)A\varepsilon \rangle + o(\|\varepsilon\|) \\ &= f(x) + \langle -A^\top \operatorname{diag}(y) \nabla L(-\operatorname{diag}(y)Ax), \varepsilon \rangle + o(\|\varepsilon\|), \end{aligned}$$

where we have used the fact that $(AB)^\top = B^\top A^\top$ and that $\operatorname{diag}(y)^\top = \operatorname{diag}(y)$. This shows that

$$\nabla f(x) = -A^\top \operatorname{diag}(y) \nabla L(-\operatorname{diag}(y)Ax).$$

Since $L(z) = \sum_i \ell(z_i)$, one has $\nabla L(z) = (\ell'(z_i))_{i=1}^n$. For instance, for the logistic classification method, $\ell(u) = \log(1 + \exp(u))$ so that $\ell'(u) = \frac{e^u}{1+e^u} \in [0, 1]$ (which can be interpreted as a probability of predicting $+1$).

Chain rule. One can formalize the previous computation, if $f(x) = g(Bx)$ with $B \in \mathbb{R}^{q \times p}$ and $g : \mathbb{R}^q \to \mathbb{R}$, then

$$f(x+\varepsilon) = g(Bx + B\varepsilon) = g(Bx) + \langle \nabla g(Bx), B\varepsilon \rangle + o(\|B\varepsilon\|) = f(x) + \langle \varepsilon, B^\top \nabla g(Bx) \rangle + o(\|\varepsilon\|),$$

which shows that

$$\nabla(g \circ B) = B^\top \circ \nabla g \circ B \tag{1.10}$$

where "$\circ$" denotes the composition of functions.

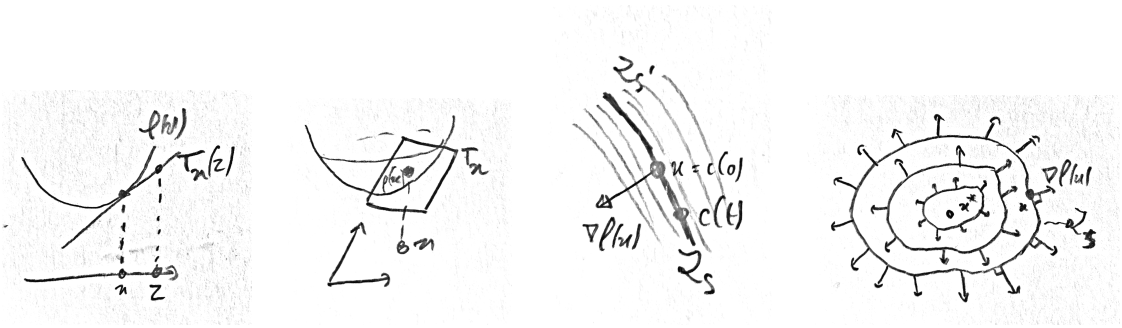


Figure 1.8: Left: First order Taylor expansion in 1-D and 2-D. Right: orthogonality of gradient and level sets and schematic of the proof.

To generalize this to composition of possibly non-linear functions, one needs to use the notion of differential. For a function $F : \mathbb{R}^p \to \mathbb{R}^q$, its differentiable at x is a linear operator $\partial F(x) : \mathbb{R}^p \to \mathbb{R}^q$, i.e. it can be represented as a matrix (still denoted $\partial F(x)$) $\partial F(x) \in \mathbb{R}^{q \times p}$. The entries of this matrix are the partial differential, denoting $F(x) = (F_1(x), \ldots, F_q(x))$,

$$\forall\, (i,j) \in \{1,\ldots,q\} \times \{1,\ldots,p\}, \quad [\partial F(x)]_{i,j} \stackrel{\text{def.}}{=} \frac{\partial F_i(x)}{\partial x_j}.$$

The function F is then said to be differentiable at x if and only if one has the following Taylor expansion

$$F(x+\varepsilon) = F(x) + [\partial F(x)](\varepsilon) + o(\|\varepsilon\|). \tag{1.11}$$

where $[\partial F(x)](\varepsilon)$ is the matrix-vector multiplication. As for the definition of the gradient, this matrix is the only one that satisfies this expansion, so it can be used as a way to compute this differential in practice.

For the special case $q = 1$, i.e. if $f : \mathbb{R}^p \to \mathbb{R}$, then the differential $\partial f(x) \in \mathbb{R}^{1 \times p}$ and the gradient $\nabla f(x) \in \mathbb{R}^{p \times 1}$ are linked by equating the Taylor expansions (1.11) and (1.7)

$$\forall\, \varepsilon \in \mathbb{R}^p, \quad [\partial f(x)](\varepsilon) = \langle \nabla f(x), \varepsilon \rangle \quad \Leftrightarrow \quad [\partial f(x)](\varepsilon) = \nabla f(x)^\top.$$

The differential satisfies the following chain rule

$$\partial(G \circ H)(x) = [\partial G(H(x))] \times [\partial H(x)]$$

where "$\times$" is the matrix product. For instance, if $H : \mathbb{R}^p \to \mathbb{R}^q$ and $G = g : \mathbb{R}^q \mapsto \mathbb{R}$, then $f = g \circ H : \mathbb{R}^p \to \mathbb{R}$ and one can compute its gradient as follow

$$\nabla f(x) = (\partial f(x))^\top = ([\partial g(H(x))] \times [\partial H(x)])^\top = [\partial H(x)]^\top \times [\partial g(H(x))]^\top = [\partial H(x)]^\top \times \nabla g(H(x)).$$

When $H(x) = Bx$ is linear, one recovers formula (1.10).

1.3 Gradient Descent

Steepest descent direction. The Taylor expansion (1.7) computes an affine approximation of the function f near x, since it can be written as

$$f(z) = T_x(z) + o(\|x - z\|) \quad \text{where} \quad T_x(z) \stackrel{\text{def.}}{=} f(x) + \langle \nabla f(x), z - x \rangle,$$

see Fig. 1.8. First order methods operate by locally replacing f by T_x.

The gradient $\nabla f(x)$ should be understood as a direction along which the function increases. This means that to improve the value of the function, one should move in the direction $-\nabla f(x)$. Given some fixed x, let us look as the function f along the 1-D half line

$$\tau \in \mathbb{R}^+ = [0, +\infty[\longmapsto f(x - \tau \nabla f(x)) \in \mathbb{R}.$$

If f is differentiable at x, one has

$$f(x - \tau \nabla f(x)) = f(x) - \tau \langle \nabla f(x), \nabla f(x) \rangle + o(\tau) = f(x) - \tau \|\nabla f(x)\|^2 + o(\tau).$$

So there are two possibility: either $\nabla f(x) = 0$, in which case we are already at a minimum (possibly a local minimizer if the function is non-convex) or if τ is chosen small enough,

$$f(x - \tau \nabla f(x)) < f(x)$$

which means that moving from x to $x - \tau \nabla f(x)$ has improved the objective function.

Remark 2 (Orthogonality to level sets). The level sets of f are the set of point sharing the same value of f, i.e. for any $s \in \mathbb{R}$

$$\mathcal{L}_s \stackrel{\text{def.}}{=} \{x \; ; \; f(x) = s\}.$$

At some $x \in \mathbb{R}^p$, denoting $s = f(x)$, then $x \in \mathcal{L}_s$ (x belong to its level set). The gradient vector $\nabla f(x)$ is orthogonal to the level set (as shown on Fig. 1.8 right), and points toward level set of higher value (which is consistent with the previous computation showing that it is a valid ascent direction). Indeed, lets consider around x inside $\mathcal{L}_s$ a smooth curve of the form $t \in \mathbb{R} \mapsto c(t)$ where $c(0) = x$. Then the function $h(t) \stackrel{\text{def.}}{=} f(c(t))$ is constant $h(t) = s$ since $c(t)$ belong to the level set. So $h'(t) = 0$. But at the same time, we can compute its derivate at $t = 0$ as follow

$$h(t) = f(c(0) + tc'(0) + o(t)) = h(0) + \delta \langle c'(0), \nabla f(c(0)) \rangle + o(t)$$

i.e. $h'(0) = \langle c'(0), \nabla f(x) \rangle = 0$, so that $\nabla f(x)$ is orthogonal to the tangent $c'(0)$ of the curve c, which lies in the tangent plane of $\mathcal{L}_s$ (as shown on Fig. 1.8, right). Since the curve c is arbitrary, the whole tangent plane is thus orthogonal to $\nabla f(x)$.

Remark 3 (Local optimal descent direction). One can prove something even stronger, that among all possible direction u with $\|u\| = r$, $r \frac{\nabla f(x)}{\|\nabla f(x)\|}$ becomes the optimal one as $r \to 0$ (so for very small step this is locally the best choice), more precisely,

$$\frac{1}{r} \operatorname*{argmin}_{\|u\|=r} f(x+u) \stackrel{r \to 0}{\longrightarrow} \frac{\nabla f(x)}{\|\nabla f(x)\|}.$$

Gradient descent algorithm The gradient descent algorithm reads, starting with some $x_0 \in \mathbb{R}^p$

$$x_{k+1} \stackrel{\text{def.}}{=} x_k - \tau_k \nabla f(x_k) \tag{1.12}$$

where $\tau_k > 0$ is the step size (also called learning rate). For a small enough τ_k, the previous discussion shows that the function f is decaying through the iteration. So intuitively, to ensure convergence, τ_k should be chosen small enough, but not too small so that the algorithm is as fast as possible. In general, one use a fix step size $\tau_k = \tau$, or try to adapt τ_k at each iteration (see Fig. 1.9).

