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

December 9, 2024

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 **1**
1.1 Unconstraint optimization . . . 1
1.2 Regression . . . 2
1.3 Classification . . . 2

2 Basics of Convex Analysis **2**
2.1 Existence of Solutions . . . 2
2.2 Convexity . . . 3
2.3 Convex Sets . . . 4

3 Derivative and gradient **4**
3.1 Gradient . . . 4
3.2 First Order Conditions . . . 5
3.3 Least Squares . . . 6
3.4 Link with PCA . . . 7
3.5 Classification . . . 8
3.6 Chain Rule . . . 8

4 Gradient Descent Algorithm **9**
4.1 Steepest Descent Direction . . . 9
4.2 Gradient Descent . . . 10

5 Convergence Analysis **11**
5.1 Quadratic Case . . . 11
5.2 General Case . . . 14
5.3 Acceleration . . . 17

6 Mirror Descent and Implicit Bias **18**
6.1 Bregman Divergences . . . 18
6.2 Mirror descent . . . 19
6.3 Re-parameterized flows . . . 20
6.4 Implicit Bias . . . 21

7 Regularization **22**
7.1 Penalized Least Squares . . . 22
7.2 Ridge Regression . . . 23
7.3 Lasso . . . 24
7.4 Iterative Soft Thresholding . . . 25

8 Stochastic Optimization **26**
8.1 Minimizing Sums and Expectation . . . 26
8.2 Batch Gradient Descent (BGD) . . . 26
8.3 Stochastic Gradient Descent (SGD) . . . 27
8.4 Stochastic Gradient Descent with Averaging (SGA) . . . 29
8.5 Stochastic Averaged Gradient Descent (SAG) . . . 30

9 Automatic Differentiation **30**
9.1 Finite Differences and Symbolic Calculus . . . 31
9.2 Computational Graphs . . . 31
9.3 Forward Mode of Automatic Differentiation . . . 32
9.4 Reverse Mode of Automatic Differentiation . . . 34
9.5 Feed-forward Compositions . . . 35
9.6 Feed-forward Architecture . . . 36
9.7 Recurrent Architectures . . . 37

1 Motivation in Machine Learning

1.1 Unconstraint optimization

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

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

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

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

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

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

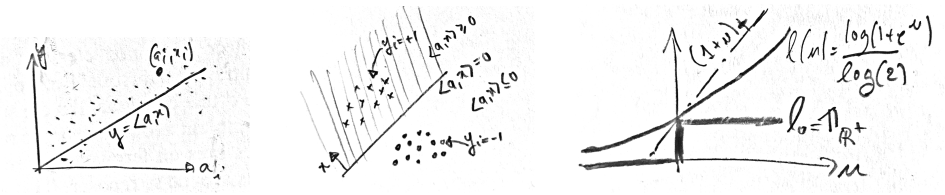


Figure 1: Left: linear regression, middle: linear classifier, right: loss function for classification.

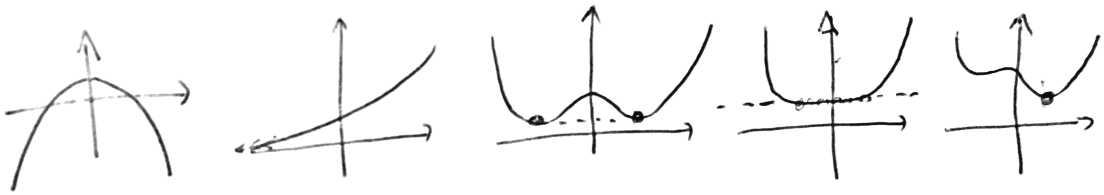


Figure 2: Left: non-existence of minimizer, middle: multiple minimizers, right: uniqueness.

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.

1.2 Regression

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

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

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

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

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

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 below, 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.

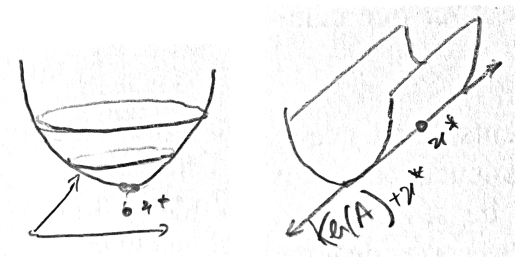


Figure 3: Coercivity condition for least squares.

