

Course notes on Optimization for Machine Learning

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

September 13, 2020

Abstract

This document presents first order optimization methods and their applications to machine learning. This is not a course on machine learning (in particular it does not cover modeling and statistical considerations) and it is focussed on the use and analysis of cheap methods that can scale to large datasets and models with lots of parameters. These methods are variations around the notion of “gradient descent”, so that the computation of gradients plays a major role. This course covers basic theoretical properties of optimization problems (in particular convex analysis and first order differential calculus), the gradient descent method, the stochastic gradient method, automatic differentiation, shallow and deep networks.

Contents

1 Motivation in Machine Learning	3
1.1 Unconstraint optimization	3
1.2 Regression	4
1.3 Classification	4
2 Basics of Convex Analysis	4
2.1 Existence of Solutions	4
2.2 Convexity	6
2.3 Convex Sets	8
3 Derivative and gradient	8
3.1 Gradient	8
3.2 First Order Conditions	10
3.3 Least Squares	11
3.4 Link with PCA	12
3.5 Classification	14
3.6 Chain Rule	14
4 Gradient Descent Algorithm	15
4.1 Steepest Descent Direction	15
4.2 Gradient Descent	17

5	Convergence Analysis	18
5.1	Quadratic Case	18
5.2	General Case	22
6	Regularization	28
6.1	Penalized Least Squares	28
6.2	Ridge Regression	29
6.3	Lasso	30
6.4	Iterative Soft Thresholding	32
7	Stochastic Optimization	33
7.1	Minimizing Sums and Expectation	33
7.2	Batch Gradient Descent (BGD)	35
7.3	Stochastic Gradient Descent (SGD)	35
7.4	Stochastic Gradient Descent with Averaging (SGA)	39
7.5	Stochastic Averaged Gradient Descent (SAG)	40
8	Multi-Layers Perceptron	41
8.1	MLP and its derivative	41
8.2	MLP and Gradient Computation	42
8.3	Universality	44
9	Automatic Differentiation	47
9.1	Finite Differences and Symbolic Calculus	48
9.2	Computational Graphs	48
9.3	Forward Mode of Automatic Differentiation	49
9.4	Reverse Mode of Automatic Differentiation	52
9.5	Feed-forward Compositions	54
9.6	Feed-forward Architecture	55
9.7	Recurrent Architectures	57

1 Motivation in Machine Learning

1.1 Unconstraint optimization

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

$$\inf_{x \in \mathbb{R}^p} f(x), \quad (1)$$

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

$$\min_{x \in \mathbb{R}^p} f(x). \quad (2)$$

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 .

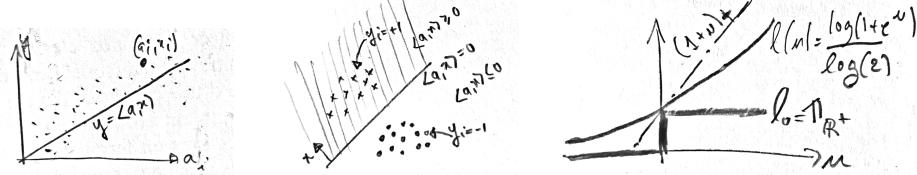


Figure 1: Left: linear regression, middle: linear classifier, right: loss function for classification.

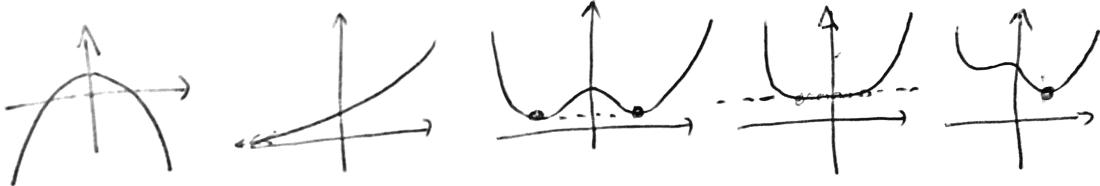


Figure 2: Left: non-existence of minimizer, middle: multiple minimizers, right: uniqueness.

1.2 Regression

For regression, $y_i \in \mathbb{R}$, in which case

$$f(x) = \frac{1}{2} \sum_{i=1}^n (y_i - \langle x, a_i \rangle)^2 = \frac{1}{2} \|Ax - y\|^2, \quad (3)$$

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

1.3 Classification

For classification, $y_i \in \{-1, 1\}$, in which case

$$f(x) = \sum_{i=1}^n \ell(-y_i \langle x, a_i \rangle) = L(-\text{diag}(y) Ax) \quad (4)$$

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

2 Basics of Convex Analysis

2.1 Existence of Solutions

In general, there might be no solution to the optimization (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 $f(x) \rightarrow +\infty$ when

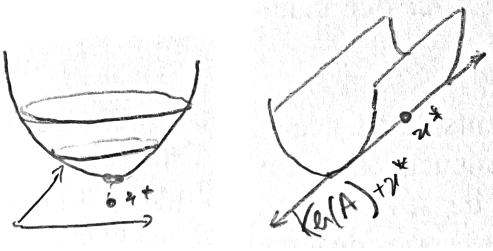


Figure 3: Coercivity condition for least squares.

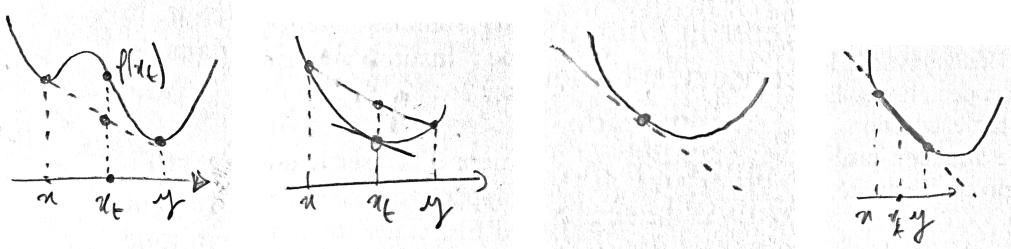


Figure 4: Convex vs. non-convex functions ; Strictly convex vs. non strictly convex functions.

$x \rightarrow +\infty$. 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

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

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 1 (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^* is a solution, then $x^* + u$ 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. 3. If ℓ is strictly convex, the same conclusion holds in the case of classification.

2.2 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 $(x, y) \in (\mathbb{R}^p)^2$,

$$\forall t \in [0, 1], \quad f((1-t)x + ty) \leq (1-t)f(x) + tf(y) \tag{5}$$

which means that the function is bellow its secant (and actually also above its tangent when this is well defined), see Fig. 4. If x^* is a local minimizer of a convex f , then x^* is a global minimizer, i.e. $x^* \in \operatorname{argmin} f$.

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 $g : \mathbb{R}^q \rightarrow \mathbb{R}$ 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 (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 (4) is itself convex.

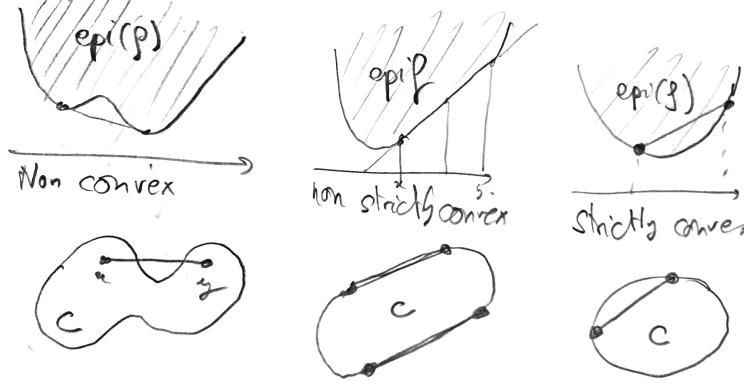


Figure 5: Comparison of convex functions $f : \mathbb{R}^p \rightarrow \mathbb{R}$ (for $p = 1$) and convex sets $C \subset \mathbb{R}^p$ (for $p = 2$).

Strict convexity. When f is convex, one can strengthen the condition (5) and impose that the inequality is strict for $t \in]0, 1[$ (see Fig. 4, right), i.e.

$$\forall t \in]0, 1[, \quad f((1-t)x + ty) < (1-t)f(x) + tf(y). \quad (6)$$

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

Example 2 (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 $A^\top Az = 0$ implies $\langle A^\top Az, z \rangle = \|Az\|^2 = 0$ i.e. $z \in \ker(A)$.

2.3 Convex Sets

A set $\Omega \subset \mathbb{R}^p$ is said to be convex if for any $(x, y) \in \Omega^2$, $(1-t)x + ty \in \Omega$ for $t \in [0, 1]$. The connexion between convex function and convex sets is that a function f is convex if and only if its epigraph $\text{epi}(f) \stackrel{\text{def.}}{=} \{(x, t) \in \mathbb{R}^{p+1} ; t \geq f(x)\}$ is a convex set.

Remark 1 (Convexity of the set of minimizers). In general, minimizers x^* might be non-unique, as shown on Figure 3. When f is convex, the set $\text{argmin}(f)$ of minimizers is itself a convex set. Indeed, if x_1^* and x_2^* are minimizers, so that in particular $f(x_1^*) = f(x_2^*) = \min(f)$, then $f((1-t)x_1^* + tx_2^*) \leq (1-t)f(x_1^*) + tf(x_2^*) = f(x_1^*) = \min(f)$, so that $(1-t)x_1^* + tx_2^*$ is itself a minimizer. Figure 5 shows convex and non-convex sets.

3 Derivative and gradient

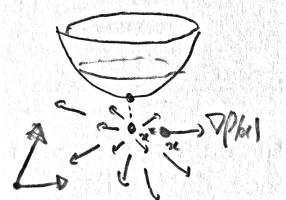
3.1 Gradient

If f is differentiable along each axis, we denote

$$\nabla f(x) \stackrel{\text{def.}}{=} \left(\frac{\partial f(x)}{\partial x_1}, \dots, \frac{\partial f(x)}{\partial x_p} \right)^\top \in \mathbb{R}^p$$

the gradient vector, so that $\nabla f : \mathbb{R}^p \rightarrow \mathbb{R}^p$ is a vector field. Here the partial derivative (when they exists) are defined as

$$\frac{\partial f(x)}{\partial x_k} \stackrel{\text{def.}}{=} \lim_{\eta \rightarrow 0} \frac{f(x + \eta \delta_k) - f(x)}{\eta}$$



where $\delta_k = (0, \dots, 0, 1, 0, \dots, 0)^\top \in \mathbb{R}^p$ is the k^{th} canonical basis vector.

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

$$f(x + \varepsilon) = f(x) + \langle \varepsilon, \nabla f(x) \rangle + o(\|\varepsilon\|). \quad (7)$$

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

Also, $\nabla f(x)$ is the only vector such that the relation (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*

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

Proof. One can write the convexity condition as

$$f((1-t)x + tx') \leq (1-t)f(x) + tf(x') \implies \frac{f(x + t(x' - x)) - f(x)}{t} \leq f(x') - f(x)$$

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

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

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

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

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

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

□

3.2 First Order Conditions

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

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

Proof. One has for ε small enough and u fixed

$$f(x^*) \leq f(x^* + \varepsilon u) = f(x^*) + \varepsilon \langle \nabla f(x^*), u \rangle + o(\varepsilon) \implies \langle \nabla f(x^*), u \rangle \geq o(1) \implies \langle \nabla f(x^*), u \rangle \geq 0.$$

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



Figure 6: Function with local maxima/minima (left), saddle point (middle) and global minimum (right).