Remark 4 (Greedy choice). Although this is in general too costly to perform exactly, one can use a "greedy" choice, where the step size is optimal at each iteration, i.e.

$$\tau_k \stackrel{\text{def.}}{=} \operatorname*{argmin}_{\tau} h(\tau) \stackrel{\text{def.}}{=} f(x_k - \tau \nabla f(x_k)).$$

Here $h(\tau)$ is a function of a single variable. One can compute the derivative of h as

$$h(\tau + \delta) = f(x_k - \tau \nabla f(x_k) - \delta \nabla f(x_k)) = f(x_k - \tau \nabla f(x_k)) - \langle \nabla f(x_k - \tau \nabla f(x_k)), \nabla f(x_k) \rangle + o(\delta).$$

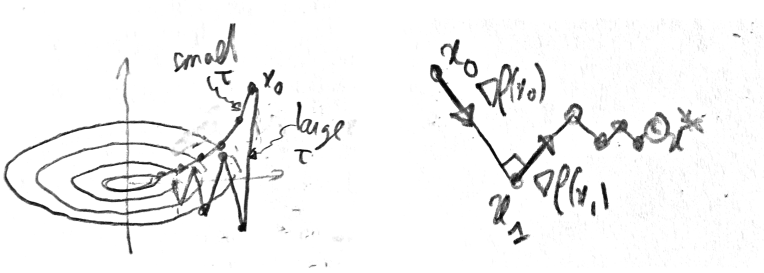


Figure 1.9: Influence of τ on the gradient descent (left) and optimal step size choice (right).

One note that at $\tau = \tau_k$, $\nabla f(x_k - \tau \nabla f(x_k)) = \nabla f(x_{k+1})$ by definition of x_{k+1} in (1.12). Such an optimal $\tau = \tau_k$ is thus characterized by

$$h'(\tau_k) = -\langle \nabla f(x_k), \nabla f(x_{k+1}) \rangle = 0.$$

This means that for this greedy algorithm, two successive descent direction $\nabla f(x_k)$ and $\nabla f(x_{k+1})$ are orthogonal (see Fig. 1.9).

1.4 Convergence Analysis for the Quadratic Case

Convergence analysis for the quadratic case. We first analyze this algorithm in the case of the quadratic loss, which can be written as

$$f(x) = \frac{1}{2}\|Ax - y\|^2 = \frac{1}{2}\langle Cx, x\rangle - \langle x, b\rangle + \text{cst} \quad \text{where} \quad \begin{cases} C \overset{\text{def.}}{=} A^\top A \in \mathbb{R}^{p \times p}, \\ b \overset{\text{def.}}{=} A^\top y \in \mathbb{R}^p. \end{cases}$$

We already saw that in (1.8) if $\ker(A) = \{0\}$, which is equivalent to C being invertible, then there exists a single global minimizer $x^\star = (A^\top A)^{-1} A^\top y = C^{-1} u$.

Note that a function of the form $\frac{1}{2}\langle Cx, x\rangle - \langle x, b\rangle$ is convex if and only if the symmetric matrix C is positive semi-definite, i.e. that all its eigenvalues are non-negative (as already seen in (1.9)).

Proposition 4. *For $f(x) = \langle Cx, x\rangle - \langle b, x\rangle$ (C being symmetric semi-definite positive) with the eigen-values of C upper-bounded by L and lower-bounded by $\mu > 0$, assuming there exists $(\tau_{\min}, \tau_{\max})$ such that*

$$0 < \tau_{\min} \leqslant \tau_\ell \leqslant \tilde{\tau}_{\max} < \frac{2}{L} \tag{1.13}$$

then there exists $0 \leqslant \tilde{\rho} < 1$ such that

$$\|x_k - x^\star\| \leqslant \tilde{\rho}^\ell \|x_0 - x^\star\|. \tag{1.14}$$

The best rate $\tilde{\rho}$ is obtained for

$$\tau_\ell = \frac{2}{L+\mu} \quad \Longrightarrow \quad \tilde{\rho} \overset{\text{def.}}{=} \frac{L-\mu}{L+\mu} = 1 - \frac{2\varepsilon}{1+\varepsilon} \quad \text{where} \quad \varepsilon \overset{\text{def.}}{=} \mu/L. \tag{1.15}$$

Proof. One iterate of gradient descent reads

$$x_{k+1} = x_k - \tau_\ell (Cx_k - b).$$

Since the solution $x^\star$ (which by the way is unique by strict convexity) satisfy the first order condition $Cx^\star = b$, it gives

$$x_{k+1} - x^\star = x_k - x^\star - \tau_\ell C(x_k - x^\star) = (\mathrm{Id}_p - \tau_\ell C)(x_k - x^\star).$$

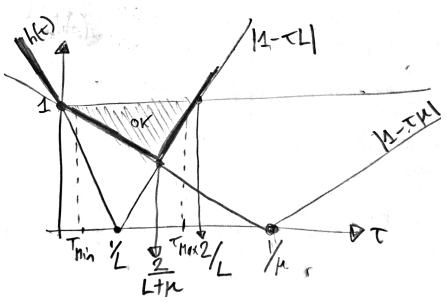


Figure 1.10: Contraction constant $h(\tau)$ for a quadratic function (right).

If $S \in \mathbb{R}^{p \times p}$ is a symmetric matrix, one has

$$\|Sz\| \leqslant \|S\|_{\text{op}} \|z\| \quad \text{where} \quad \|S\|_{\text{op}} \stackrel{\text{def.}}{=} \max_k |\lambda_k(S)|,$$

where $\lambda_k(S)$ are the eigenvalues of S and $\sigma_k(S) \stackrel{\text{def.}}{=} |\lambda_k(S)|$ are its singular values. Indeed, S can be diagonalized in an orthogonal basis U, so that $S = U \operatorname{diag}(\lambda_k(S)) U^\top$, and $S^\top S = S^2 = U \operatorname{diag}(\lambda_k(S)^2) U^\top$ so that

$$\begin{aligned} \|Sz\|^2 &= \langle S^\top S z,\, z \rangle = \langle U \operatorname{diag}(\lambda_k) U^\top z,\, z \rangle = \langle \operatorname{diag}(\lambda_k^2) U^\top z,\, U^\top z \rangle \\ &= \sum_i \lambda_k^2 (U^\top z)_k^2 \leqslant \max_k(\lambda_k^2) \|U^\top z\|^2 = \max_k(\lambda_k^2) \|z\|^2. \end{aligned}$$

Applying this to $S = \text{Id}_p - \tau_\ell C$, one has

$$h(\tau) \stackrel{\text{def.}}{=} \|\text{Id}_p - \tau_\ell C\|_{\text{op}} = \max_k |\lambda_k(\text{Id}_p - \tau_\ell C)| = \max_k |1 - \tau_\ell \lambda_k(C)| = \max(|1 - \tau_\ell \sigma_{\max}(C)|, |1 - \tau \sigma_{\min}(C)|)$$

For a quadratic function, one has $\sigma_{\min}(C) = \mu, \sigma_{\max}(C) = L$. Figure 1.10, right, shows a display of $h(\tau)$. One has that for $0 < \tau < 2/L$, $h(\tau) < 1$. The optimal value is reached at $\tau^\star = \frac{2}{L+\mu}$ and then

$$h(\tau^\star) = \left| .1 - \frac{2L}{L+\mu} \right| = \frac{L-\mu}{L+\mu}.$$

□

Note that when the condition number $\varepsilon \stackrel{\text{def.}}{=} \mu/L \ll 1$ is small (which is the typical setup for ill-posed problems), then the contraction constant appearing in (1.15) scales like

$$\tilde{\rho} \sim 1 - 2\varepsilon. \tag{1.16}$$

The quantity ε in some sense reflects the inverse-conditioning of the problem. For quadratic function, it indeed corresponds exactly to the inverse of the condition number (which is the ratio of the largest to smallest singular value). The condition number is minimum and equal to 1 for orthogonal matrices.

The error decay rate (1.14), although it is geometrical $O(\rho^\ell)$ is called a "linear rate" in the optimization literature. It is a "global" rate because it hold for all ℓ (and not only for large enough ℓ).

If $\ker(A) \neq \{0\}$, then C is not definite positive (some of its eigenvalues vanish), and the set of solution is infinite. One can however still show a linear rate, by showing that actually the iterations x_k are orthogonal to $\ker(A)$ and redo the above proof replacing μ by the smaller non-zero eigenvalue of C. This analysis however leads to a very poor rate ρ (very close to 1) because μ can be arbitrary close to 0. Furthermore, such a proof does not extends to non-quadratic functions. It is thus necessary to do a different theoretical analysis, which only shows a sublinear rate on the objective function f itself rather than on the iterates x_k.

Proposition 5. *For $f(x) = \langle Cx, x\rangle - \langle b, x\rangle$, assuming the eigenvalue of C are bounded by L, then if $0 < \tau_\ell = \tau < 2/L$ is constant, then*

$$f(x_k) - f(x^\star) \leqslant \frac{\mathrm{dist}(x_0, \mathrm{argmin}\, f)^2}{\tau 8k}.$$

where

$$\mathrm{dist}(x_0, \mathrm{argmin}\, f) \overset{\text{def.}}{=} \min_{x^\star \in \mathrm{argmin}\, f} \|x_0 - x^\star\|.$$

Proof. We have $Cx^\star = b$ for any minimizer $x^\star$ and $x_{k+1} = x_k - \tau(Cx_k - b)$ so that as before

$$x_k - x^\star = (\mathrm{Id}_p - \tau C)^k (x_0 - x^\star).$$

Now one has

$$\frac{1}{2}\langle C(x_k - x^\star, x_k - x^\star\rangle = \frac{1}{2}\langle Cx_k, x_k\rangle - \langle Cx_k, x^\star\rangle + \frac{1}{2}\langle Cx^\star, x^\star\rangle$$

and we have $\langle Cx_k, x^\star\rangle = \langle x_k, Cx^\star\rangle = \langle x_k, b\rangle$ and also $\langle Cx^\star, x^\star\rangle = \langle x^\star, x\rangle$ so that

$$\frac{1}{2}\langle C(x_k - x^\star, x_k - x^\star\rangle = \frac{1}{2}\langle Cx_k, x_k\rangle - \langle x_k, b\rangle + \frac{1}{2}\langle x^\star, b\rangle = f(x_k) - f(x^\star)$$

where we have used the fact that

$$f(x^\star) = \frac{1}{2}\langle (A^\top A)(A^\top A)^{-1}b, (A^\top A)^{-1}b\rangle - \langle (A^\top A)^{-1}b, b\rangle \quad \Longrightarrow \quad \frac{1}{2}\langle x^\star, b\rangle = -f(x^\star).$$

This thus implies

$$f(x_k) - f(x^\star) = \frac{1}{2}\langle (\mathrm{Id}_p - \tau C)^k C(\mathrm{Id}_p - \tau C)^k (x_0 - x^\star), x_0 - x^\star\rangle \leqslant \frac{\sigma_{\max}(M_k)}{2} \min_{x^\star} \|x_0 - x^\star\|^2$$

where we have denoted

$$M_k \overset{\text{def.}}{=} (\mathrm{Id}_p - \tau C)^k C(\mathrm{Id}_p - \tau C)^k.$$

One has

$$\sigma_\ell(M_k) = \sigma_\ell(C)(1 - \tau\sigma_\ell(C))^{2k} \leqslant \frac{1}{\tau 4k}$$

since one can show that (setting $t = \tau\sigma_\ell(C) \leqslant 1$ because of the hypotheses)

$$\forall\, t \in [0,1], \quad (1-t)^{2k} t \leqslant \frac{1}{4k}.$$

Indeed, one has

$$(1-t)^{2k} t \leqslant (e^{-t})^{2k} t = \frac{1}{2k}(2kt)e^{-2kt} \leqslant \frac{1}{2k}\sup_{u \geqslant 0} ue^{-u} = \frac{1}{2ek} \leqslant \frac{1}{4k}.$$

□

1.5 Convergence Analysis for the General Case

We detail the theoretical analysis of convergence for general smooth convex functions. The general idea is to replace the linear operator C involved in the quadratic case by the second order derivative (the hessian matrix).