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

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

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

Example 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 $x^\star$ is a solution, then $x^\star + u$ is also solution for any $u \in \ker(A)$, so that the set of minimizer is unbounded. On contrary, if $\ker(A) = \{0\}$, we will show later that the set of minimizer is unique, see Fig. 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 $(x, y) \in (\mathbb{R}^p)^2$,

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

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

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

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.

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

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

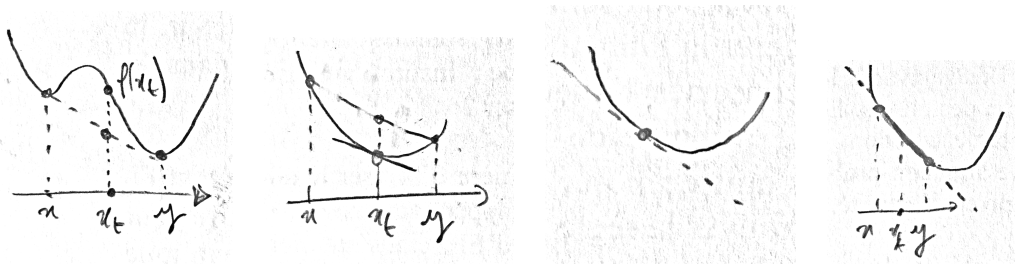


Figure 4: Convex vs. non-convex functions ; Strictly convex vs. non strictly convex functions.

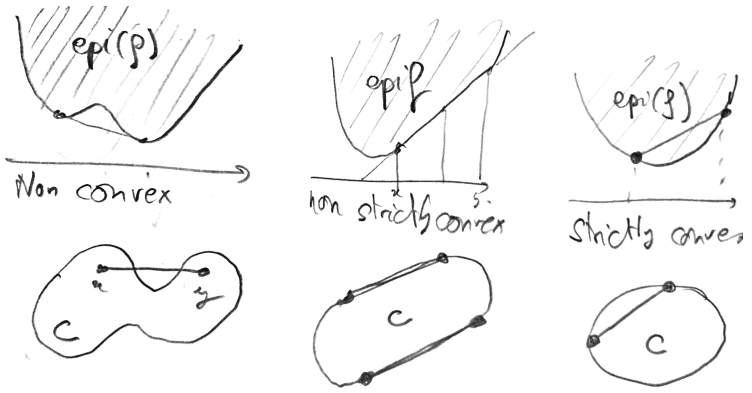


Figure 5: Comparison of convex functions $f : \mathbb{R}^p \to \mathbb{R}$ (for $p = 1$) and convex sets $C \subset \mathbb{R}^p$ (for $p = 2$).

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 $\mathrm{epi}(f) \overset{\text{def.}}{=} \big\{(x, t) \in \mathbb{R}^{p+1} \;;\; t \geqslant f(x)\big\}$ is a convex set.

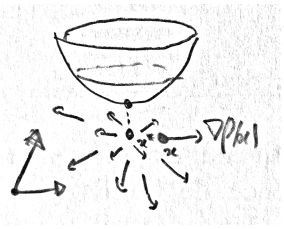
Remark 1 (Convexity of the set of minimizers). In general, minimizers $x^\star$ might be non-unique, as shown on Figure 3. When f is convex, the set $\mathrm{argmin}(f)$ of minimizers is itself a convex set. Indeed, if $x_1^\star$ and $x_2^\star$ are minimizers, so that in particular $f(x_1^\star) =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

3 Derivative and gradient

3.1 Gradient

If f is differentiable along each axis, we denote

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



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

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

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

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

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

Here $R(\varepsilon) = o(\|\varepsilon\|)$ denotes a quantity which decays faster than ε toward 0, i.e. $\frac{R(\varepsilon)}{\|\varepsilon\|} \to 0$ as $\varepsilon \to 0$. Existence of partial derivative corresponds to f being differentiable along the axes, while differentiability should hold for any converging sequence of $\varepsilon \to 0$ (i.e. not along along a fixed direction). A counter example in 2-D is $f(x) = \frac{2x_1 x_2 (x_1 + x_2)}{x_1^2 + x_2^2}$ with $f(0) = 0$, which is affine with different slope along each radial lines.