Note that the converse is not true in general, since one might have $\nabla f(x) = 0$ but x is not a local minimum. 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. 6. Note however that in practice, if $\nabla f(x^*) = 0$ but x^* is not a local minimum, then x^* tends to be an unstable equilibrium. Thus most often a gradient-based algorithm will converge to points with $\nabla f(x^*) = 0$ that are local minimizers. The following proposition shows that a much stronger result holds if f is convex.

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

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

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

$$f(x^*) \leq f(tx + (1 - t)x^*) \leq tf(x) + (1 - t)f(x^*) \implies f(x^*) \leq f(x)$$

and thus x^* is a global minimum.

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

$$f(x) \geq f(x^*) + \langle \nabla f(x^*), x - x^* \rangle = f(x^*).$$

□

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

3.3 Least Squares

The most important gradient formula is the one of the square loss (3), which can be obtained by expanding the norm

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

Here, we have used the fact that $\|A\varepsilon\|^2 = o(\|\varepsilon\|)$ and used 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 remembered 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

$$\nabla f(x) = A^\top(Ax - y). \quad (8)$$

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

$$x^* = (A^\top A)^{-1}A^\top y. \quad (9)$$

This shows that in this case, x^* depends linearly on the data y , and the corresponding linear operator $(A^\top A)^{-1}A^*$ 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

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

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

3.4 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 $m \stackrel{\text{def.}}{=} \frac{1}{n} \sum_{i=1}^n a_i \in \mathbb{R}^p$ 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 $u \in \mathbb{R}^p$, $\langle Cu, u \rangle / n \geq 0$ 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, $UU^\top = U^\top U = \text{Id}_p$, and equivalently $U^{-1} = U^\top$. The diagonalization property can be conveniently written as $\frac{1}{n}C = U \text{diag}(\lambda_k)U^\top$. One can thus re-write the covariance quadratic form in the basis U as being a separable sum of p squares

$$\frac{1}{n} \langle Cx, x \rangle = \langle U \text{diag}(\lambda_k)U^\top x, x \rangle = \langle \text{diag}(\lambda_k)(U^\top x), (U^\top x) \rangle = \sum_{k=1}^p \lambda_k \langle x, u_k \rangle^2. \quad (10)$$

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 $\lambda_k \geq 0$ are positive.

If one assumes that the eigenvalues are ordered $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_p$, 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 (see next paragraph), 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 ($m = 3$).

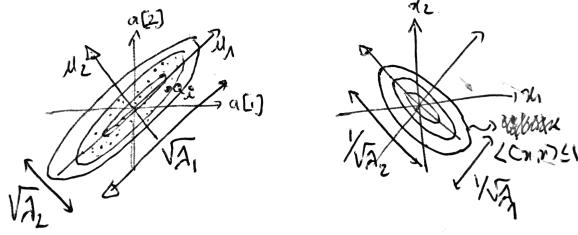


Figure 7: Left: point clouds $(a_i)_i$ with associated PCA directions, right: quadratic part of $f(x)$.

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 $\{x ; \frac{1}{n}\langle Cx, x\rangle \leq 1\}$ has the same main axes, but the widths are the inverse $1/\sqrt{\lambda_k}$. Figure 7 shows this in dimension $p = 2$.

3.5 Classification

We can do a similar computation for the gradient of the classification loss (4). Assuming that L is differentiable, and using the Taylor expansion (7) at point $-\text{diag}(y)Ax$, one has

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

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

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

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

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

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).

3.6 Chain Rule

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

$$\nabla(g \circ B) = B^\top \circ \nabla g \circ B \tag{11}$$

where “ \circ ” denotes the composition of functions.

To generalize this to composition of possibly non-linear functions, one needs to use the notion of differential. For a function $F : \mathbb{R}^p \rightarrow \mathbb{R}^q$, its differentiable at x is a linear operator $\partial F(x) : \mathbb{R}^p \rightarrow \mathbb{R}^q$, i.e. it can

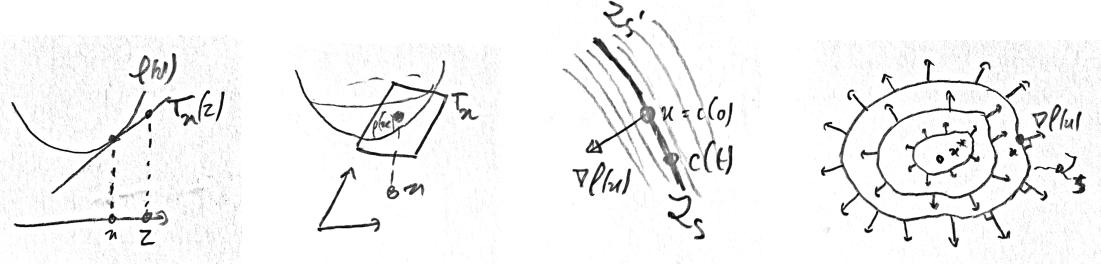


Figure 8: Left: First order Taylor expansion in 1-D and 2-D. Right: orthogonality of gradient and level sets and schematic of the proof.

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 $F(x) = (F_1(x), \dots, F_q(x))$,

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

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

$$F(x + \varepsilon) = F(x) + [\partial F(x)](\varepsilon) + o(\|\varepsilon\|). \quad (12)$$

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 $f : \mathbb{R}^p \rightarrow \mathbb{R}$, 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 (12) and (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 $H : \mathbb{R}^p \rightarrow \mathbb{R}^q$ and $G = g : \mathbb{R}^q \mapsto \mathbb{R}$, then $f = g \circ H : \mathbb{R}^p \rightarrow \mathbb{R}$ 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 (11).

4 Gradient Descent Algorithm

4.1 Steepest Descent Direction

The Taylor expansion (7) computes an affine approximation of the function f near x , since it can be written as

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

see Fig. 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 at the function f along the 1-D half line

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

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 sets 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. 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, let's 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. 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 $r \rightarrow 0$ (so for very small step this is locally the best choice), more precisely,

$$\frac{1}{r} \underset{\|u\|=r}{\operatorname{argmin}} f(x+u) \xrightarrow{r \rightarrow 0} -\frac{\nabla f(x)}{\|\nabla f(x)\|}.$$

Indeed, introducing a Lagrange multiplier $\lambda \in \mathbb{R}$ for this constraint optimization problem, one obtains that the optimal u satisfies $\nabla f(x+u) = \lambda u$ and $\|u\| = r$. Thus $\frac{u}{r} = \pm \frac{\nabla f(x+u)}{\|\nabla f(x+u)\|}$, and assuming that ∇f is continuous, when $\|u\| = r \rightarrow 0$, this converges to $\frac{u}{\|u\|} = \pm \frac{\nabla f(x)}{\|\nabla f(x)\|}$. The sign \pm should be $+1$ to obtain a maximizer and -1 for the minimizer.

4.2 Gradient Descent

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

$$x_{k+1} \stackrel{\text{def.}}{=} x_k - \tau_k \nabla f(x_k) \tag{13}$$

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 uses a fix step size $\tau_k = \tau$, or try to adapt τ_k at each iteration (see Fig. 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.

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



Figure 9: Influence of τ on the gradient descent (left) and optimal step size choice (right).

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

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

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 (13). 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. 9).

5 Convergence Analysis

5.1 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

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

We already saw that in (9) if $\ker(A) = \{0\}$, which is equivalent to C being invertible, then there exists a single global minimizer $x^* = (A^\top A)^{-1} A^\top y = C^{-1} u$.

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 (10)).

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

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

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

$$\|x_k - x^*\| \leq \tilde{\rho}^\ell \|x_0 - x^*\|. \quad (14)$$

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

$$\tau_\ell = \frac{2}{L + \mu} \implies \tilde{\rho} \stackrel{\text{def.}}{=} \frac{L - \mu}{L + \mu} = 1 - \frac{2\varepsilon}{1 + \varepsilon} \quad \text{where} \quad \varepsilon \stackrel{\text{def.}}{=} \mu/L. \quad (15)$$

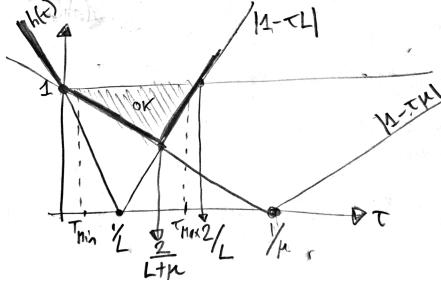


Figure 10: Contraction constant $h(\tau)$ for a quadratic function (right).

Proof. One iterate of gradient descent reads

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

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

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

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

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

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 $S = U \text{diag}(\lambda_k(S)) U^\top$, and $S^\top S = S^2 = U \text{diag}(\lambda_k(S)^2) U^\top$ so that

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

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 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 $\tau^* = \frac{2}{L+\mu}$ and then

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

□

Note that when the condition number $\xi \stackrel{\text{def.}}{=} \mu/L \ll 1$ is small (which is the typical setup for ill-posed problems), then the contraction constant appearing in (15) scales like

$$\tilde{\rho} \sim 1 - 2\xi. \tag{16}$$

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 (14), although it is geometrical $O(\rho^k)$ is called a “linear rate” in the optimization literature. It is a “global” rate because it holds for all k (and not only for large enough k).

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 extend 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 eigenvalues of C are bounded by L , then if $0 < \tau_k = \tau < 2/L$ is constant, then*

$$f(x_k) - f(x^*) \leq \frac{\text{dist}(x_0, \arg\min f)^2}{\tau k}.$$

where

$$\text{dist}(x_0, \arg\min f) \stackrel{\text{def.}}{=} \min_{x^* \in \arg\min f} \|x_0 - x^*\|.$$

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

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

Now one has

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

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

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

where we have used the fact that

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

This thus implies

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

where we have denoted

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

One has

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

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

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

Indeed, one has

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

□

5.2 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. If the function is twice differentiable along the axes, the hessian matrix is

$$(\partial^2 f)(x) = \left(\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \right)_{1 \leq i, j \leq p} \in \mathbb{R}^{p \times p}.$$

Where recall that $\frac{\partial^2 f(x)}{\partial x_i \partial x_j}$ is the differential along direction x_j of the function $x \mapsto \frac{\partial f(x)}{\partial x_i}$. We also recall that $\frac{\partial^2 f(x)}{\partial x_i \partial x_j} = \frac{\partial^2 f(x)}{\partial x_j \partial x_i}$ so that $\partial^2 f(x)$ is a symmetric matrix.

A differentiable function f is said to be twice differentiable at x if

$$f(x + \varepsilon) = f(x) + \langle \nabla f(x), \varepsilon \rangle + \frac{1}{2} \langle \partial^2 f(x) \varepsilon, \varepsilon \rangle + o(\|\varepsilon\|^2). \quad (17)$$

This means that one can approximate f near x by a quadratic function. The hessian matrix is uniquely determined by this relation, so that if one is able to write down an expansion with some matrix H

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

then equating this with the expansion (17) ensure that $\partial^2 f(x) = H$. This is thus a way to actually determine the hessian without computing all the p^2 partial derivative. This Hessian can equivalently 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

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

and thus

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

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

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

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 whether a point x^* with $\nabla f(x^*)$ is a local minimum or not. Indeed, if $\partial^2 f(x^*)$ is a positive matrix (i.e. its eigenvalues are strictly positive), then x^* is a strict local minimum. Note that if $\partial^2 f(x^*)$ 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^* is a local minimum then $\partial^2 f(x)$

Remark 6 (Second order algorithms). 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 $H_k = \tau_k \text{Id}_p$, 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 . Roughly speaking, the theoretical analysis of the gradient descent for a generic function is obtained by applying this approximation and using the proofs of the previous section.

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.