Hessian. A differentiable function f is said to be twice differentiable at x if there exists a symmetric matrix (the hessian) $\partial^2 f(x) \in \mathbb{R}^{p\times p}$ such that

$$f(x+\varepsilon) = f(x) + \langle \nabla f(x), \varepsilon\rangle + \frac{1}{2}\langle \partial^2 f(x)\varepsilon, \varepsilon\rangle + o(\|\varepsilon\|^2).$$

This means that one can approximate f near x by a quadratic function. Roughly speaking, the theoretical analysis of the gradient descent for a generic function is obtained by applying this approximation and using the previous proof.

This Hessian can be obtained by performing an expansion (i.e. computing the differential) of the gradient since

$$\nabla f(x+\varepsilon) = \nabla f(x) + [\partial^2 f(x)](\varepsilon) + o(\|\varepsilon\|)$$

where $[\partial^2 f(x)](\varepsilon) \in \mathbb{R}^p$ denotes the multiplication of the matrix $\partial^2 f(x)$ with the vector ε.

One can show that a twice differentiable function f on $\mathbb{R}^p$ is convex if and only if for all x the symmetric matrix $\partial^2 f(x)$ is positive semi-definite, i.e. all its eigenvalues are non-negative. Furthermore, if these eigenvalues are strictly positive then f is strictly convex (but the converse is not true, for instance x^4 is strictly convex on $\mathbb{R}$ but its second derivative vanishes at $x = 0$).

For instance, for a quadratic function $f(x) = \langle Cx, x\rangle - \langle x, u\rangle$, one has $\nabla f(x) = Cx - u$ and thus $\partial^2 f(x) = C$ (which is thus constant). For the classification function, one has

$$\nabla f(x) = -A^\top \operatorname{diag}(y)\nabla L(-\operatorname{diag}(y)Ax).$$

and thus

$$\begin{aligned}\nabla f(x+\varepsilon) &= -A^\top \operatorname{diag}(y)\nabla L(-\operatorname{diag}(y)Ax - -\operatorname{diag}(y)A\varepsilon)\\ &= \nabla f(x) - A^\top \operatorname{diag}(y)[\partial^2 L(-\operatorname{diag}(y)Ax)](-\operatorname{diag}(y)A\varepsilon)\end{aligned}$$

Since $\nabla L(u) = (\ell'(u_i))$ one has $\partial^2 L(u) = \operatorname{diag}(\ell''(u_i))$. This means that

$$\partial^2 f(x) = A^\top \operatorname{diag}(y) \times \operatorname{diag}(\ell''(-\operatorname{diag}(y)Ax)) \times \operatorname{diag}(y)A.$$

One verifies that this matrix is symmetric and positive if ℓ is convex and thus ℓ'' is positive.

Remark 5 (Second order optimality condition). The first use of Hessian is to decide wether a point $x^\star$ with $\nabla f(x^\star)$ is a local minimum or not. Indeed, if $\partial^2 f(x^\star)$ is a positive matrix (i.e. its eigenvalues are strictly positive), then $x^\star$ is a strict local minimum. Note that if $\partial^2 f(x^\star)$ is only non-negative (i.e. some its eigenvalues might vanish) then one cannot deduce anything (such as for instance x^3 on $\mathbb{R}$). Conversely, if $x^\star$ is a local minimum then $\partial^2 f(x)$

Remark 6 (Second order optimality condition). A second use, is to be used in practice to define second order method (such as Newton's algorithm), which converge faster than gradient descent, but are more costly. The generalized gradient descent reads

$$x_{k+1} = x_k - H_k \nabla f(x_k)$$

where $H_k \in \mathbb{R}^{p\times p}$ is a positive symmetric matrix. One recovers the gradient descent when using $H_k = \tau_k \mathrm{Id}_p$, and Newton's algorithm corresponds to using the inverse of the Hessian $H_k = [\partial^2 f(x_k)]^{-1}$. Note that

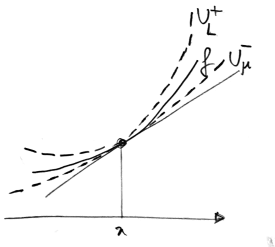
$$f(x_k) = f(x_k) - \langle H_k \nabla f(x_k), \nabla f(x_k)\rangle + o(\|H_k \nabla f(x_k)\|).$$

Since H_k is positive, if x_k is not a minimizer, i.e. $\nabla f(x_k) \neq 0$, then $\langle H_k \nabla f(x_k), \nabla f(x_k)\rangle > 0$. So if H_k is small enough one has a valid descent method in the sense that $f(x_{k+1}) < f(x_k)$. It is not the purpose of this chapter to explain in more detail these type of algorithm.

The last use of Hessian, that we explore next, is to study theoretically the convergence of the gradient descent. One simply needs to replace the boundedness of the eigenvalue of C of a quadratic function by a boundedness of the eigenvalues of $\partial^2 f(x)$ for all x.

Smoothness and strong convexity. One also needs to quantify the smoothness of f. This is enforced by requiring that the gradient is L-Lipschitz, i.e.

$$\forall\,(x,x')\in(\mathbb{R}^p)^2,\quad \|\nabla f(x)-\nabla f(x')\|\leqslant L\|x-x'\|. \tag{$\mathcal{R}_L$}$$



In order to obtain fast convergence of the iterates themselve, it is needed that the function has enough "curvature" (i.e. is not too flat), which corresponds to imposing that f is μ-strongly convex

$$\forall\,(x,x'),\in(\mathbb{R}^p)^2,\quad \langle\nabla f(x)-\nabla f(x'),x-x'\rangle\geqslant\mu\|x-x'\|^2. \tag{$\mathcal{S}_\mu$}$$

The following proposition express these conditions as constraints on the hessian for $\mathcal{C}^2$ functions.

Proposition 6. *Conditions ($\mathcal{R}_L$) and ($\mathcal{S}_\mu$) imply*

$$\forall\,(x,x'),\quad f(x')+\langle\nabla f(x),x'-x\rangle+\frac{\mu}{2}\|x-x'\|^2\leqslant f(x)\leqslant f(x')+\langle\nabla f(x'),x'-x\rangle+\frac{L}{2}\|x-x'\|^2. \tag{1.17}$$

If f is of class $\mathcal{C}^2$, conditions ($\mathcal{R}_L$) and ($\mathcal{S}_\mu$) are equivalent to

$$\forall\,x,\quad \mu\mathrm{Id}_p\preceq\partial^2 f(x)\preceq L\mathrm{Id}_p \tag{1.18}$$

where $\partial^2 f(x)\in\mathbb{R}^{p\times p}$ is the Hessian of f, and where $\preceq$ is the natural order on symmetric matrices, i.e.

$$A\preceq B\quad\Longleftrightarrow\quad\forall\,x\in\mathbb{R}^p,\quad\langle Au,u\rangle\leqslant\langle Bu,u\rangle.$$

Proof. We prove (1.17), using Taylor expansion with integral remain

$$f(x')-f(x)=\int_0^1\langle\nabla f(x_t),x'-x\rangle\mathrm{d}t=\langle\nabla f(x),x'-x\rangle+\int_0^1\langle\nabla f(x_t)-\nabla f(x),x'-x\rangle\mathrm{d}t$$

where $x_t\overset{\text{def.}}{=}x+t(x'-x)$. Using Cauchy-Schwartz, and then the smoothness hypothesis ($\mathcal{R}_L$)

$$f(x')-f(x)\leqslant\langle\nabla f(x),x'-x\rangle+\int_0^1 L\|x_t-x\|\|x'-x\|\mathrm{d}t\leqslant\langle\nabla f(x),x'-x\rangle+L\|x'-x\|^2\int_0^1 t\mathrm{d}t$$

which is the desired upper-bound. Using directly ($\mathcal{S}_\mu$) gives

$$f(x')-f(x)=\langle\nabla f(x),x'-x\rangle+\int_0^1\langle\nabla f(x_t)-\nabla f(x),\frac{x_t-x}{t}\rangle\mathrm{d}t\geqslant\langle\nabla f(x),x'-x\rangle+\mu\int_0^1\frac{1}{t}\|x_t-x\|^2\mathrm{d}t$$

which gives the desired result since $\|x_t-x\|^2/t=t\|x'-x\|^2$. □

The relation (1.17) shows that a smooth (resp. strongly convex) functional is bellow a quadratic tangential majorant (resp. minorant).

Condition (1.18) thus reads that the singular values of $\partial^2 f(x)$ should be contained in the interval $[\mu,L]$. The upper bound is also equivalent to $\|\partial^2 f(x)\|_{\text{op}}\leqslant L$ where $\|\cdot\|_{\text{op}}$ is the operator norm, i.e. the largest singular value. In the special case of a quadratic function of the form $\langle Cx,x\rangle-\langle b,x\rangle$ (recall that necessarily C is semi-definite symmetric positive for this function to be convex), $\partial^2 f(x)=C$ is constant, so that $[\mu,L]$ can be chosen to be the range of the eigenvalues of C.

Convergence analysis. We now give convergence theorem for a general convex function. On contrast to quadratic function, if one does not assumes strong convexity, one can only show a sub-linear rate on the function values (and no rate at all on the iterates themselves!). It is only when one assume strong convexity that linear rate is obtained. Note that in this case, the solution of the minimization problem is not necessarily unique.

Theorem 1. *If f satisfy conditions $(\mathcal{R}_L)$, assuming there exists $(\tau_{\min}, \tau_{\max})$ such that*

$$0 < \tau_{\min} \leqslant \tau_\ell \leqslant \tau_{\max} < \frac{2}{L}, \tag{1.19}$$

then x_k converges to a solution $x^\star$ of (1.1) and there exists $C > 0$ such that

$$f(x_k) - f(x^\star) \leqslant \frac{C}{\ell + 1}. \tag{1.20}$$

If furthermore f is μ-strongly convex, then there exists $0 \leqslant \rho < 1$ such that $\|x_k - x^\star\| \leqslant \rho^\ell \|x_0 - x^\star\|$.

Proof. In the case where f is not strongly convex, we only prove (1.20) since the proof that x_k converges is more technical. Note indeed that if the minimizer $x^\star$ is non-unique, then it might be the case that the iterate x_k "cycle" while approaching the set of minimizer, but actually convexity of f prevents this kind of pathological behavior. For simplicity, we do the proof in the case $\tau_\ell = 1/L$, but it extends to the general case. The L-smoothness property imply (1.17), which reads

$$f(x_{k+1}) \leqslant f(x_k) + \langle \nabla f(x_k),\, x_{k+1} - x_k \rangle + \frac{L}{2}\|x_{k+1} - x_k\|^2.$$

Using the fact that $x_{k+1} - x_k = -\frac{1}{L}\nabla f(x_k)$, one obtains

$$f(x_{k+1}) \leqslant f(x_k) - \frac{1}{L}\|\nabla f(x_k)\|^2 + \frac{1}{2L}\|\nabla f(x_k)\|^2 \leqslant f(x_k) - \frac{1}{2L}\|\nabla f(x_k)\|^2 \tag{1.21}$$

This shows that $(f(x_k))_\ell$ is a decaying sequence. By convexity

$$f(x_k) + \langle \nabla f(x_k),\, x^\star - x_k \rangle \leqslant f(x^\star)$$

and plugging this in (1.21) shows

$$\begin{aligned} f(x_{k+1}) &\leqslant f(x^\star) - \langle \nabla f(x_k),\, x^\star - x_k \rangle - \frac{1}{2L}\|\nabla f(x_k)\|^2 && (1.22)\\ &= f(x^\star) + \frac{L}{2}\left(\|x_k - x^\star\|^2 - \|x_k - x^\star - \frac{1}{L}\nabla f(x_k)\|^2 \right) && (1.23)\\ &= f(x^\star) + \frac{L}{2}\left(\|x_k - x^\star\|^2 - \|x^\star - x_{k+1}\|^2 \right). && (1.24) \end{aligned}$$

Summing these inequalities for $\ell = 0, \ldots, k$, one obtains

$$\sum_{\ell=0}^{k} f(x_{k+1}) - (k+1) f(x^\star) \leqslant \frac{L}{2}\left(\|x_0 - x^\star\|^2 - \|x^{(k+1)} - x^\star\|^2 \right)$$

and since $f(x_{k+1})$ is decaying $\sum_{\ell=0}^{k} f(x_{k+1}) \geqslant (k+1) f(x^{(k+1)})$, thus

$$f(x^{(k+1)}) - f(x^\star) \leqslant \frac{L\|x_0 - x^\star\|^2}{2(k+1)}$$

which gives (1.20) for $C \overset{\text{def.}}{=} L\|x_0 - x^\star\|^2/2$.

If we now assume f is μ-strongly convex, then, using $\nabla f(x^\star) = 0$, one has $\frac{\mu}{2}\|x^\star - x\|^2 \leqslant f(x) - f(x^\star)$ for all x. Re-manipulating (1.24) gives

$$\frac{\mu}{2}\|x_{k+1} - x^\star\|^2 \leqslant f(x_{k+1}) - f(x^\star) \leqslant \frac{L}{2}\left(\|x_k - x^\star\|^2 - \|x^\star - x_{k+1}\|^2 \right),$$

and hence

$$\|x_{k+1} - x^\star\| \leqslant \sqrt{\frac{L}{L+\mu}}\|x_{k+1} - x^\star\|, \tag{1.25}$$

which is the desired result. □

Note that in the low conditioning setting $\varepsilon \ll 1$, one retrieve a dependency of the rate (1.25) similar to the one of quadratic functions (1.16), indeed

$$\sqrt{\frac{L}{L+\mu}} = (1+\varepsilon)^{-\frac{1}{2}} \sim 1 - \frac{1}{2}\varepsilon.$$

1.6 Stochastic Optimization

We detail some important stochastic Gradient Descent methods, which enable to perform optimization in the setting where the number of samples n is large and even infinite.

1.6.1 Minimizing Sums and Expectation

A large class of functionals in machine learning can be expressed as minimizing large sums of the form

$$\min_{x \in \mathbb{R}^p} f(x) \overset{\text{def.}}{=} \frac{1}{n} \sum_{i=1}^{n} f_i(x) \tag{1.26}$$

or even expectations of the form

$$\min_{x \in \mathbb{R}^p} f(x) \overset{\text{def.}}{=} \mathbb{E}_{\mathbf{z} \sim \pi}(f(x, \mathbf{z})) = \int_{\mathcal{Z}} f(x, z) \mathrm{d}\pi(z). \tag{1.27}$$

Problem (1.26) can be seen as a special case of (1.27), when using a discrete empirical uniform measure $\pi = \sum_{i=1}^{n} \delta_i$ and setting $f(x, i) = f_i(x)$. One can also viewed (1.26) as a discretized "empirical" version of (1.27) when drawing $(z_i)_i$ i.i.d. according to $\mathbf{z}$ and defining $f_i(x) = f(x, z_i)$. In this setup, (1.26) converges to (1.27) as $n \to +\infty$.

A typical example of such a class of problems is empirical risk minimization for linear model, where in these cases

$$f_i(x) = \ell(\langle a_i,\, x \rangle, y_i) \quad \text{and} \quad f(x, z) = \ell(\langle a,\, x \rangle, y) \tag{1.28}$$

for $z = (a, y) \in \mathcal{Z} = (\mathcal{A} = \mathbb{R}^p) \times \mathcal{Y}$ (typically $\mathcal{Y} = \mathbb{R}$ or $\mathcal{Y} = \{-1, +1\}$ for regression and classification), where ℓ is some loss function. We illustrate bellow the methods on binary logistic classification, where

$$L(s, y) \overset{\text{def.}}{=} \log(1 + \exp(-sy)), \tag{1.29}$$

see Section **??** for details. But this extends to arbitrary parametric models, and in particular deep neural networks as detailed in Section **??**.

While some algorithms (in particular batch gradient descent) are specific to finite sums (1.26), the stochastic methods we detail next work verbatim (with the same convergence guarantees) in the expectation case (1.27). For the sake of simplicity, we however do the exposition for the finite sums case, which is sufficient in the vast majority of cases. But one should keep in mind that n can be arbitrarily large, so it is not acceptable in this setting to use algorithms whose complexity per iteration depend on n.

The general idea underlying stochastic optimization methods is *not* to have faster algorithms with respect to traditional optimization schemes such as those detailed in Chapter 1. In almost all cases, if n is not too large so that one afford the price of doing a few non-stochastic iterations, then deterministic methods are faster. But if n is so large that one cannot do even a single deterministic iteration, then stochastic methods allow one to have a fine grained scheme by breaking the cost of determinstic iterations in smaller chunks. Another advantage is that they are quite easy to parallelize.

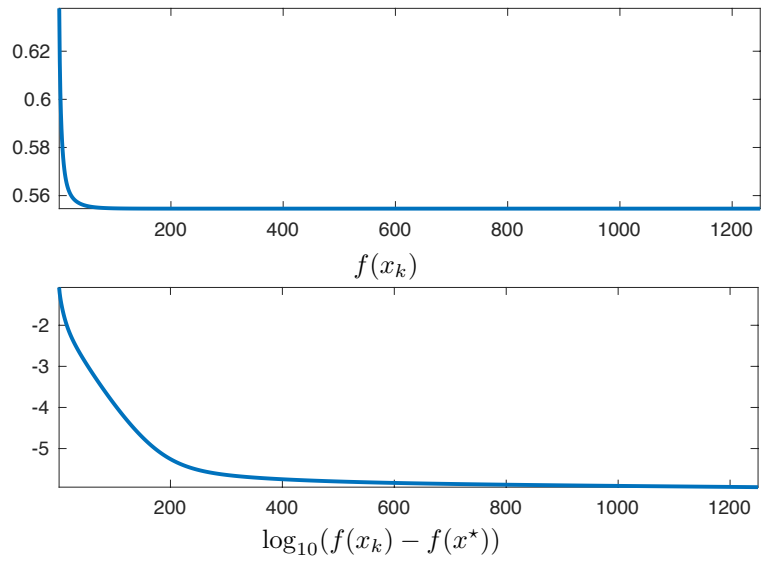


Figure 1.11: Evolution of the error of the BGD for logistic classification.

1.6.2 Batch Gradient Descent (BGD)

The usual deterministic (batch) gradient descent (BGD) is studied in details in Section 1.1. Its iterations read

$$x_{k+1} = x_k - \tau_k \nabla f(x_k)$$

and the step size should be chosen as $0 < \tau_{\min} < \tau_k < \tau_{\max} \overset{\text{def.}}{=} 2/L$ where L is the Lipschitz constant of the gradient ∇f. In particular, in this deterministic setting, this step size should not go to zero and this ensures quite fast convergence (even linear rates if f is strongly convex).