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

In order to obtain fast convergence of the iterates themselves, 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

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

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}_μ) imply

$$\forall (x, x'), \quad f(x') + \langle \nabla f(x), x' - x \rangle + \frac{\mu}{2} \|x - x'\|^2 \leq f(x) \leq f(x') + \langle \nabla f(x'), x' - x \rangle + \frac{L}{2} \|x - x'\|^2. \quad (18)$$

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

$$\forall x, \quad \mu \text{Id}_p \preceq \partial^2 f(x) \preceq L \text{Id}_p \quad (19)$$

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.

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

Proof. We prove (18), using Taylor expansion with integral remain

$$f(x') - f(x) = \int_0^1 \langle \nabla f(x_t), x' - x \rangle dt = \langle \nabla f(x), x' - x \rangle + \int_0^1 \langle \nabla f(x_t) - \nabla f(x), x' - x \rangle dt$$

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

$$f(x') - f(x) \leq \langle \nabla f(x), x' - x \rangle + \int_0^1 L \|x_t - x\| \|x' - x\| dt \leq \langle \nabla f(x), x' - x \rangle + L \|x' - x\|^2 \int_0^1 t dt$$

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

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

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

The relation (18) shows that a smooth (resp. strongly convex) functional is bounded by bellow (resp. above) by a quadratic tangential majorant (resp. minorant).

Condition (19) 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 $\|\partial^2 f(x)\|_{\text{op}} \leq L$ 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*

$$0 < \tau_{\min} \leq \tau_\ell \leq \tau_{\max} < \frac{2}{L},$$

then x_k converges to a solution x^ of (1) and there exists $C > 0$ such that*

$$f(x_k) - f(x^*) \leq \frac{C}{\ell + 1}. \quad (20)$$

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

Proof. In the case where f is not strongly convex, we only prove (20) since the proof that x_k converges is more technical. Note indeed that if the minimizer x^* 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 (18), which reads

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

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

$$f(x_{k+1}) \leq f(x_k) - \frac{1}{L} \|\nabla f(x_k)\|^2 + \frac{1}{2L} \|\nabla f(x_k)\|^2 \leq f(x_k) - \frac{1}{2L} \|\nabla f(x_k)\|^2 \quad (21)$$

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

$$f(x_k) + \langle \nabla f(x_k), x^* - x_k \rangle \leq f(x^*)$$

and plugging this in (21) shows

$$f(x_{k+1}) \leq f(x^*) - \langle \nabla f(x_k), x^* - x_k \rangle - \frac{1}{2L} \|\nabla f(x_k)\|^2 \quad (22)$$

$$= f(x^*) + \frac{L}{2} \left(\|x_k - x^*\|^2 - \|x_k - x^* - \frac{1}{L} \nabla f(x_k)\|^2 \right) \quad (23)$$

$$= f(x^*) + \frac{L}{2} (\|x_k - x^*\|^2 - \|x^* - x_{k+1}\|^2). \quad (24)$$

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

$$\sum_{\ell=0}^k f(x_{k+1}) - (k+1)f(x^*) \leq \frac{L}{2} (\|x_0 - x^*\|^2 - \|x^{(k+1)} - x^*\|^2)$$

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

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

which gives (20) for $C \stackrel{\text{def.}}{=} L\|x_0 - x^*\|^2/2$.

If we now assume f is μ -strongly convex, then, using $\nabla f(x^*) = 0$, one has $\frac{\mu}{2}\|x^* - x\|^2 \leq f(x) - f(x^*)$ for all x . Re-manipulating (24) gives

$$\frac{\mu}{2}\|x_{k+1} - x^*\|^2 \leq f(x_{k+1}) - f(x^*) \leq \frac{L}{2}(\|x_k - x^*\|^2 - \|x^* - x_{k+1}\|^2),$$

and hence

$$\|x_{k+1} - x^*\| \leq \sqrt{\frac{L}{L+\mu}}\|x_k - x^*\|, \quad (25)$$

which is the desired result. \square

Note that in the low conditioning setting $\varepsilon \ll 1$, one retrieve a dependency of the rate (25) similar to the one of quadratic functions (16), indeed

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

6 Regularization

When the number n of sample is not large enough with respect to the dimension p of the model, it makes sense to regularize the empirical risk minimization problem.

6.1 Penalized Least Squares

For the sake of simplicity, we focus here on regression and consider

$$\min_{x \in \mathbb{R}^p} f_\lambda(x) \stackrel{\text{def.}}{=} \frac{1}{2}\|Ax - y\|^2 + \lambda R(x) \quad (26)$$

where $R(x)$ is the regularizer and $\lambda \geq 0$ the regularization parameter. The regularizer enforces some prior knowledge on the weight vector x (such as small amplitude or sparsity, as we detail next) and λ needs to be tuned using cross-validation.

We assume for simplicity that R is positive and coercive, i.e. $R(x) \rightarrow +\infty$ as $\|x\| \rightarrow +\infty$. The following proposition that in the small λ limit, the regularization select a sub-set of the possible minimizer. This is especially useful when $\ker(A) \neq 0$, i.e. the equation $Ax = y$ has an infinite number of solutions.

Proposition 7. *If $(x_{\lambda_k})_k$ is a sequence of minimizers of f_λ , then this sequence is bounded, and any accumulation x^* is a solution of the constrained optimization problem*

$$\min_{Ax=y} R(x). \quad (27)$$

Proof. Let x_0 be so that $Ax_0 = y$, then by optimality of x_{λ_k}

$$\frac{1}{2}\|Ax_{\lambda_k} - y\|^2 + \lambda_k R(x_{\lambda_k}) \leq \lambda_k R(x_0). \quad (28)$$

Since all the term are positive, one has $R(x_{\lambda_k}) \leq R(x_0)$ so that $(x_{\lambda_k})_k$ is bounded by coercivity of R . Then also $\|Ax_{\lambda_k} - y\| \leq \lambda_k R(x_0)$, and passing to the limit, one obtains $Ax^* = y$. And passing to the limit in $R(x_{\lambda_k}) \leq R(x_0)$ one has $R(x^*) \leq R(x_0)$ which shows that x^* is a solution of (27). \square

6.2 Ridge Regression

Ridge regression is by far the most popular regularizer, and corresponds to using $R(x) = \|x\|_{\mathbb{R}^p}^2$. Since it is strictly convex, the solution of (26) is unique

$$x_\lambda \stackrel{\text{def.}}{=} \underset{x \in \mathbb{R}^p}{\operatorname{argmin}} f_\lambda(x) = \frac{1}{2} \|Ax - y\|_{\mathbb{R}^n}^2 + \lambda \|x\|_{\mathbb{R}^p}^2.$$

One has

$$\nabla f_\lambda(x) = A^\top(Ax_\lambda - y) + \lambda x_\lambda = 0$$

so that x_λ depends linearly on y and can be obtained by solving a linear system. The following proposition shows that there are actually two alternate formulae.

Proposition 8. *One has*

$$x_\lambda = (A^\top A + \lambda \operatorname{Id}_p)^{-1} A^\top y, \quad (29)$$

$$= A^\top (AA^\top + \lambda \operatorname{Id}_n)^{-1} y. \quad (30)$$

Proof. Denoting $B \stackrel{\text{def.}}{=} (A^\top A + \lambda \operatorname{Id}_p)^{-1} A^\top$ and $C \stackrel{\text{def.}}{=} A^\top (AA^\top + \lambda \operatorname{Id}_n)^{-1}$, one has $(A^\top A + \lambda \operatorname{Id}_p)B = A^\top$ while

$$(A^\top A + \lambda \operatorname{Id}_p)C = (A^\top A + \lambda \operatorname{Id}_p)A^\top (AA^\top + \lambda \operatorname{Id}_n)^{-1} = A^\top (AA^\top + \lambda \operatorname{Id}_n)(AA^\top + \lambda \operatorname{Id}_n)^{-1} = A^\top.$$

Since $A^\top A + \lambda \operatorname{Id}_p$ is invertible, this gives the desired result. \square

The solution of these linear systems can be computed using either a direct method such as Cholesky factorization or an iterative method such as a conjugate gradient (which is vastly superior to the vanilla gradient descent scheme).

If $n > p$, then one should use (29) while if $n < p$ one should rather use (30).

Pseudo-inverse. As $\lambda \rightarrow 0$, then $x_\lambda \rightarrow x_0$ which is, using (27)

$$\underset{Ax=y}{\operatorname{argmin}} \|x\|.$$

If $\ker(A) = \{0\}$ (overdetermined setting), $A^\top A \in \mathbb{R}^{p \times p}$ is an invertible matrix, and $(A^\top A + \lambda \operatorname{Id}_p)^{-1} \rightarrow (A^\top A)^{-1}$, so that

$$x_0 = A^+ y \quad \text{where} \quad A^+ \stackrel{\text{def.}}{=} (A^\top A)^{-1} A^\top.$$

Conversely, if $\ker(A^\top) = \{0\}$, or equivalently $\operatorname{Im}(A) = \mathbb{R}^n$ (underdetermined setting) then one has

$$x_0 = A^+ y \quad \text{where} \quad A^+ \stackrel{\text{def.}}{=} A^\top (AA^\top)^{-1}.$$

In the special case $n = p$ and A is invertible, then both definitions of A^+ coincide, and $A^+ = A^{-1}$. In the general case (where A is neither injective nor surjective), A^+ can be computed using the Singular Value Decomposition (SVD). The matrix A^+ is often called the Moore-Penrose pseudo-inverse.

6.3 Lasso

The Lasso corresponds to using a ℓ^1 penalty

$$R(x) = \|x\|_1 \stackrel{\text{def.}}{=} \sum_{k=1}^p |x_k|.$$

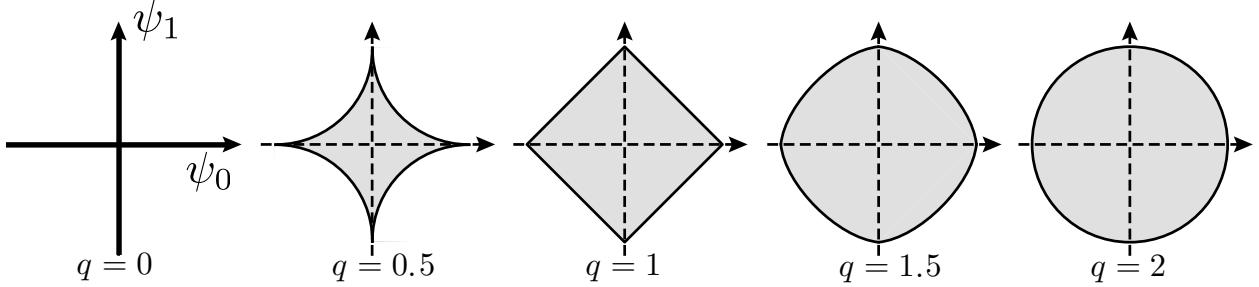


Figure 11: ℓ^q balls $\{x ; \sum_k |x_k|^q \leq 1\}$ for varying q .

The underlying idea is that solutions x_λ of a Lasso problem

$$x_\lambda \in \operatorname{argmin}_{x \in \mathbb{R}^p} f_\lambda(x) = \frac{1}{2} \|Ax - y\|_{\mathbb{R}^n}^2 + \lambda \|x\|_1$$

are sparse, i.e. solutions x_λ (which might be non-unique) have many zero entries. To get some insight about this, Fig. 11 display the ℓ^q “balls” which shrink toward the axes as $q \rightarrow 0$ (thus enforcing more sparsity) but are non-convex for $q < 1$.

This can serve two purposes: (i) one knows before hand that the solution is expected to be sparse, which is the case for instance in some problems in imaging, (ii) one want to perform model selection by pruning some of the entries in the feature (to have simpler predictor, which can be computed more efficiently at test time, or that can be more interpretable). For typical ML problems though, the performance of the Lasso predictor is usually not better than the one obtained by Ridge.