The computation of the gradient in our setting reads

$$\nabla f(x) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x) \tag{1.30}$$

so it typically has complexity $O(np)$ if computing ∇f_i has linear complexity in p.

For ERM-type functions of the form (1.28), one can do the Taylor expansion of f_i

$$\begin{aligned} f_i(x+\varepsilon) &= \ell(\langle a_i,\, x\rangle + \langle a_i,\, \varepsilon\rangle, y_i) = \ell(\langle a_i,\, x\rangle, y_i) + \ell'(\langle a_i,\, x\rangle, y_i)\langle a_i,\, \varepsilon\rangle + o(\|\varepsilon\|) \\ &= f_i(x) + \langle \ell'(\langle a_i,\, x\rangle, y_i) a_i,\, x\rangle + o(\|\varepsilon\|), \end{aligned}$$

where $\ell(y, y') \in \mathbb{R}$ is the derivative with respect to the first variable, i.e. the gradient of the map $y \in \mathbb{R} \mapsto L(y, y') \in \mathbb{R}$. This computation shows that

$$\nabla f_i(x) = \ell'(\langle a_i,\, x\rangle, y_i) a_i. \tag{1.31}$$

For the logistic loss, one has

$$L'(s, y) = -s \frac{e^{-sy}}{1 + e^{-sy}}.$$

1.6.3 Stochastic Gradient Descent (SGD)

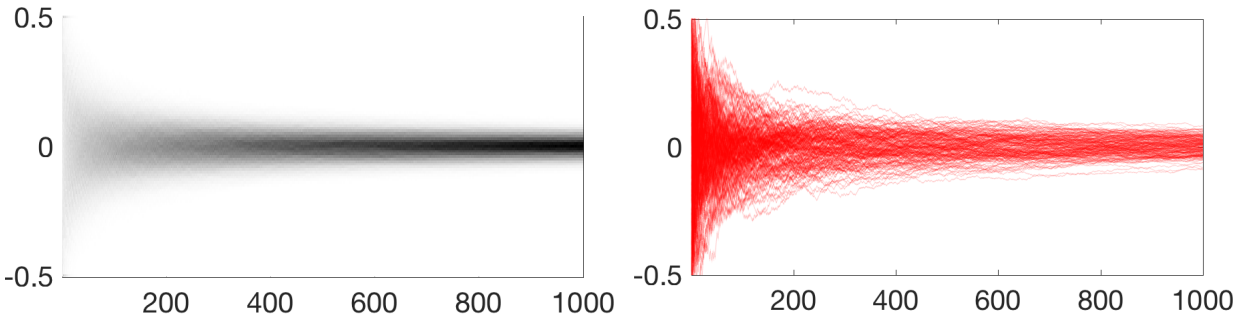


Figure 1.14: Display of a large number of trajectories $k \mapsto x_k \in \mathbb{R}$ generated by several runs of SGD. On the top row, each curve is a trajectory, and the bottom row displays the corresponding density.

For very large n, computing the full gradient ∇f as in (1.30) is prohibitive. The idea of SGD is to trade this exact full gradient by an inexact proxy using a single functional f_i where i is drawn uniformly at random. The main idea that makes this work is that this sampling scheme provides an unbiased estimate of the gradient, in the sense that

$$\mathbb{E}_{\mathbf{i}} \nabla f_{\mathbf{i}}(x) = \nabla f(x) \tag{1.32}$$

where $\mathbf{i}$ is a random variable distributed uniformly in $\{1, \ldots, n\}$.

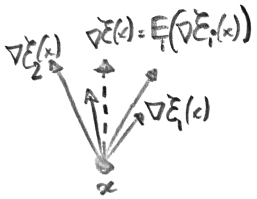


Figure 1.12: Unbiased gradient estimate

Starting from some x_0,the iterations of stochastic gradient descent (SGD) read

$$x_{k+1} = x_k - \tau_k \nabla f_{i(k)}(x_k)$$

where, for each iteration index k, $i(k)$ is drawn uniformly at random in $\{1, \ldots, n\}$. It is important that the iterates x_{k+1} are thus random vectors, and the theoretical analysis of the method thus studies wether this sequence of random vectors converges (in expectation or in probability for instance) toward a deterministic vector (minimizing f), and at which speed.

Note that each step of a batch gradient descent has complexity $O(np)$, while a step of SGD only has complexity $O(p)$. SGD is thus advantageous when n is very large, and one cannot afford to do several passes through the data. In some situation, SGD can provide accurate results even with $k \ll n$, exploiting redundancy between the samples.

A crucial question is the choice of step size schedule τ_k. It must tends to 0 in order to cancel the noise induced on the gradient by the stochastic sampling. But it should not go too fast to zero in order for the method to keep converging.

A typical schedule that ensures both properties is to have asymptotically $\tau_k \sim k^{-1}$ for $k \to +\infty$. We thus propose to use

$$\tau_k \overset{\text{def.}}{=} \frac{\tau_0}{1 + k/k_0} \tag{1.33}$$

Figure 1.13: Schematic view of SGD iterates

where k_0 indicates roughly the number of iterations serving as a "warmup" phase.

Figure 1.14 shows a simple 1-D example to minimize $f_1(x) + f_2(x)$ for $x \in \mathbb{R}$ and $f_1(x) = (x-1)^2$ and $f_2(x) = (x+1)^2$. One can see how the density of the distribution of x_k progressively clusters around the minimizer $x^\star = 0$. Here the distribution of x_0 is uniform on $[-1/2, 1/2]$.

The following theorem shows the convergence in expectation with a $1/\sqrt{k}$ rate on the objective.

Theorem 2. *We assume f is μ-strongly convex as defined in ($\mathcal{S}_\mu$) (i.e. $\mathrm{Id}_p \preceq \partial^2 f(x)$ if f is $\mathcal{C}^2$), and is such that $\|\nabla f_i(x)\|^2 \leqslant C^2$. For the step size choice $\tau_k = \frac{1}{\mu(k+1)}$, one has*

$$\mathbb{E}(\|x_k - x^\star\|^2) \leqslant \frac{R}{k+1} \quad \textit{where} \quad R = \max(\|x_0 - x^\star\|, C^2/\mu^2), \tag{1.34}$$

where $\mathbb{E}$ indicates an expectation with respect to the i.i.d. sampling performed at each iteration.

Proof. By strong convexity, one has

$$f(x^\star) - f(x_k) \geqslant \langle \nabla f(x_k),\, x^\star - x_k \rangle + \frac{\mu}{2}\|x_k - x^\star\|^2$$
$$f(x_k) - f(x^\star) \geqslant \langle \nabla f(x^\star),\, x_k - x^\star \rangle + \frac{\mu}{2}\|x_k - x^\star\|^2.$$

Summing these two inequalities and using $\nabla f(x^\star) = 0$ leads to

$$\langle \nabla f(x_k) - \nabla f(x^\star),\, x_k - x^\star \rangle = \langle \nabla f(x_k),\, x_k - x^\star \rangle \geqslant \mu \|x_k - x^\star\|^2. \tag{1.35}$$

Considering only the expectation with respect to the ransom sample of $i(k) \sim \mathbf{i}_k$, one has

$$\begin{aligned}
\mathbb{E}_{\mathbf{i}_k}(\|x_{k+1} - x^\star\|^2) &= \mathbb{E}_{\mathbf{i}_k}(\|x_k - \tau_k \nabla f_{\mathbf{i}_k}(x_k) - x^\star\|^2) \\
&= \|x_k - x^\star\|^2 + 2\tau_k \langle \mathbb{E}_{\mathbf{i}_k}(\nabla f_{\mathbf{i}_k}(x_k)),\, x^\star - x_k \rangle + \tau_k^2 \mathbb{E}_{\mathbf{i}_k}(\|\nabla f_{\mathbf{i}_k}(x_k)\|^2) \\
&\leqslant \|x_k - x^\star\|^2 + 2\tau_k \langle \nabla f(x_k)),\, x^\star - x_k \rangle + \tau_k^2 C^2
\end{aligned}$$

where we used the fact (1.32) that the gradient is unbiased. Taking now the full expectation with respect to all the other previous iterates, and using (1.35) one obtains

$$\mathbb{E}(\|x_{k+1} - x^\star\|^2) \leqslant \mathbb{E}(\|x_k - x^\star\|^2) - 2\mu\tau_k \mathbb{E}(\|x_k - x^\star\|^2) + \tau_k^2 C^2 = (1 - 2\mu\tau_k)\mathbb{E}(\|x_k - x^\star\|^2) + \tau_k^2 C^2. \tag{1.36}$$

We show by recursion that the bound (1.34) holds. We denote $\varepsilon_k \overset{\text{def.}}{=} \mathbb{E}(\|x_k - x^\star\|^2)$. Indeed, for $k = 0$, this it is true that

$$\varepsilon_0 \leqslant \frac{\max(\|x_0 - x^\star\|, C^2/\mu^2)}{1} = \frac{R}{1}.$$

We now assume that $\varepsilon_k \leqslant \frac{R}{k+1}$. Using (1.36) in the case of $\tau_k = \frac{1}{\mu(k+1)}$, one has, denoting $m = k+1$

$$\begin{aligned}
\varepsilon_{k+1} &\leqslant (1 - 2\mu\tau_k)\varepsilon_k + \tau_k^2 C^2 = \left(1 - \frac{2}{m}\right)\varepsilon_k + \frac{C^2}{(\mu m)^2} \\
&\leqslant \left(1 - \frac{2}{m}\right)\frac{R}{m} + \frac{R}{m^2} = \left(\frac{1}{m} - \frac{1}{m^2}\right) R = \frac{m-1}{m^2} R = \frac{m^2 - 1}{m^2}\frac{1}{m+1} R \leqslant \frac{R}{m+1}
\end{aligned}$$

□

A weakness of SGD (as well as the SGA scheme studied next) is that it only weakly benefit from strong convexity of f. This is in sharp contrast with BGD, which enjoy a fast linear rate for strongly convex functionals, see Theorem 1.

Figure 1.15 displays the evolution of the energy $f(x_k)$. It overlays on top (black dashed curve) the convergence of the batch gradient descent, with a careful scaling of the number of iteration to account for the fact that the complexity of a batch iteration is n times larger.

1.6.4 Stochastic Gradient Descent with Averaging (SGA)

Stochastic gradient descent is slow because of the fast decay of τ_k toward zero. To improve somehow the convergence speed, it is possible to average the past iterate, i.e. run a "classical" SGD on auxiliary variables $(\tilde{x}_k)_k$

$$\tilde{x}^{(\ell+1)} = \tilde{x}_k - \tau_k \nabla f_{i(k)}(\tilde{x}_k)$$

and output as estimated weight vector the Cesaro average

$$x_k \overset{\text{def.}}{=} \frac{1}{k}\sum_{\ell=1}^{k} \tilde{x}_\ell.$$

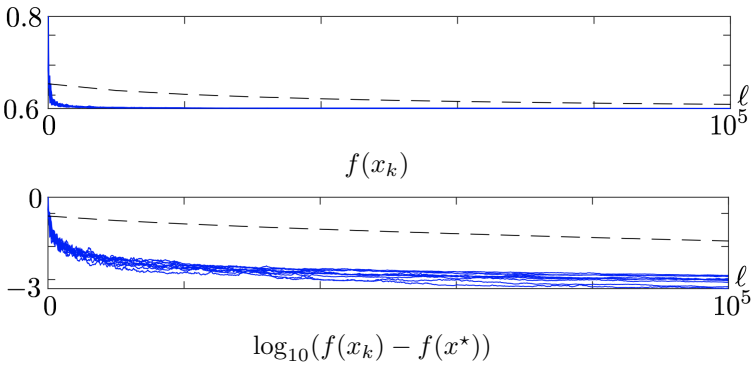


Figure 1.15: Evolution of the error of the SGD for logistic classification (dashed line shows BGD).

This defines the Stochastic Gradient Descent with Averaging (SGA) algorithm.

Note that it is possible to avoid explicitly storing all the iterates by simply updating a running average as follow

$$x_{k+1} = \frac{1}{k}\tilde{x}_k + \frac{k-1}{k}x_k.$$

In this case, a typical choice of decay is rather of the form

$$\tau_k \overset{\text{def.}}{=} \frac{\tau_0}{1+\sqrt{k/k_0}}.$$

Notice that the step size now goes much slower to 0, at rate $k^{-1/2}$.

Typically, because the averaging stabilizes the iterates, the choice of (k_0, τ_0) is less important than for SGD.

Bach proves that for logistic classification, it leads to a faster convergence (the constant involved are smaller) than SGD, since on contrast to SGD, SGA is adaptive to the local strong convexity of E.

1.6.5 Stochastic Averaged Gradient Descent (SAG)

For problem size n where the dataset (of size $n \times p$) can fully fit into memory, it is possible to further improve the SGA method by bookkeeping the previous gradients. This gives rise to the Stochastic Averaged Gradient Descent (SAG) algorithm.

We store all the previously computed gradients in $(G^i)_{i=1}^n$, which necessitates $O(n \times p)$ memory. The iterates are defined by using a proxy g for the batch gradient, which is progressively enhanced during the iterates.

The algorithm reads

$$x_{k+1} = x_k - \tau g \quad \text{where} \quad \begin{cases} h \leftarrow \nabla f_{i(k)}(\tilde{x}_k), \\ g \leftarrow g - G^{i(k)} + h, \\ G^{i(k)} \leftarrow h. \end{cases}$$

Note that in contrast to SGD and SGA, this method uses a fixed step size τ. Similarly to the BGD, in order to ensure convergence, the step size τ should be of the order of $1/L$ where L is the Lipschitz constant of f.

This algorithm improves over SGA and SGD since it has a convergence rate of $O(1/k)$ as does BGD. Furthermore, in the presence of strong convexity (for instance when X is injective for logistic classification), it has a linear convergence rate, i.e.

$$\mathbb{E}(f(x_k)) - f(x^\star) = O\left(\rho^k\right),$$

for some $0 < \rho < 1$.

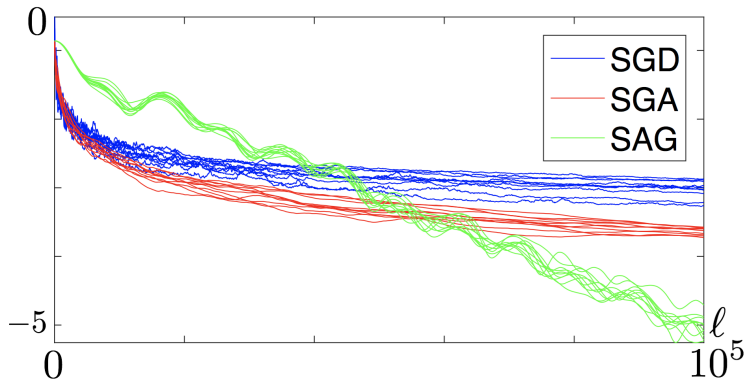


Figure 1.16: Evolution of $\log_{10}(f(x_k) - f(x^\star))$ for SGD, SGA and SAG.

Note that this improvement over SGD and SGA is made possible only because SAG explicitly uses the fact that n is finite (while SGD and SGA can be extended to infinite n and more general minimization of expectations (1.27)).

Figure 1.16 shows a comparison of SGD, SGA and SAG.

1.7 Automatic Differentiation

The main computational bottleneck of these gradient descent methods (batch or stochastic) is the computation of gradients $\nabla f(x)$. We have seen that for simple functionals, such as those encountered in ERM for linear models, and also for MLP with a single hidden layer (see Section **??**), it is possible to compute these gradients in closed form, and that the main computational burden is the evaluation of matrix-vector products. For more complicated functionals (such as those involving deep networks), computing the formula for the gradient quickly becomes cumbersome. Even worse: computing these gradients using the usual chain rule formula is sub-optimal. This section presents a method to compute recursively in an optimal manner these gradients. The purpose of this approach is to automatize this computational step.

We consider $f : \mathbb{R}^p \to \mathbb{R}$ and want to derive a method to evaluate $\nabla f : \mathbb{R}^p \mapsto \mathbb{R}^p$. Approximating this vector field using finite differences, i.e. introducing $\varepsilon > 0$ small enough and computing

$$\frac{1}{\varepsilon}(f(x + \varepsilon\delta_1) - f(x), \ldots, f(x + \varepsilon\delta_p) - f(x))^\top \approx \nabla f(x)$$

requires $p+1$ evaluations of f. For a large p, this is prohibitive. The method we describe in this section (the so-called reverse mode automatic differentiation) has in most cases a cost proportional to a single evaluation of f.

1.7.1 Computational Graphs

We consider a generic function $f(x)$ where $x = (x_1, \ldots, x_s)$ are the input variables. We assume that f is implemented in an algorithm, with intermediate variable $(x_{s+1}, \ldots, x_t$ where t is the total number of variables. The output is x_t, and we thus denote $x_t = f(x)$ this function. We denote $x_k \in \mathbb{R}^{n_k}$ the dimensionality of the variables. The goal is to compute the derivatives $\frac{\partial f(x)}{\partial x_k} \in \mathbb{R}^{n_t \times n_k}$ for $k = 1, \ldots, s$. For the sake of simplicity, one can assume in what follows that $n_k = 1$ so that all the involved quantities are scalar (but if this is not the case, beware that the order of multiplication of the matrices of course matters).

A numerical algorithm can be represented as a succession of functions of the form

$$\forall\, k = s+1, \ldots, t, \quad x_k = f_k(x_1, \ldots, x_{k-1})$$

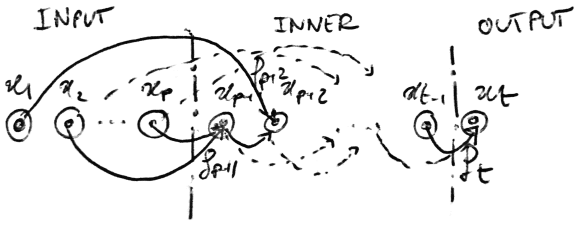


Figure 1.17: A computational graph.

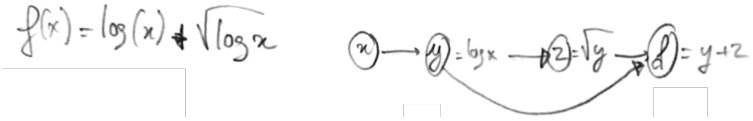


Figure 1.18: Example of a simple computational graph.

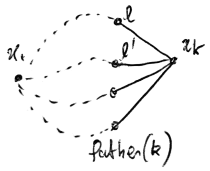
where f_k is a function which only depends on the previous variables, see Fig. 1.17. One can represent this algorithm using a directed acyclic graph (DAG), linking the variables involved in f_k to x_k. The node of this graph are thus conveniently ordered by their indexing, and the directed edges only link a variable to another one with a strictly larger index. The evaluation of $f(x)$ thus corresponds to a forward traversal of this graph.

1.7.2 Forward Mode of Automatic Differentiation

The forward mode correspond to the usual way of computing differentials. It compute the derivative $\frac{\partial x_k}{\partial x_1}$ of all variables x_k with respect to x_1. One then needs to repeat this method p times to compute all the derivative with respect to $x_1, x_2, \ldots, x_p$ (we only write thing for the first variable, the method being of course the same with respect to the other ones).

The method initialize the derivative of the input nodes

$$\frac{\partial x_1}{\partial x_1} = \mathrm{Id}_{n_1 \times n_1}, \quad \frac{\partial x_2}{\partial x_1} = 0_{n_2 \times n_1}, \ldots, \quad \frac{\partial x_s}{\partial x_1} = 0_{n_s \times n_1},$$



(and thus 1 and 0's for scalar variables), and then iteratively make use of the following recursion formula

$$\forall\, k = s{+}1, \ldots, t, \quad \frac{\partial x_k}{\partial x_1} = \sum_{\ell \in \mathrm{father}(k)} \left[\frac{\partial x_k}{\partial x_\ell}\right] \times \frac{\partial x_\ell}{\partial x_1} = \sum_{\ell \in \mathrm{father}(k)} \frac{\partial f_k}{\partial x_\ell}(x_1, \ldots, x_{k-1}) \times \frac{\partial x_\ell}{\partial x_1}.$$

The notation "father(k)" denotes the nodes $\ell < k$ of the graph that are connected to k. Here the quantities being computed (i.e. stored in computer variables) are the derivatives $\frac{\partial x_\ell}{\partial x_1}$, and $\times$ denotes in full generality matrix-matrix multiplications. We have put in $[\ldots]$ an informal notation, since here $\frac{\partial x_k}{\partial x_\ell}$ should be interpreted not as a numerical variable but needs to be interpreted as derivative of the function f_k, which can be evaluated on the fly (we assume that the derivative of the function involved are accessible in closed form).