Minimizing $f(x)$ is still a convex problem, but R is non-smooth, so that one cannot use a gradient descent. Section 6.4 shows how to modify the gradient descent to cope with this issue. In general, solutions x_λ cannot be computed in closed form, excepted when the design matrix A is orthogonal.

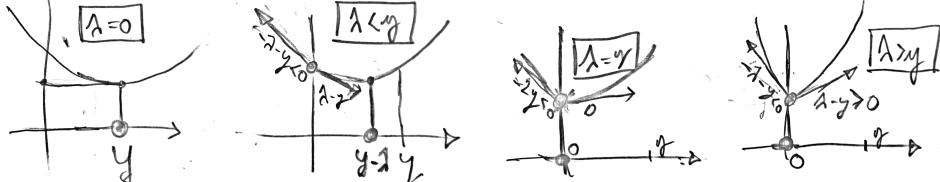


Figure 12: Evolution with λ of the function $F(x) \stackrel{\text{def.}}{=} \frac{1}{2} \|x - y\|^2 + \lambda |x|$.

Proposition 9. When $n = p$ and $A = \operatorname{Id}_n$, one has

$$\operatorname{argmin}_{x \in \mathbb{R}^p} \frac{1}{2} \|x - y\|^2 + \lambda \|x\| = S_\lambda(x) \quad \text{where} \quad S_\lambda(x) = (\operatorname{sign}(x_k) \max(|x_k| - \lambda, 0))_k$$

Proof. One has $f_\lambda(x) = \sum_k \frac{1}{2} (x_k - y_k)^2 + \lambda |x_k|$, so that one needs to find the minimum of the 1-D function $x \in \mathbb{R} \mapsto \frac{1}{2} (x - y)^2 + \lambda |x|$. We can do this minimization “graphically” as shown on Fig. 12. For $x > 0$, one has $F'(x) = x - y + \lambda$ which is 0 at $x = y - \lambda$. The minimum is at $x = y - \lambda$ for $\lambda \leq y$, and stays at 0 for all $\lambda > y$. The problem is symmetric with respect to the switch $x \mapsto -x$. \square

Here, S_λ is the celebrated soft-thresholding non-linear function.

6.4 Iterative Soft Thresholding

We now derive an algorithm using a classical technic of surrogate function minimization. We aim at minimizing

$$f(x) \stackrel{\text{def.}}{=} \frac{1}{2} \|y - Ax\|^2 + \lambda \|x\|_1$$

and we introduce for any fixed x' the function

$$f_\tau(x, x') \stackrel{\text{def.}}{=} f(x) - \frac{1}{2} \|Ax - Ax'\|^2 + \frac{1}{2\tau} \|x - x'\|^2.$$

We notice that $f_\tau(x, x) = 0$ and one the quadratic part of this function reads

$$K(x, x') \stackrel{\text{def.}}{=} -\frac{1}{2} \|Ax - Ax'\|^2 + \frac{1}{2\tau} \|x - x'\|^2 = \frac{1}{2} \left(\frac{1}{\tau} \text{Id}_N - A^\top A \right) (x - x'), x - x' \rangle.$$

This quantity $K(x, x')$ is positive if $\lambda_{\max}(A^\top A) \leqslant 1/\tau$ (maximum eigenvalue), i.e. $\tau \leqslant 1/\|A\|_{\text{op}}^2$, where we recall that $\|A\|_{\text{op}} = \sigma_{\max}(A)$ is the operator (algebra) norm. This shows that $f_\tau(x, x')$ is a valid surrogate functional, in the sense that

$$f(x) \leqslant f_\tau(x, x'), \quad f_\tau(x, x') = 0, \quad \text{and} \quad f(\cdot) - f_\tau(\cdot, x') \text{ is smooth.}$$

We also note that this majorant $f_\tau(\cdot, x')$ is convex. This leads to define

$$x_{k+1} \stackrel{\text{def.}}{=} \underset{x}{\operatorname{argmin}} f_\tau(x, x_k) \tag{31}$$

which by construction satisfies

$$f(x_{k+1}) \leqslant f(x_k).$$

Proposition 10. *The iterates x_k defined by (31) satisfy*

$$x_{k+1} = S_{\lambda\tau} (x_k - \tau A^\top (Ax_k - y)) \tag{32}$$

where $S_\lambda(x) = (s_\lambda(x_m))_m$ where $s_\lambda(r) = \text{sign}(r) \max(|r| - \lambda, 0)$ is the soft thresholding operator.

Proof. One has

$$\begin{aligned} f_\tau(x, x') &= \frac{1}{2} \|Ax - y\|^2 - \frac{1}{2} \|Ax - Ax'\|^2 + \frac{1}{2\tau} \|x - x'\|^2 + \lambda \|x\|_1 \\ &= C + \frac{1}{2} \|Ax\|^2 - \frac{1}{2} \|Ax\|^2 + \frac{1}{2\tau} \|x\|^2 - \langle Ax, y \rangle + \langle Ax, Ax' \rangle - \frac{1}{\tau} \langle x, x' \rangle + \lambda \|x\|_1 \\ &= C + \frac{1}{2\tau} \|x\|^2 + \langle x, -A^\top y + AA^\top x' - \frac{1}{\tau} x' \rangle + \lambda \|x\|_1 \\ &= C' + \frac{1}{\tau} \left(\frac{1}{2} \|x - (x' - \tau A^\top (Ax' - y))\|^2 + \tau \lambda \|x\|_1 \right) \end{aligned}$$

Proposition (9) shows that the minimizer of $f_\tau(x, x')$ is thus indeed $S_{\lambda\tau}(x' - \tau A^\top (Ax' - y))$ as claimed. \square

Equation (32) defines the iterative soft-thresholding algorithm. It follows from a valid convex surrogate function if $\tau \leqslant 1/\|A\|^2$, but one can actually shows that it converges to a solution of the Lasso as soon as $\tau < 2/\|A\|^2$, which is exactly as for the classical gradient descent.

7 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.

7.1 Minimizing Sums and Expectation

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

$$\min_{x \in \mathbb{R}^p} f(x) \stackrel{\text{def.}}{=} \frac{1}{n} \sum_{i=1}^n f_i(x) \quad (33)$$

or even expectations of the form

$$\min_{x \in \mathbb{R}^p} f(x) \stackrel{\text{def.}}{=} \mathbb{E}_{\mathbf{z} \sim \pi}(f(x, \mathbf{z})) = \int_{\mathcal{Z}} f(x, z) d\pi(z). \quad (34)$$

Problem (33) can be seen as a special case of (34), 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 (33) as a discretized ‘‘empirical’’ version of (34) when drawing $(z_i)_i$ i.i.d. according to \mathbf{z} and defining $f_i(x) = f(x, z_i)$. In this setup, (33) converges to (34) as $n \rightarrow +\infty$.

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

$$f_i(x) = \ell(\langle a_i, x \rangle, y_i) \quad \text{and} \quad f(x, z) = \ell(\langle a, x \rangle, y) \quad (35)$$

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

$$L(s, y) \stackrel{\text{def.}}{=} \log(1 + \exp(-sy)). \quad (36)$$

But this extends to arbitrary parametric models, and in particular deep neural networks.

While some algorithms (in particular batch gradient descent) are specific to finite sums (33), the stochastic methods we detail next work verbatim (with the same convergence guarantees) in the expectation case (34). 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 .

If the functions $f_i(x)$ are very similar (the extreme case being that they are all equal), then of course there is a gain in using stochastic optimization (since in this case, $\nabla f_i \approx \nabla f$ but ∇f_i is n times cheaper). But in general stochastic optimization methods are not necessarily faster than batch gradient descent. If n is not too large so that one afford the price of doing a few non-stochastic iterations, then deterministic methods can be 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 deterministic iterations in smaller chunks. Another advantage is that they are quite easy to parallelize.

7.2 Batch Gradient Descent (BGD)

The usual deterministic (batch) gradient descent (BGD) is studied in details in Section 4. Its iterations read

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

and the step size should be chosen as $0 < \tau_{\min} < \tau_k < \tau_{\max} \stackrel{\text{def.}}{=} 2/L$ 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

$$\nabla f(x) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(x) \quad (37)$$

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

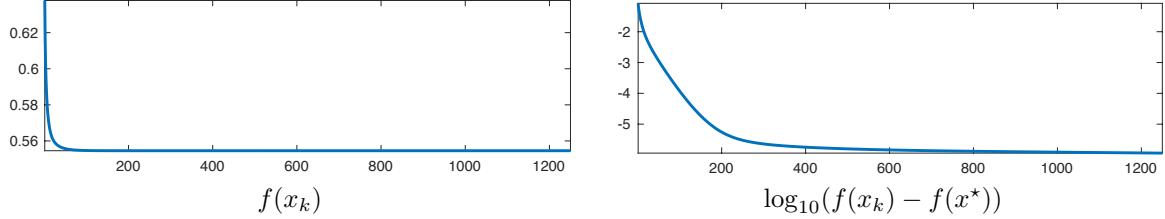


Figure 13: Evolution of the error of the BGD for logistic classification.

For ERM-type functions of the form (35), one can do the Taylor expansion of f_i

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

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

$$\nabla f_i(x) = \ell'(\langle a_i, x \rangle, y_i) a_i. \quad (38)$$

For the logistic loss, one has

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

7.3 Stochastic Gradient Descent (SGD)

For very large n , computing the full gradient ∇f as in (37) 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

$$\mathbb{E}_{\mathbf{i}} \nabla f_{\mathbf{i}}(x) = \nabla f(x) \quad (39)$$

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

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 $\{1, \dots, n\}$. It is important that the iterates x_{k+1} are thus random vectors, and the theoretical analysis of the method thus studies whether 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 situations, 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 tend 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 $k \rightarrow +\infty$. We thus propose to use

$$\tau_k \stackrel{\text{def.}}{=} \frac{\tau_0}{1 + k/k_0} \quad (40)$$

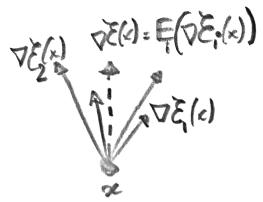


Figure 14: Unbiased gradient estimate

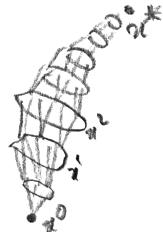


Figure 15:
Schematic view
of SGD iterates

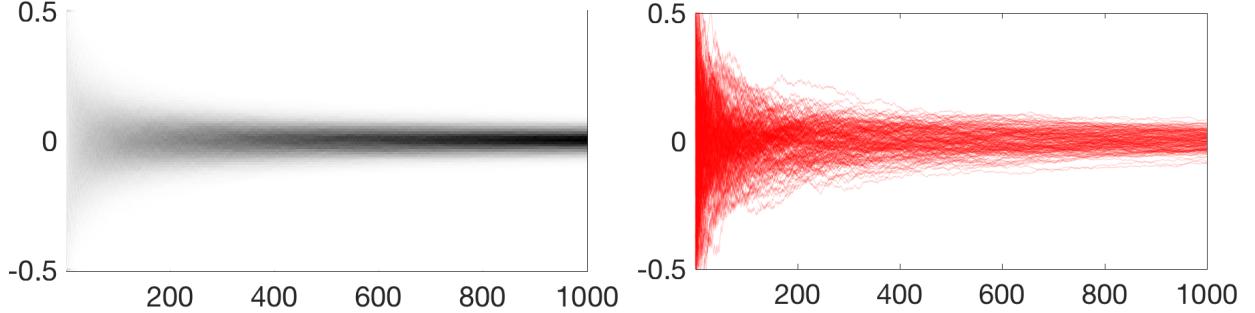


Figure 16: Display of a large number of trajectories $k \mapsto x_k \in \mathbb{R}$ generated by several runs of SGD. On the top row, each curve is a trajectory, and the bottom row displays the corresponding density.

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

Figure 16 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 $x^* = 0$. 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}_μ) (i.e. $\text{Id}_p \preceq \partial^2 f(x)$ if f is C^2), and is such that $\|\nabla f_i(x)\|^2 \leq C^2$. For the step size choice $\tau_k = \frac{1}{\mu(k+1)}$, one has*

$$\mathbb{E}(\|x_k - x^*\|^2) \leq \frac{R}{k+1} \quad \text{where} \quad R = \max(\|x_0 - x^*\|, C^2/\mu^2), \quad (41)$$

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

Proof. By strong convexity, one has

$$\begin{aligned} f(x^*) - f(x_k) &\geq \langle \nabla f(x_k), x^* - x_k \rangle + \frac{\mu}{2} \|x_k - x^*\|^2 \\ f(x_k) - f(x^*) &\geq \langle \nabla f(x^*), x_k - x^* \rangle + \frac{\mu}{2} \|x_k - x^*\|^2. \end{aligned}$$

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

$$\langle \nabla f(x_k) - \nabla f(x^*), x_k - x^* \rangle = \langle \nabla f(x_k), x_k - x^* \rangle \geq \mu \|x_k - x^*\|^2. \quad (42)$$

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

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

where we used the fact (39) that the gradient is unbiased. Taking now the full expectation with respect to all the other previous iterates, and using (42) one obtains

$$\mathbb{E}(\|x_{k+1} - x^*\|^2) \leq \mathbb{E}(\|x_k - x^*\|^2) - 2\mu\tau_k \mathbb{E}(\|x_k - x^*\|^2) + \tau_k^2 C^2 = (1 - 2\mu\tau_k) \mathbb{E}(\|x_k - x^*\|^2) + \tau_k^2 C^2. \quad (43)$$

We show by recursion that the bound (41) holds. We denote $\varepsilon_k \stackrel{\text{def.}}{=} \mathbb{E}(\|x_k - x^*\|^2)$. Indeed, for $k = 0$, this it is true that

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

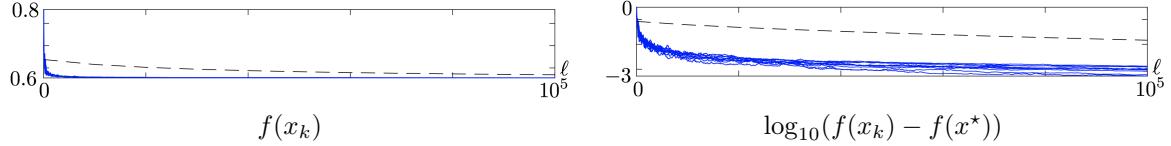


Figure 17: Evolution of the error of the SGD for logistic classification (dashed line shows BGD).

We now assume that $\varepsilon_k \leq \frac{R}{k+1}$. Using (43) in the case of $\tau_k = \frac{1}{\mu(k+1)}$, one has, denoting $m = k + 1$

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

□

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 17 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.

7.4 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

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

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

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

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 .

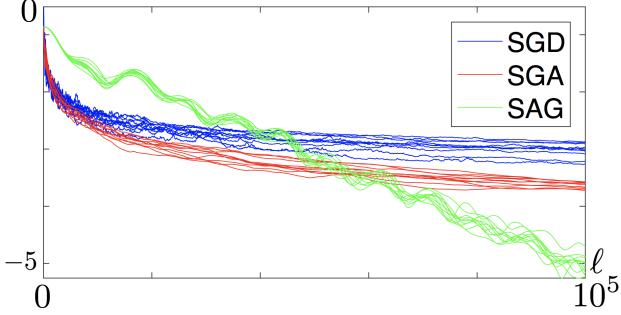


Figure 18: Evolution of $\log_{10}(f(x_k) - f(x^*))$ for SGD, SGA and SAG.

7.5 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.

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

for some $0 < \rho < 1$.

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 (34)).

Figure 18 shows a comparison of SGD, SGA and SAG.

8 Multi-Layers Perceptron

In this section, we study the simplest example of non-linear parametric models, namely Multi-Layers Perceptron (MLP) with a single hidden layer (so they have in total 2 layers). Perceptron (with no hidden layer) corresponds to the linear models studied in the previous sections. MLP with more layers are obtained by stacking together several such simple MLP, and are studied in Section ??, since the computation of their derivatives is very suited to automatic-differentiation methods.

8.1 MLP and its derivative

The basic MLP $a \mapsto h_{W,u}(a)$ takes as input a feature vector $a \in \mathbb{R}^p$, computes an intermediate hidden representation $b = Wa \in \mathbb{R}^q$ using q “neurons” stored as the rows $w_k \in \mathbb{R}^p$ of the weight matrix $W \in \mathbb{R}^{q \times p}$,

passes these through a non-linearity $\rho : \mathbb{R} \rightarrow \mathbb{R}$, i.e. $\rho(b) = (\rho(b_k))_{k=1}^q$ and then outputs a scalar value as a linear combination with output weights $u \in \mathbb{R}^q$, i.e.

$$h_{W,u}(a) = \langle \rho(Wa), u \rangle = \sum_{k=1}^q u_k \rho((Wa)_k) = \sum_{k=1}^q u_k \rho(\langle a, w_k \rangle).$$

This function $h_{W,u}(\cdot)$ is thus a weighted sum of q “ridge functions” $\rho(\langle \cdot, w_k \rangle)$. These functions are constant in the direction orthogonal to the neuron w_k and have a profile defined by ρ .

The most popular non-linearities are sigmoid functions such as

$$\rho(r) = \frac{e^r}{1 + e^r} \quad \text{and} \quad \rho(r) = \frac{1}{\pi} \operatorname{atan}(r) + \frac{1}{2}$$

and the rectified linear unit (ReLU) function $\rho(r) = \max(r, 0)$.

One often add a bias term in these models, and consider functions of the form $\rho(\langle \cdot, w_k \rangle + z_k)$ but this bias term can be integrated in the weight as usual by considering $(\langle a, w_k \rangle + z_k = \langle (a, 1), (w_k, z_k) \rangle)$, so we ignore it in the following section. This simply amount to replacing $a \in \mathbb{R}^p$ by $(a, 1) \in \mathbb{R}^{p+1}$ and adding a dimension $p \mapsto p + 1$, as a pre-processing of the features.

Expressiveness. In order to define function of arbitrary complexity when q increases, it is important that ρ is non-linear. Indeed, if $\rho(s) = s$, then $h_{W,u}(a) = \langle Wa, u \rangle = \langle a, W^\top u \rangle$. It is thus a linear function with weights $W^\top u$, whatever the number q of neurons. Similarly, if ρ is a polynomial on \mathbb{R} of degree d , then $h_{W,u}(\cdot)$ is itself a polynomial of degree d in \mathbb{R}^p , which is a linear space V of finite dimension $\dim(V) = O(p^d)$. So even if q increases, the dimension $\dim(V)$ stays fixed and $h_{W,u}(\cdot)$ cannot approximate an arbitrary function outside V . In sharp contrast, one can show that if ρ is not polynomial, then $h_{W,u}(\cdot)$ can approximate any continuous function, as studied in Section 8.3.

8.2 MLP and Gradient Computation

Given pairs of features and data values $(a_i, y_i)_{i=1}^n$, and as usual storing the features in the rows of $A \in \mathbb{R}^{n \times p}$, we consider the following least square regression function (similar computation can be done for classification losses)

$$\min_{x=(W,u)} f(W, u) \stackrel{\text{def.}}{=} \frac{1}{2} \sum_{i=1}^n (h_{W,u}(a_i) - y_i)^2 = \frac{1}{2} \|\rho(AW^\top)u - y\|^2.$$

Note that here, the parameters being optimized are $(W, u) \in \mathbb{R}^{q \times p} \times \mathbb{R}^q$.

Optimizing with respect to u . This function f is convex with respect to u , since it is a quadratic function. Its gradient with respect to u can be computed as in (8) and thus

$$\nabla_u f(W, u) = \rho(AW^\top)^\top (\rho(AW^\top)u - y)$$

and one can compute in closed form the solution (assuming $\ker(\rho(AW^\top)) = \{0\}$) as

$$u^* = [\rho(AW^\top)^\top \rho(AW^\top)]^{-1} \rho(AW^\top)^\top y = [\rho(WA^\top) \rho(AW^\top)]^{-1} \rho(WA^\top) y$$

When $W = \operatorname{Id}_p$ and $\rho(s) = s$ one recovers the least square formula (9).

Optimizing with respect to W . The function f is non-convex with respect to W because the function ρ is itself non-linear. Training a MLP is thus a delicate process, and one can only hope to obtain a local minimum of f . It is also important to initialize correctly the neurons $(w_k)_k$ (for instance as unit norm random vector, but bias terms might need some adjustment), while u can be usually initialized at 0.

To compute its gradient with respect to W , we first note that for a perturbation $\varepsilon \in \mathbb{R}^{q \times p}$, one has

$$\rho(A(W + \varepsilon)^\top) = \rho(AW^\top + A\varepsilon^\top) = \rho(AW^\top) + \rho'(AW^\top) \odot (A\varepsilon^\top)$$

where we have denoted “ \odot ” the entry-wise multiplication of matrices, i.e. $U \odot V = (U_{i,j}V_{i,j})_{i,j}$. One thus has,

$$\begin{aligned} f(W + \varepsilon, u) &= \frac{1}{2}\|e + [\rho'(AW^\top) \odot (A\varepsilon^\top)]y\|^2 \quad \text{where } e \stackrel{\text{def.}}{=} \rho(AW^\top)u - y \in \mathbb{R}^n \\ &= f(W, u) + \langle e, [\rho'(AW^\top) \odot (A\varepsilon^\top)]y \rangle + o(\|\varepsilon\|) \\ &= f(W, u) + \langle A\varepsilon^\top, \rho'(AW^\top) \odot (eu^\top) \rangle \\ &= f(W, u) + \langle \varepsilon^\top, A^\top \times [\rho'(AW^\top) \odot (eu^\top)] \rangle. \end{aligned}$$

The gradient thus reads

$$\nabla_W f(W, u) = [\rho'(WA^\top) \odot (ue^\top)] \times A \in \mathbb{R}^{q \times p}.$$

8.3 Universality

In this section, to ease the exposition, we explicitly introduce the bias and use the variable “ $x \in \mathbb{R}^p$ ” in place of “ $a \in \mathbb{R}^p$ ”. We thus write the function computed by the MLP (including explicitly the bias z_k) as

$$h_{W,z,u}(x) \stackrel{\text{def.}}{=} \sum_{k=1}^q u_k \varphi_{w_k, z_k}(x) \quad \text{where } \varphi_{w,z}(x) \stackrel{\text{def.}}{=} \rho(\langle x, w \rangle + z).$$

The function $\varphi_{w,z}(x)$ is a ridge function in the direction orthogonal to $\bar{w} \stackrel{\text{def.}}{=} w/\|w\|$ and passing around the point $-\frac{z}{\|w\|}\bar{w}$.

In the following we assume that $\rho : \mathbb{R} \rightarrow \mathbb{R}$ is a bounded function such that

$$\rho(r) \xrightarrow{r \rightarrow -\infty} 0 \quad \text{and} \quad \rho(r) \xrightarrow{r \rightarrow +\infty} 1. \tag{44}$$

Note in particular that such a function cannot be a polynomial and that the ReLu function does not satisfy these hypothesis (universality for the ReLu is more involved to show). The goal is to show the following theorem.