Assuming all the involved functions $\frac{\partial f_k}{\partial x_k}$ have the same complexity (which is likely to be the case if all the n_k are for instance scalar or have the same dimension), and that the number of father node is bounded, one sees that the complexity of this scheme is p times the complexity of the evaluation of f (since this needs to be repeated p times for $\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_p}$). For a large p, this is prohibitive.

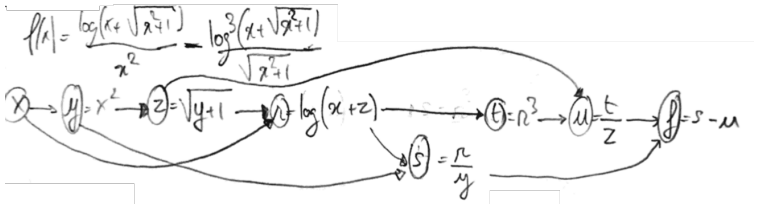


Figure 1.19: Example of a more complex computational graph.

Simple example. We consider the fonction

$$f(x) = \log(x) + \sqrt{\log(x)} \tag{1.37}$$

whose computational graph is displayed on Figure 1.18. The iterations of the forward mode read

$$\begin{aligned}
&\frac{\partial x}{\partial x} = 1 \\
&\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial x}\right]\frac{\partial x}{\partial x} = \frac{1}{x}\frac{\partial x}{\partial x} && \{x \mapsto y = \log(x)\} \\
&\frac{\partial z}{\partial x} = \left[\frac{\partial z}{\partial y}\right]\frac{\partial y}{\partial x} = \frac{1}{2\sqrt{y}}\frac{\partial y}{\partial x} && \{y \mapsto z = \sqrt{y}\} \\
&\frac{\partial f}{\partial x} = \left[\frac{\partial f}{\partial z}\right]\frac{\partial z}{\partial x} + \left[\frac{\partial f}{\partial y}\right]\frac{\partial y}{\partial x} = 1\frac{\partial z}{\partial x} + 1\frac{\partial y}{\partial x} && \{(x,z) \mapsto f = x + z\}
\end{aligned}$$

More complex example. We now consider the fonction

$$f(x) = \frac{\log(x + \sqrt{x^2+1})}{x^2} - \frac{\log^3(x + \sqrt{x^2+1})}{\sqrt{x^2+1}} \tag{1.38}$$

whose computational graph is displayed on Figure 1.19. The iterations of the forward mode read

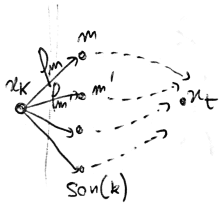
$$\begin{aligned}
&\frac{\partial x}{\partial x} = 1 \\
&\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial x}\right]\frac{\partial x}{\partial x} = 2x\frac{\partial x}{\partial x} && \{x \mapsto y = x^2\} \\
&\frac{\partial z}{\partial x} = \left[\frac{\partial z}{\partial y}\right]\frac{\partial y}{\partial x} = \frac{1}{2\sqrt{y+1}}\frac{\partial y}{\partial x} && \{y \mapsto z = \sqrt{y+1}\} \\
&\frac{\partial r}{\partial x} = \left[\frac{\partial r}{\partial z}\right]\frac{\partial x}{\partial x} + \left[\frac{\partial r}{\partial z}\right]\frac{\partial z}{\partial x} = \frac{1}{x+r}\frac{\partial x}{\partial x} + \frac{1}{x+r}\frac{\partial z}{\partial x} && \{(x,z) \mapsto r = \log(x+z)\} \\
&\frac{\partial s}{\partial x} = \left[\frac{\partial s}{\partial r}\right]\frac{\partial r}{\partial x} + \left[\frac{\partial s}{\partial y}\right]\frac{\partial y}{\partial x} = \frac{1}{y}\frac{\partial r}{\partial x} - \frac{r}{y^2}\frac{\partial y}{\partial x} && \{(r,y) \mapsto s = \frac{r}{y}\} \\
&\frac{\partial t}{\partial x} = \left[\frac{\partial t}{\partial r}\right]\frac{\partial r}{\partial x} = 3r^2\frac{\partial r}{\partial x} && \{r \mapsto t = r^3\} \\
&\frac{\partial u}{\partial x} = \left[\frac{\partial u}{\partial t}\right]\frac{\partial t}{\partial x} + \left[\frac{\partial u}{\partial z}\right]\frac{\partial z}{\partial x} = \frac{1}{z}\frac{\partial t}{\partial x} - \frac{t}{z^2}\frac{\partial z}{\partial x} && \{(t,z) \mapsto u = \frac{t}{z}\} \\
&\frac{\partial f}{\partial x} = \left[\frac{\partial f}{\partial s}\right]\frac{\partial s}{\partial x} + \left[\frac{\partial f}{\partial u}\right]\frac{\partial u}{\partial x} = 1\frac{\partial s}{\partial x} - 1\frac{\partial u}{\partial x} && \{(s,u) \mapsto f = s - u\}
\end{aligned}$$

1.7.3 Reverse Mode of Automatic Differentiation

Instead of evaluating the differentials $\frac{\partial x_k}{\partial x_1}$ which is problematic for a large p, the reverse mode evaluates the differentials $\frac{\partial x_t}{\partial x_k}$, i.e. it computes the derivative of the output node with respect to the all the inner nodes.

The method initialize the derivative of the final node

$$\frac{\partial x_t}{\partial x_t} = \mathrm{Id}_{n_t \times n_t},$$



and then iteratively makes use, from the last node to the first, of the following recursion formula

$$\forall\, k = t-1, t-2, \ldots, 1, \quad \frac{\partial x_t}{\partial x_k} = \sum_{m \in \mathrm{son}(k)} \frac{\partial x_t}{\partial x_m} \times \left[\frac{\partial x_m}{\partial x_k}\right] = \sum_{m \in \mathrm{son}(k)} \frac{\partial x_t}{\partial x_m} \times \frac{\partial f_m(x_1, \ldots, x_m)}{\partial x_k}.$$

The notation "father(k)" denotes the nodes $\ell < k$ of the graph that are connected to k.

Back-propagation. In the special case where $x_t \in \mathbb{R}$, then $\frac{\partial x_t}{\partial x_k} = [\nabla_{x_k} f(x)]^\top \in \mathbb{R}^{1 \times n_k}$ and one can write the recursion on the gradient vector as follow

$$\forall\, k = t-1, t-2, \ldots, 1, \quad \nabla_{x_k} f(x) = \sum_{m \in \mathrm{son}(k)} \left(\frac{\partial f_m(x_1, \ldots, x_m)}{\partial x_k}\right)^\top (\nabla_{x_m} f(x)).$$

where $\left(\frac{\partial f_m(x_1, \ldots, x_m)}{\partial x_k}\right)^\top \in \mathbb{R}^{n_k \times n_m}$ is the adjoint of the Jacobian of f_m. This form of recursion using adjoint is often referred to as "back-propagation", and is the most frequent setting in applications to ML.

In general, when $n_t = 1$, the backward is the optimal way to compute the gradient of a function. Its drawback is that it necessitate the pre-computation of all the intermediate variables $(x_k)_{k=p}^t$, which can be prohibitive in term of memory usage when t is large. There exists check-pointing method to alleviate this issue, but it is out of the scope of this course.

Simple example. We consider once again the fonction $f(x)$ of (1.37), the iterations of the reverse mode read

$$\begin{aligned}
&\frac{\partial f}{\partial f} = 1 \\
&\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f}\left[\frac{\partial f}{\partial z}\right] = \frac{\partial f}{\partial f} 1 && \{z \mapsto f = x + z\} \\
&\frac{\partial f}{\partial y} = \frac{\partial f}{\partial z}\left[\frac{\partial z}{\partial y}\right] + \frac{\partial f}{\partial f}\left[\frac{\partial f}{\partial y}\right] = \frac{\partial f}{\partial z}\frac{1}{2\sqrt{y}} + \frac{\partial f}{\partial f} 1 && \{y \mapsto z = \sqrt{y}, y \mapsto f = y + z\} \\
&\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y}\left[\frac{\partial y}{\partial x}\right] = \frac{\partial f}{\partial y}\frac{1}{x} && \{x \mapsto y = \log(x)\}
\end{aligned}$$

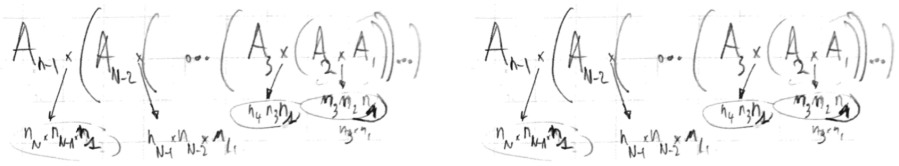


Figure 1.20: Complexity of forward (left) and backward (right) modes for feedforward graphs.