Theorem 3 (Cybenko, 1989). *For any compact set $\Omega \subset \mathbb{R}^p$, the space spanned by the functions $\{\varphi_{w,z}\}_{w,z}$ is dense in $C(\Omega)$ for the uniform convergence. This means that for any continuous function f and any $\varepsilon > 0$, there exists $q \in \mathbb{N}$ and weights $(w_k, z_k, u_k)_{k=1}^q$ such that*

$$\forall x \in \Omega, \quad |f(x) - \sum_{k=1}^q u_k \varphi_{w_k, z_k}(x)| \leq \varepsilon.$$

In a typical ML scenario, this implies that one can “overfit” the data, since using a q large enough ensures that the training error can be made arbitrary small. Of course, there is a bias-variance tradeoff, and q needs to be cross-validated to account for the finite number n of data, and ensure a good generalization properties.

Proof in dimension $p = 1$. In 1D, the approximation $h_{W,z,u}$ can be thought as an approximation using smoothed step functions. Indeed, introducing a parameter $\varepsilon > 0$, one has (assuming the function is Lipschitz to ensure uniform convergence),

$$\varphi_{\frac{w}{\varepsilon}, \frac{z_k}{\varepsilon}} \xrightarrow{\varepsilon \rightarrow 0} 1_{[-z/w, +\infty[}$$

This means that

$$h_{\frac{w}{\varepsilon}, \frac{z}{\varepsilon}, u} \xrightarrow{\varepsilon \rightarrow 0} \sum_k u_k 1_{[-z_k/w_k, +\infty[},$$

which is a piecewise constant function. Inversely, any piecewise constant function can be written this way. Indeed, if h assumes the value d_k on each interval $[t_k, t_{k+1}[$, then it can be written as

$$h = \sum_k d_k (1_{[t_k, +\infty[} - 1_{[t_k, +\infty[}).$$

Since the space of piecewise constant functions is dense in continuous function over an interval, this proves the theorem.

Proof in arbitrary dimension p . We start by proving the following dual characterization of density, using bounded Borel measure $\mu \in \mathcal{M}(\Omega)$ i.e. such that $\mu(\Omega) < +\infty$.

Proposition 11. *If ρ is such that for any Borel measure $\mu \in \mathcal{M}(\Omega)$*

$$\left(\forall (w, z), \int \rho(\langle x, w \rangle + z) d\mu(x) = 0 \right) \implies \mu = 0, \quad (45)$$

then Theorem 3 holds.

Proof. We consider the linear space

$$\mathcal{S} \stackrel{\text{def.}}{=} \left\{ \sum_{k=1}^q u_k \varphi_{w_k, z_k} ; q \in \mathbb{N}, w_k \in \mathbb{R}^p, u_k \in \mathbb{R}, z_k \in \mathbb{R} \right\} \subset \mathcal{C}(\Omega).$$

Let $\bar{\mathcal{S}}$ be its closure in $\mathcal{C}(\Omega)$ for $\|\cdot\|_\infty$, which is a Banach space. If $\bar{\mathcal{S}} \neq \mathcal{C}(\Omega)$, let us pick $g \neq 0$, $g \in \mathcal{C}(\Omega) \setminus \bar{\mathcal{S}}$. We define the linear form L on $\bar{\mathcal{S}} \oplus \text{span}(g)$ as

$$\forall s \in \bar{\mathcal{S}}, \forall \lambda \in \mathbb{R}, \quad L(s + \lambda g) = \lambda$$

so that $L = 0$ on $\bar{\mathcal{S}}$. L is a bounded linear form, so that by Hahn-Banach theorem, it can be extended in a bounded linear form $\bar{L} : \mathcal{C}(\Omega) \rightarrow \mathbb{R}$. Since $L \in \mathcal{C}(\Omega)^*$ (the dual space of continuous linear form), and that this dual space is identified with Borel measures, there exists $\mu \in \mathcal{M}(\Omega)$, with $\mu \neq 0$, such that for any continuous function h , $\bar{L}(h) = \int_{\Omega} h(x) d\mu(x)$. But since $\bar{L} = 0$ on $\bar{\mathcal{S}}$, $\int \rho(\langle \cdot, w \rangle + z) d\mu = 0$ for all (w, z) and thus by hypothesis, $\mu = 0$, which is a contradiction. \square

The theorem now follows from the following proposition.

Proposition 12. *If ρ is continuous and satisfies (44), then it satisfies (45).*

Proof. One has

$$\varphi_{\frac{w}{\varepsilon}, \frac{u}{\varepsilon} + t}(x) = \rho \left(\frac{\langle x, w \rangle + u}{\varepsilon} + t \right) \xrightarrow{\varepsilon \rightarrow 0} \gamma(x) \stackrel{\text{def.}}{=} \begin{cases} 1 & \text{if } H_{w,u}, \\ \rho(t) & \text{if } x \in P_{w,u}, \\ 0 & \text{if } \langle w, x \rangle + u < 0, \end{cases}$$

where we defined $H_{w,u} \stackrel{\text{def.}}{=} \{x ; \langle w, x \rangle + u > 0\}$ and $P_{w,u} \stackrel{\text{def.}}{=} \{x ; \langle w, x \rangle + u = 0\}$. By Lebesgue dominated convergence (since the involved quantities are bounded uniformly on a compact set)

$$\int \varphi_{\frac{w}{\varepsilon}, \frac{u}{\varepsilon} + t} d\mu \xrightarrow{\varepsilon \rightarrow 0} \int \gamma d\mu = \varphi(t) \mu(P_{w,u}) + \mu(H_{w,u}).$$

Thus if μ is such that all these integrals vanish, then

$$\forall (w, u, t), \quad \varphi(t)\mu(P_{w,u}) + \mu(H_{w,u}) = 0.$$

By selecting (t, t') such that $\varphi(t) \neq \varphi(t')$, one has that

$$\forall (w, u), \quad \mu(P_{w,u}) = \mu(H_{w,u}) = 0.$$

We now need to show that $\mu = 0$. For a fixed $w \in \mathbb{R}^p$, we consider the function

$$h \in L^\infty(\mathbb{R}), \quad F(h) \stackrel{\text{def}}{=} \int_{\Omega} h(\langle w, x \rangle) d\mu(x).$$

$F : L^\infty(\mathbb{R}) \rightarrow \mathbb{R}$ is a bounded linear form since $|F(\mu)| \leq \|h\|_\infty \mu(\Omega)$ and $\mu(\Omega) < +\infty$. One has

$$F(1_{[-u, +\infty[} = \int_{\Omega} 1_{[-u, +\infty[}(\langle w, x \rangle) d\mu(x) = \mu(P_{w,u}) + \mu(H_{w,u}) = 0.$$

By linearity, $F(h) = 0$ for all piecewise constant functions, and F is a continuous linear form, so that by density $F(h) = 0$ for all functions $h \in L^\infty(\mathbb{R})$. Applying this for $h(r) = e^{ir}$ one obtains

$$\hat{\mu}(w) \stackrel{\text{def}}{=} \int_{\Omega} e^{i\langle x, w \rangle} d\mu(x) = 0.$$

This means that the Fourier transform of μ is zero, so that $\mu = 0$. \square

Quantitative rates. Note that Theorem 3 is not constructive in the sense that it does not explain how to compute the weights $(w_k, u_k, z_k)_k$ to reach a desired accuracy. Since for a fixed q the function is non-convex, this is not surprising. Some recent studies show that if q is large enough, a simple gradient descent is able to reach an arbitrary good accuracy, but it might require a very large q .

Theorem 3 is also not quantitative since it does not tell how much neurons q is needed to reach a desired accuracy. To obtain quantitative bounds, continuity is not enough, it requires to add smoothness constraints. For instance, Barron proved that if

$$\int \|\omega\| |\hat{f}(\omega)| d\omega \leq C_f$$

where $\hat{f}(\omega) = \int f(x) e^{-i\langle x, \omega \rangle} dx$ is the Fourier transform of f , then for $q \in \mathbb{N}$ there exists $(w_k, u_k, z_k)_k$

$$\frac{1}{\text{Vol}(B(0, r))} \int_{\|x\| \leq r} |f(x) - \sum_{k=1}^q u_k \varphi_{w_k, z_k}(x)|^2 dx \leq \frac{(2rC_f)^2}{q}.$$

The surprising part of this Theorem is that the $1/q$ decay is independent of the dimension p . Note however that the constant involved C_f might depend on p .

9 Automatic Differentiation

The main computational bottleneck of gradient descent methods (batch or stochastic) is the computation of gradients $\nabla f(x)$. For simple functionals, such as those encountered in ERM for linear models, and also for MLP with a single hidden layer, 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. We present methods to compute recursively in an optimal manner these gradients. The purpose of this approach is to automatize this computational step.

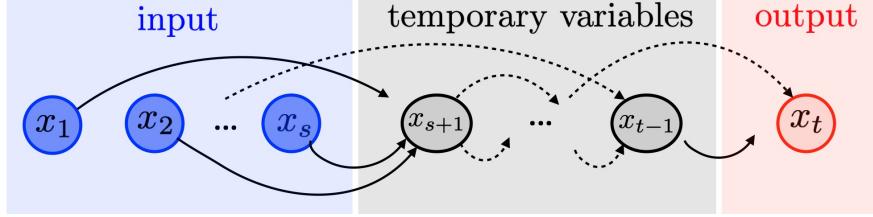


Figure 19: A computational graph.

9.1 Finite Differences and Symbolic Calculus

We consider $f : \mathbb{R}^p \rightarrow \mathbb{R}$ 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

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

requires $p + 1$ evaluations of f , where we denoted $\delta_k = (0, \dots, 0, 1, 0, \dots, 0)$ where the 1 is at index k . 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 . This type of method is similar to symbolic calculus in the sense that it provides (up to machine precision) exact gradient computation. But symbolic calculus does not take into account the underlying algorithm which compute the function, while automatic differentiation factorizes the computation of the derivative according to an efficient algorithm.

9.2 Computational Graphs

We consider a generic function $f(x)$ where $x = (x_1, \dots, x_s)$ are the input variables. We assume that f is implemented in an algorithm, with intermediate variable (x_{s+1}, \dots, 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 $k = 1, \dots, s$. 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

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

where f_k is a function which only depends on the previous variables, see Fig. 19. One can represent this algorithm using a directed acyclic graph (DAG), linking the variables involved in f_k to x_k . The nodes 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. Note that the goal of automatic differentiation is not to define an efficient computational graph, it is up to the user to provide this graph. Computing an efficient graph associated to a mathematical formula is a complicated combinatorial problem, which still has to be solved by the user. Automatic differentiation thus leverage the availability of an efficient graph to provide an efficient algorithm to evaluate derivatives.

9.3 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, \dots, x_p (we only write thing for the first variable, the method being of course the same with respect to the other ones).

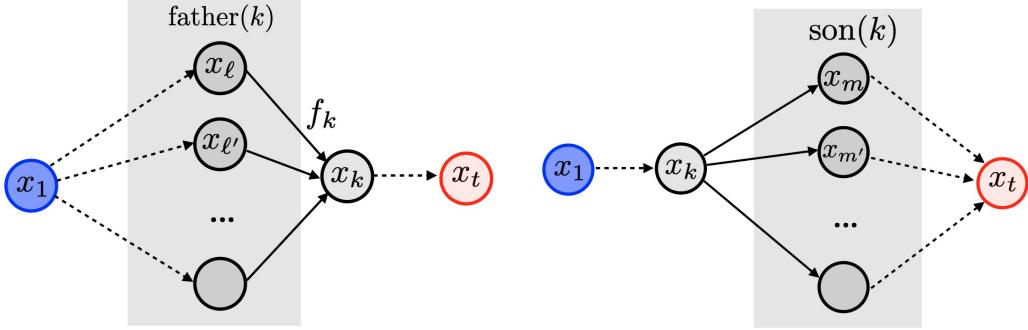


Figure 20: Relation between the variable for the forward (left) and backward (right) modes.