More complex example. We consider once again the fonction $f(x)$ of (1.38), the iterations of the reverse mode read

$$
\begin{aligned}
&\frac{\partial f}{\partial f} = 1 \\
&\frac{\partial f}{\partial u} = \frac{\partial f}{\partial f}\left[\frac{\partial f}{\partial u}\right] = \frac{\partial f}{\partial f}(-1) && \{u \mapsto f = s - u\} \\
&\frac{\partial f}{\partial t} = \frac{\partial f}{\partial u}\left[\frac{\partial u}{\partial t}\right] = \frac{\partial f}{\partial u}\frac{1}{z} && \{t \mapsto u = \sqrt{y}, y\frac{t}{z}\} \\
&\frac{\partial f}{\partial s} = \frac{\partial f}{\partial f}\left[\frac{\partial f}{\partial s}\right] = \frac{\partial f}{\partial f}1 && \{s \mapsto f = s - u\} \\
&\frac{\partial f}{\partial r} = \frac{\partial f}{\partial t}\left[\frac{\partial t}{\partial r}\right] + \frac{\partial f}{\partial s}\left[\frac{\partial s}{\partial r}\right] = \frac{\partial f}{\partial t}3r^2 + \frac{\partial f}{\partial s}\frac{1}{y} && \{r \mapsto s = \frac{r}{y}, r \mapsto t = r^3\} \\
&\frac{\partial f}{\partial z} = \frac{\partial f}{\partial u}\left[\frac{\partial u}{\partial z}\right] + \frac{\partial f}{\partial r}\left[\frac{\partial r}{\partial z}\right] = \frac{\partial f}{\partial u}\frac{-t}{z^2} + \frac{\partial f}{\partial r}\left[\frac{\partial 1}{\partial x + z}\right] && \{z \mapsto u = \frac{t}{z}, z \mapsto r = \log(x + z)\} \\
&\frac{\partial f}{\partial y} = \frac{\partial f}{\partial z}\left[\frac{\partial z}{\partial y}\right] + \frac{\partial f}{\partial s}\left[\frac{\partial s}{\partial y}\right] = \frac{\partial f}{\partial z}\frac{1}{2\sqrt{y+1}} + \frac{\partial f}{\partial s}\frac{-r}{y^2} && \{y \mapsto z = \sqrt{y+1}, y \mapsto s = \frac{r}{y}\} \\
&\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y}\left[\frac{\partial y}{\partial x}\right] + \frac{\partial f}{\partial r}\left[\frac{\partial r}{\partial x}\right] = \frac{\partial f}{\partial y}2x + \frac{\partial f}{\partial r}\frac{1}{x+z} && \{x \mapsto y = x^2, x \mapsto r = \log(x + z)\}
\end{aligned}
$$

1.7.4 Feed-forward Architectures

The simplest computational graphs are purely feedforward, and corresponds to the computation of

$$f = f_{t-1} \circ f_{t-2} \circ \ldots \circ f_2 \circ f_1 \tag{1.39}$$

for functions $f_k : \mathbb{R}^{n_k} \to \mathbb{R}^{n_{k+1}}$.

The forward function evaluation algorithm initializes $x_0 = x$ and then computes

$$\forall\, k = 1, \ldots, t-1, \quad x_{k+1} = f_k(x_k)$$

where at the output, one retrieves $f(x) = x_t$.

Denoting $A_k \stackrel{\text{def.}}{=} \partial f_k(x_k) \in \mathbb{R}^{n_{k+1} \times n_k}$ the Jacobian, one has

$$\partial f(x) = A_{t-1} \times A_{t-2} \times \ldots A_2 \times A_1.$$

The forward (resp. backward) mode corresponds to the computation of the product of the Jacobian from right to left (resp. left to right)

$$
\begin{aligned}
\partial f(x) &= A_{t-1} \times (A_{t-2} \times (\ldots \times (A_3 \times (A_2 \times A_1)))), \\
\partial f(x) &= ((((A_{t-1} \times A_{t-2}) \times A_{t-3}) \times \ldots) \times A_2) \times A_1.
\end{aligned}
$$

We note that the computation of the product $A \times B$ of $A \in \mathbb{R}^{n \times p}$ with $B \in \mathbb{R}^{p \times q}$ necessitates npq operations. As shown on Figure 1.20, the complexity of the forward and backward modes are

$$n_1 \sum_{k=2}^{t-1} n_k n_{k+1} \quad \text{and} \quad n_t \sum_{k=1}^{t-2} n_k n_{k+1}$$

So if $n_t \ll n_1$ (which is the typical case in ML scenario where $n_t = 1$) then the backward mode is cheaper.

1.7.5 Multilayer Perceptron

We consider a feedforward deep network (fully connected for simplicity), initialized as $a_0 = a$ and defined through

$$\forall\, k = 1, \ldots, t, \quad \begin{cases} b_k \overset{\text{def.}}{=} W_k a_{k-1} \\ a_k \overset{\text{def.}}{=} \rho(b_k) \end{cases} \tag{1.40}$$

where ρ is a point wise non-linearity (so $\rho(u)$ actually denoted $(\rho(u_i))_i$), $W_k \in \mathbb{R}^{n_k \times n_{k-1}}$ are the weight to be trained, and the final output is set as, for $W = (W_1, \ldots, W_t)$,

$$f(W) \overset{\text{def.}}{=} \ell(b_t, y)$$

where $\ell : \mathbb{R} \times \mathbb{R} \to 0$ is some loss function. Figure 1.21 displays the computational graph.

The backward automatic differentiation method (aka back-propagation) proceeds by first computing the $(a_k)_{k=1}^t$ by forward evaluation, and then intializing the gradient vector

$$\nabla_{b_t} f = \ell'(b_t, y)$$

where ℓ' is the derivative with respect to the first variable, and then applying the adjoint on the chain rule of $a_{k-1} \mapsto b_k = W_k a_{k-1} \mapsto a_k = \rho(b_k)$

$$\begin{aligned} \nabla_{b_k} f &= \operatorname{diag}(\rho'(b_k)) \nabla_{a_k} f \\ \nabla_{a_{k-1}} f &= W_k^\top b_k \end{aligned}$$

The gradient of with respect to the weight matrices are obtained by also applying the adjoint on the chain rule, but this time to of $W_k \mapsto b_k = W_k a_{k-1}$

$$\nabla_{W_k} f = (\nabla_{b_k} f) a_{k-1}^\top.$$

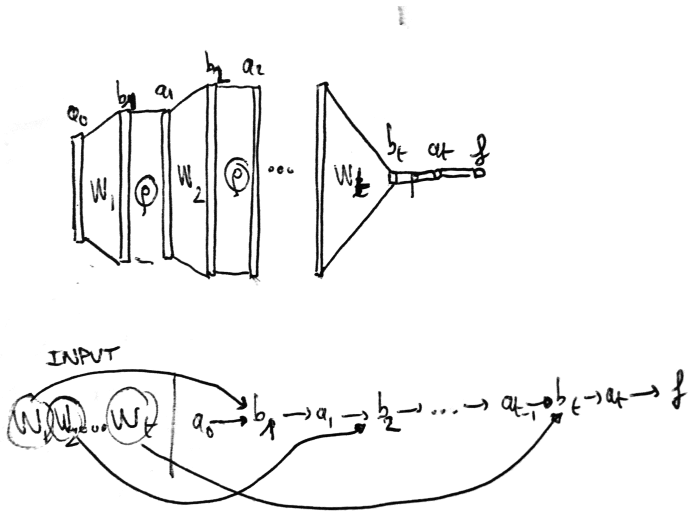


Figure 1.21: Computational graph for a MLP.

Bibliography

[1] Amir Beck. *Introduction to Nonlinear Optimization: Theory, Algorithms, and Applications with MATLAB*. SIAM, 2014.

[2] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[4] E. Candès and D. Donoho. New tight frames of curvelets and optimal representations of objects with piecewise C^2 singularities. *Commun. on Pure and Appl. Math.*, 57(2):219–266, 2004.

[5] E. J. Candès, L. Demanet, D. L. Donoho, and L. Ying. Fast discrete curvelet transforms. *SIAM Multiscale Modeling and Simulation*, 5:861–899, 2005.

[6] A. Chambolle. An algorithm for total variation minimization and applications. *J. Math. Imaging Vis.*, 20:89–97, 2004.

[7] Antonin Chambolle, Vicent Caselles, Daniel Cremers, Matteo Novaga, and Thomas Pock. An introduction to total variation for image analysis. *Theoretical foundations and numerical methods for sparse recovery*, 9(263-340):227, 2010.

[8] Antonin Chambolle and Thomas Pock. An introduction to continuous optimization for imaging. *Acta Numerica*, 25:161–319, 2016.

[9] S.S. Chen, D.L. Donoho, and M.A. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61, 1999.

[10] Philippe G Ciarlet. Introduction à l'analyse numérique matricielle et à l'optimisation. 1982.

[11] P. L. Combettes and V. R. Wajs. Signal recovery by proximal forward-backward splitting. *SIAM Multiscale Modeling and Simulation*, 4(4), 2005.

[12] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Commun. on Pure and Appl. Math.*, 57:1413–1541, 2004.

[13] D. Donoho and I. Johnstone. Ideal spatial adaptation via wavelet shrinkage. *Biometrika*, 81:425–455, Dec 1994.

[14] Heinz Werner Engl, Martin Hanke, and Andreas Neubauer. *Regularization of inverse problems*, volume 375. Springer Science & Business Media, 1996.

[15] M. Figueiredo and R. Nowak. An EM Algorithm for Wavelet-Based Image Restoration. *IEEE Trans. Image Proc.*, 12(8):906–916, 2003.

[16] Simon Foucart and Holger Rauhut. *A mathematical introduction to compressive sensing*, volume 1. Birkhäuser Basel, 2013.

[17] Stephane Mallat. *A wavelet tour of signal processing: the sparse way*. Academic press, 2008.

[18] D. Mumford and J. Shah. Optimal approximation by piecewise smooth functions and associated variational problems. *Commun. on Pure and Appl. Math.*, 42:577–685, 1989.

[19] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[20] Gabriel Peyré. *L'algèbre discrète de la transformée de Fourier*. Ellipses, 2004.

[21] J. Portilla, V. Strela, M.J. Wainwright, and Simoncelli E.P. Image denoising using scale mixtures of Gaussians in the wavelet domain. *IEEE Trans. Image Proc.*, 12(11):1338–1351, November 2003.

[22] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Phys. D*, 60(1-4):259–268, 1992.

[23] Otmar Scherzer, Markus Grasmair, Harald Grossauer, Markus Haltmeier, Frank Lenzen, and L Sirovich. *Variational methods in imaging*. Springer, 2009.

[24] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[25] Jean-Luc Starck, Fionn Murtagh, and Jalal Fadili. *Sparse image and signal processing: Wavelets and related geometric multiscale analysis*. Cambridge university press, 2015.