The method initialize the derivative of the input nodes

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

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

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

The notation “parent(k)” denotes the nodes $\ell < k$ of the graph that are connected to k , see Figure 20, left. 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 [...] 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 parent 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 $\frac{\partial}{\partial x_1}, \dots, \frac{\partial}{\partial x_p}$). For a large p , this is prohibitive.

Simple example. We consider the fonction

$$f(x, y) = y \log(x) + \sqrt{y \log(x)} \tag{46}$$

whose computational graph is displayed on Figure 21. The iterations of the forward mode to compute the derivative with respect to x read

$$\begin{aligned} \frac{\partial x}{\partial x} &= 1, \quad \frac{\partial y}{\partial x} = 0 \\ \frac{\partial a}{\partial x} &= \left[\frac{\partial a}{\partial x} \right] \frac{\partial x}{\partial x} = \frac{1}{x} \frac{\partial x}{\partial x} && \{x \mapsto a = \log(x)\} \\ \frac{\partial b}{\partial x} &= \left[\frac{\partial b}{\partial a} \right] \frac{\partial a}{\partial x} + \left[\frac{\partial b}{\partial y} \right] \frac{\partial y}{\partial x} = y \frac{\partial a}{\partial x} + 0 && \{(y, a) \mapsto b = ya\} \\ \frac{\partial c}{\partial x} &= \left[\frac{\partial c}{\partial b} \right] \frac{\partial b}{\partial x} = \frac{1}{2\sqrt{b}} \frac{\partial b}{\partial x} && \{b \mapsto c = \sqrt{b}\} \\ \frac{\partial f}{\partial x} &= \left[\frac{\partial f}{\partial b} \right] \frac{\partial b}{\partial x} + \left[\frac{\partial f}{\partial c} \right] \frac{\partial c}{\partial x} = 1 \frac{\partial b}{\partial x} + 1 \frac{\partial c}{\partial x} && \{(b, c) \mapsto f = b + c\} \end{aligned}$$

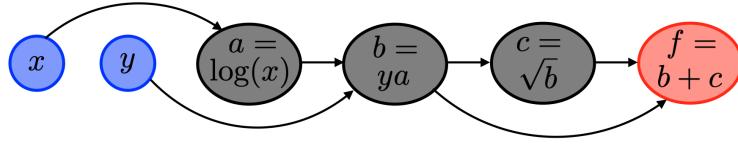


Figure 21: Example of a simple computational graph.

One needs to run another forward pass to compute the derivative with respect to y

$$\begin{aligned}
 \frac{\partial x}{\partial y} &= 0, \quad \frac{\partial y}{\partial y} = 1 \\
 \frac{\partial a}{\partial y} &= \left[\frac{\partial a}{\partial x} \right] \frac{\partial x}{\partial y} = 0 & \{x \mapsto a = \log(x)\} \\
 \frac{\partial b}{\partial y} &= \left[\frac{\partial b}{\partial a} \right] \frac{\partial a}{\partial y} + \left[\frac{\partial b}{\partial y} \right] \frac{\partial y}{\partial y} = 0 + a \frac{\partial y}{\partial y} & \{(y, a) \mapsto b = ya\} \\
 \frac{\partial c}{\partial y} &= \left[\frac{\partial c}{\partial b} \right] \frac{\partial b}{\partial y} = \frac{1}{2\sqrt{b}} \frac{\partial b}{\partial y} & \{b \mapsto c = \sqrt{b}\} \\
 \frac{\partial f}{\partial y} &= \left[\frac{\partial f}{\partial b} \right] \frac{\partial b}{\partial y} + \left[\frac{\partial f}{\partial c} \right] \frac{\partial c}{\partial y} = 1 \frac{\partial b}{\partial y} + 1 \frac{\partial c}{\partial y} & \{(b, c) \mapsto f = b + c\}
 \end{aligned}$$

Dual numbers. A convenient way to implement this forward pass is to make use of so called “dual number”, which is an algebra over the real where the number have the form $x + \varepsilon x'$ where ε is a symbol obeying the rule that $\varepsilon^2 = 0$. Here $(x, x') \in \mathbb{R}^2$ and x' is intended to store a derivative with respect to some input variable. These number thus obeys the following arithmetic operations

$$(x + \varepsilon x')(y + \varepsilon y') = xy + \varepsilon(xy' + yx') \quad \text{and} \quad \frac{1}{x + \varepsilon x'} = \frac{1}{x} - \varepsilon \frac{x'}{x^2}.$$

If f is a polynomial or a rational function, from these rules one has that

$$f(x + \varepsilon) = f(x) + \varepsilon f'(x).$$

For a more general basic function f , one needs to overload it so that

$$f(x + \varepsilon x') \stackrel{\text{def.}}{=} f(x) + \varepsilon f'(x)x'.$$

Using this definition, one has that

$$(f \circ g)(x + \varepsilon) = f(g(x)) + \varepsilon f'(g(x))g'(x)$$

which corresponds to the usual chain rule. More generally, if $f(x_1, \dots, x_s)$ is a function implemented using these overloaded basic functions, one has

$$f(x_1 + \varepsilon, x_2, \dots, x_s) = f(x_1, \dots, x_s) + \varepsilon \frac{\partial f}{\partial x_1}(x_1, \dots, x_s)$$

and this evaluation is equivalent to applying the forward mode of automatic differentiation to compute $\frac{\partial f}{\partial x_1}(x_1, \dots, x_s)$ (and similarly for the other variables).

9.4 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 all the inner nodes.

The method initializes the derivative of the final node

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

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

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

The notation “parent(k)” denotes the nodes $\ell < k$ of the graph that are connected to k , see Figure 20, right.

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 follows

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

where $\left(\frac{\partial f_m(x_1, \dots, x_m)}{\partial x_k} \right)^\top \in \mathbb{R}^{n_k \times n_m}$ 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 necessitates the pre-computation of all the intermediate variables $(x_k)_{k=p}^t$, which can be prohibitive in terms of memory usage when t is large. There exists check-pointing methods to alleviate this issue, but it is out of the scope of this course.

Simple example. We consider once again the function $f(x)$ of (46), the iterations of the reverse mode read

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

The advantage of the reverse mode is that a single traversal of the computational graph allows to compute both derivatives with respect to x, y , while the forward mode necessitates two passes.

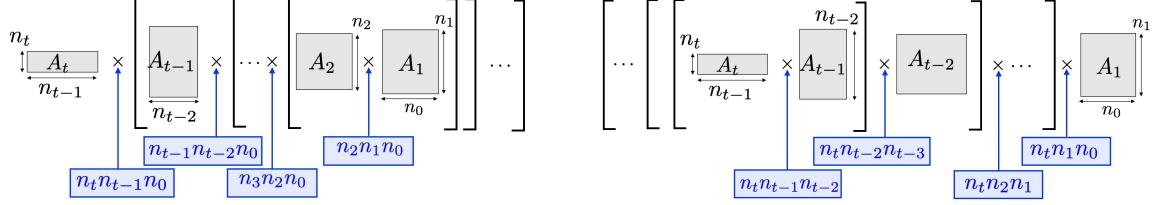


Figure 22: Complexity of forward (left) and backward (right) modes for composition of functions.

9.5 Feed-forward Compositions

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

$$f = f_t \circ f_{t-1} \circ \dots \circ f_2 \circ f_1 \quad (47)$$

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

The forward function evaluation algorithm initializes $x_0 = x \in \mathbb{R}^{n_0}$ and then computes

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

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

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

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

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

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

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 $n p q$ operations. As shown on Figure 22, the complexity of the forward and backward modes are

$$n_0 \sum_{k=1}^{t-1} n_k n_{k+1} \quad \text{and} \quad n_t \sum_{k=0}^{t-2} n_k n_{k+1}$$

So if $n_t \ll n_0$ (which is the typical case in ML scenario where $n_t = 1$) then the backward mode is cheaper.

9.6 Feed-forward Architecture

We can generalize the previous example to account for feed-forward architectures, such as neural networks, which are of the form

$$\forall k = 1, \dots, t, \quad x_k = f_k(x_{k-1}, \theta_{k-1}) \quad (48)$$

where θ_{k-1} is a vector of parameters and $x_0 \in \mathbb{R}^{n_0}$ is given. The function to minimize has the form

$$f(\theta) \stackrel{\text{def.}}{=} L(x_t) \quad (49)$$

where $L : \mathbb{R}^{n_t} \rightarrow \mathbb{R}$ is some loss function (for instance a least square or logistic prediction risk) and $\theta = (\theta_k)_{k=0}^{t-1}$. Figure 23, top, displays the associated computational graph.

One can use the reverse mode automatic differentiation to compute the gradient of f by computing successively the gradient with respect to all (x_k, θ_k) . One initializes

$$\nabla_{x_t} f = \nabla L(x_t)$$

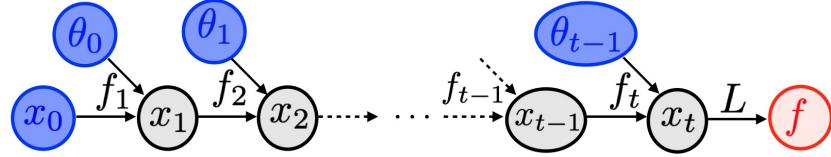


Figure 23: Computational graph for a feedforward architecture.

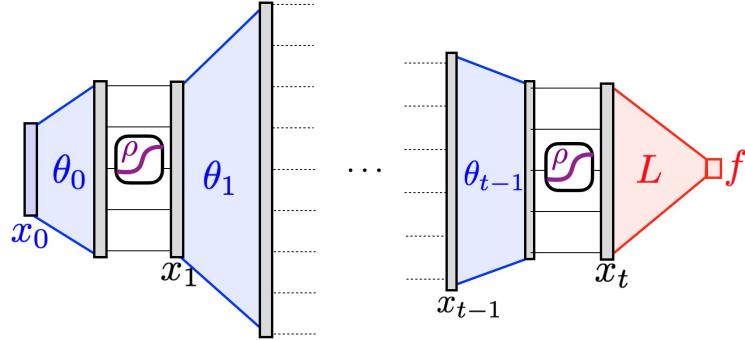


Figure 24: Multi-layer perceptron parameterization.

and then recurse from $k = t - 1$ to 0

$$z_{k-1} = [\partial_x f_k(x_{k-1}, \theta_{k-1})]^\top z_k \quad \text{and} \quad \nabla_{\theta_{k-1}} f = [\partial_\theta f_k(x_{k-1}, \theta_{k-1})]^\top (\nabla_{x_k} f) \quad (50)$$

where we denoted $z_k \stackrel{\text{def.}}{=} \nabla_{x_k} f(\theta)$ the gradient with respect to x_k .

Multilayers perceptron. For instance, feedforward deep network (fully connected for simplicity) corresponds to using

$$\forall x_{k-1} \in \mathbb{R}^{n_{k-1}}, \quad f_k(x_{k-1}, \theta_{k-1}) = \rho(\theta_{k-1} x_{k-1}) \quad (51)$$

where $\theta_{k-1} \in \mathbb{R}^{n_k \times n_{k-1}}$ are the neuron's weights and ρ a fixed pointwise linearity, see Figure 24. One has, for a vector $z_k \in \mathbb{R}^{n_k}$ (typically equal to $\nabla_{x_k} f$)

$$\begin{cases} [\partial_x f_k(x_{k-1}, \theta_{k-1})]^\top (z_k) = \theta_{k-1}^\top w_k z_k, \\ [\partial_\theta f_k(x_{k-1}, \theta_{k-1})]^\top (z_k) = w_k x_{k-1}^\top. \end{cases} \quad \text{where} \quad w_k \stackrel{\text{def.}}{=} \text{diag}(\rho'(\theta_{k-1} x_{k-1})).$$

Link with adjoint state method. One can interpret (48) as a time discretization of a continuous ODE. One imposes that the dimension $n_k = n$ is fixed, and denotes $x(t) \in \mathbb{R}^n$ a continuous time evolution, so that $x_k \rightarrow x(k\tau)$ when $k \rightarrow +\infty$ and $k\tau \rightarrow t$. Imposing then the structure

$$f_k(x_{k-1}, \theta_{k-1}) = x_{k-1} + \tau u(x_{k-1}, \theta_{k-1}, k\tau) \quad (52)$$

where $u(x, \theta, t) \in \mathbb{R}^n$ is a parameterized vector field, as $\tau \rightarrow 0$, one obtains the non-linear ODE

$$\dot{x}(t) = u(x(t), \theta(t), t) \quad (53)$$

with $x(t=0) = x_0$.

Denoting $z(t) = \nabla_{x(t)} f(\theta)$ the “adjoint” vector field, the discrete equations (55) becomes the so-called adjoint equations, which is a linear ODE

$$\dot{z}(t) = -[\partial_x u(x(t), \theta(t), t)]^\top z(t) \quad \text{and} \quad \nabla_{\theta(t)} f(\theta) = [\partial_\theta u(x(t), \theta(t), t)]^\top z(t).$$

Note that the correct normalization is $\frac{1}{\tau} \nabla_{\theta_{k-1}} f \rightarrow \nabla_{\theta(t)} f(\theta)$

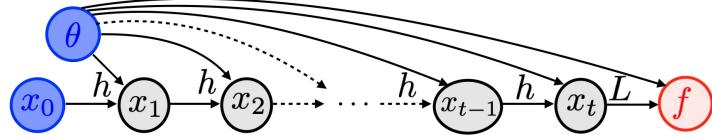


Figure 25: Computational graph for a recurrent architecture.

9.7 Recurrent Architectures

Parametric recurrent functions are obtained by using the same parameter $\theta = \theta_k$ and $f_k = h$ recursively in (51), so that

$$\forall k = 1, \dots, t, \quad x_k = h(x_{k-1}, \theta). \quad (54)$$

We consider a real valued function of the form

$$f(\theta) = L(x_t, \theta)$$

so that here the final loss depends on θ (which is thus more general than (49)). Figure 25, bottom, displays the associated computational graph.

The back-propagation then operates as

$$\nabla_{x_{k-1}} f = [\partial_x h(x_{k-1}, \theta)]^\top \nabla_{x_k} f \quad \text{and} \quad \nabla_\theta f = \nabla_\theta L(x_t, \theta) + \sum_k [\partial_\theta h(x_{k-1}, \theta)]^\top \nabla_{x_k} f. \quad (55)$$

Similarly, writing $h(x, \theta) = x + \tau u(x, \theta)$, letting $(k, k\tau) \rightarrow (+\infty, t)$, one obtains the forward non-linear ODE with a time-stationary vector field

$$\dot{x}(t) = u(x(t), \theta)$$

and the following linear backward adjoint equation, for $f(\theta) = L(x(T), \theta)$

$$\dot{z}(t) = -[\partial_x u(x(t), \theta)]^\top z(t) \quad \text{and} \quad \nabla_\theta f(\theta) = \nabla_\theta L(x(T), \theta) + \int_0^T [\partial_\theta f(x(t), \theta)]^\top z(t) dt. \quad (56)$$

with $z(0) = \nabla_x L(x_t, \theta)$.

Residual recurrent networks. A recurrent network is defined using

$$h(x, \theta) = x + W_2^\top \rho(W_1 x)$$

as displayed on Figure 26, where $\theta = (W_1, W_2) \in (\mathbb{R}^{n \times q})^2$ are the weights and ρ is a pointwise non-linearity. The number q of hidden neurons can be increased to approximate more complex functions. In the special case where $W_2 = -\tau W_1$, and $\rho = \psi'$, then this is a special case of an argmin layer (58) to minimize the function $\mathcal{E}(x, \theta) = \psi(W_1 x)$ using gradient descent, where $\psi(u) = \sum_i \psi(u_i)$ is a separable function. The Jacobians $\partial_\theta h$ and $\partial_x h$ are computed similarly to (56).

Mitigating memory requirement. The main issue of applying this backpropagation method to compute $\nabla f(\theta)$ is that it requires a large memory to store all the iterates $(x_k)_{k=0}^t$. A workaround is to use checkpointing, which stores some of these intermediate results and re-run partially the forward algorithm to reconstruct missing values during the backward pass. Clever hierarchical methods perform this recursively in order to only require $\log(t)$ stored values and a $\log(t)$ increase on the numerical complexity.

In some situations, it is possible to avoid the storage of the forward result, if one assumes that the algorithm can be run backward. This means that there exists some functions g_k so that

$$x_k = g_k(x_{k+1}, \dots, x_t).$$

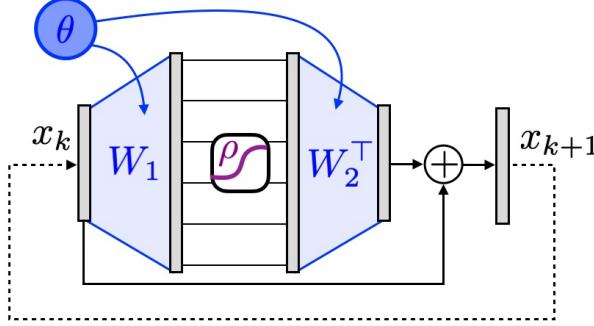


Figure 26: Recurrent residual perceptron parameterization.

In practice, this function typically also depends on a few extra variables, in particular on the input values (x_0, \dots, x_s) .

An example of this situation is when one can split the (continuous time) variable as $x(t) = (r(t), s(t))$ and the vector field u in the continuous ODE (53) has a symplectic structure of the form $u((r, s), \theta, t) = (F(s, \theta, t), G(r, \theta, t))$. One can then use a leapfrog integration scheme, which defines

$$r_{k+1} = r_k + \tau F(s_k, \theta_k, \tau k) \quad \text{and} \quad s_{k+1} = s_k + \tau G(r_{k+1}, \theta_{k+1/2}, \tau(k + 1/2)).$$

One can reverse these equations exactly as

$$s_k = s_{k+1} - \tau G(r_{k+1}, \theta_{k+1/2}, \tau(k + 1/2)). \quad \text{and} \quad r_k = r_{k+1} - \tau F(s_k, \theta_k, \tau k).$$

Fixed point maps In some applications (some of which are detailed below), the iterations x_k converges to some $x^*(\theta)$ which is thus a fixed point

$$x^*(\theta) = h(x^*(\theta), \theta).$$

Instead of applying the back-propagation to compute the gradient of $f(\theta) = L(x_t, \theta)$, one can thus apply the implicit function theorem to compute the gradient of $f^*(\theta) = L(x^*(\theta), \theta)$. Indeed, one has

$$\nabla f^*(\theta) = [\partial x^*(\theta)]^\top (\nabla_x L(x^*(\theta), \theta)) + \nabla_\theta L(x^*(\theta), \theta). \quad (57)$$

Using the implicit function theorem one can compute the Jacobian as

$$\partial x^*(\theta) = - \left(\frac{\partial h}{\partial x}(x^*(\theta), \theta) \right)^{-1} \frac{\partial h}{\partial \theta}(x^*(\theta), \theta).$$

In practice, one replaces in these formulas $x^*(\theta)$ by x_t , which produces an approximation of $\nabla f(\theta)$. The disadvantage of this method is that it requires the resolution of a linear system, but its advantage is that it bypasses the memory storage issue of the backpropagation algorithm.

Argmin layers One can define a mapping from some parameter θ to a point $x(\theta)$ by solving a parametric optimization problem

$$x(\theta) = \underset{x}{\operatorname{argmin}} \mathcal{E}(x, \theta).$$

The simplest approach to solve this problem is to use a gradient descent scheme, $x_0 = 0$ and

$$x_{k+1} = x_k - \tau \nabla \mathcal{E}(x_k, \theta). \quad (58)$$

This has the form (52) when using the vector field $u(x, \theta) = \nabla \mathcal{E}(x_k, \theta)$.

Using formula (57) in this case where $h = \nabla \mathcal{E}$, one obtains

$$\nabla f^*(\theta) = - \left(\frac{\partial^2 \mathcal{E}}{\partial x \partial \theta}(x^*(\theta), \theta) \right)^\top \left(\frac{\partial^2 \mathcal{E}}{\partial x^2}(x^*(\theta), \theta) \right)^{-1} (\nabla_x L(x^*(\theta), \theta)) + \nabla_\theta L(x^*(\theta), \theta)$$

In the special case where the function $f(\theta)$ is the minimized function itself, i.e. $f(\theta) = \mathcal{E}(x^*(\theta), \theta)$, i.e. $L = \mathcal{E}$, then one can apply the implicit function theorem formula (57), which is much simpler since in this case $\nabla_x L(x^*(\theta), \theta) = 0$ so that

$$\nabla f^*(\theta) = \nabla_\theta L(x^*(\theta), \theta). \quad (59)$$

This result is often called Danskin theorem or the envelope theorem.

Sinkhorn's algorithm Sinkhorn algorithm approximates the optimal distance between two histograms $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$ using the following recursion on multipliers, initialized as $x_0 \stackrel{\text{def.}}{=} (u_0, v_0) = (1_n, 1_m)$

$$u_{k+1} = \frac{a}{Kv_k}, \quad \text{and} \quad v_{k+1} = \frac{b}{K^\top u_k}.$$

where \div is the pointwise division and $K \in \mathbb{R}_+^{n \times m}$ is a kernel. Denoting $\theta = (a, b) \in \mathbb{R}^{n+m}$ and $x_k = (u_k, v_k) \in \mathbb{R}^{n+m}$, the OT distance is then approximately equal to

$$f(\theta) = \mathcal{E}(x_t, \theta) \stackrel{\text{def.}}{=} \langle a, \log(u_t) \rangle + \langle b, \log(v_t) \rangle - \varepsilon \langle Ku_t, v_t \rangle.$$

Sinkhorn iteration are alternate minimization to find a minimizer of \mathcal{E} .

Denoting $\mathcal{K} \stackrel{\text{def.}}{=} \begin{pmatrix} 0 & K \\ K^\top & 0 \end{pmatrix} \in \mathbb{R}^{(n+m) \times (n+m)}$, one can re-write these iterations in the form (54) using

$$h(x, \theta) = \frac{\theta}{\mathcal{K}x} \quad \text{and} \quad L(x_t, \theta) = \mathcal{E}(x_t, \theta) = \langle \theta, \log(x_t) \rangle - \varepsilon \langle Ku_t, v_t \rangle.$$

One has the following differential operator

$$[\partial_x h(x, \theta)]^\top = -\mathcal{K}^\top \operatorname{diag}\left(\frac{\theta}{(\mathcal{K}x)^2}\right), \quad [\partial_\theta h(x, \theta)]^\top = \operatorname{diag}\left(\frac{1}{\mathcal{K}x}\right).$$

Similarly as for the argmin layer, at convergence $x_k \rightarrow x^*(\theta)$, one finds a minimizer of \mathcal{E} , so that $\nabla_x L(x^*(\theta), \theta) = 0$ and thus the gradient of $f^*(\theta) = \mathcal{E}(x^*(\theta), \theta)$ can be computed using (59) i.e.

$$\nabla f^*(\theta) = \log(x^*(\theta)).$$