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*

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

Proof. One can write the convexity condition as

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

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

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

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

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

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

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

□

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 below.

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

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

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

Proof. One has for ε small enough and u fixed

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

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

Note that the converse is not true in general, since one might have $\nabla f(x) = 0$ but x is not a local mininimum. For instance $x = 0$ for $f(x) = -x^2$ (here x is a maximizer) or $f(x) = x^3$ (here x is neither a maximizer or a minimizer, it is a saddle point), see Fig. 6. Note however that in practice, if $\nabla f(x^\star) = 0$ but x is not a local minimum, then $x^\star$ tends to be an unstable equilibrium. Thus most often a gradient-based algorithm will converge to points with $\nabla f(x^\star) = 0$ that are local minimizers. The following proposition shows that a much strong result holds if f is convex.

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

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

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

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

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

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

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

□

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

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

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

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

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

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

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

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

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

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

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

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

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 $m \overset{\text{def.}}{=} \frac{1}{n}\sum_{i=1}^n a_i \in \mathbb{R}^p$ is the empirical mean. In this case, $\frac{C}{n} = A^\top A/n \in \mathbb{R}^{p\times p}$ is the empirical covariance of the point cloud $(a_i)_i$, it encodes the covariances between the coordinates of the points. Denoting $a_i = (a_{i,1},\dots,a_{i,p})^\top \in \mathbb{R}^p$ (so that $A = (a_{i,j})_{i,j}$) the coordinates, one has

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

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

For instance, in dimension $p = 2$,

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

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

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

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 $\lambda_k \geqslant 0$ are positive.

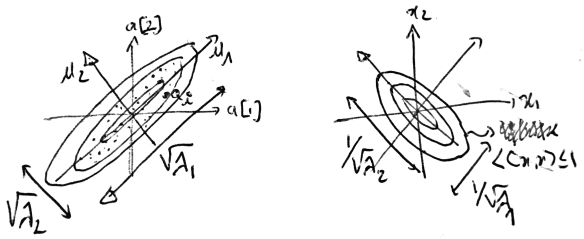


Figure 7: Left: point clouds $(a_i)_i$ with associated PCA directions, right: quadratic part of $f(x)$.

If one assumes that the eigenvalues are ordered $\lambda_1 \geqslant \lambda_2 \geqslant \ldots \geqslant \lambda_p$, then projecting the points a_i on the first m eigenvectors can be shown to be in some sense the best linear dimensionality reduction possible (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 ($m = 3$).

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

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

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

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

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

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

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

3.6 Chain Rule

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

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

which shows that

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

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 $F : \mathbb{R}^p \to \mathbb{R}^q$, its differentiable at x is a linear operator $\partial F(x) : \mathbb{R}^p \to \mathbb{R}^q$, i.e. it can be represented as a matrix (still denoted $\partial F(x)$) $\partial F(x) \in \mathbb{R}^{q \times p}$. The entries of this matrix are the partial differential, denoting $F(x) = (F_1(x), \ldots, F_q(x))$,

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

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

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

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 $f : \mathbb{R}^p \to \mathbb{R}$, then the differential $\partial f(x) \in \mathbb{R}^{1 \times p}$ and the gradient $\nabla f(x) \in \mathbb{R}^{p \times 1}$ are linked by equating the Taylor expansions (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 $H : \mathbb{R}^p \to \mathbb{R}^q$ and $G = g : \mathbb{R}^q \mapsto \mathbb{R}$, then $f = g \circ H : \mathbb{R}^p \to \mathbb{R}$ and one can compute its gradient as follow

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

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

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

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 as the function f along the 1-D half line

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

If f is differentiable at x, one has

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

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

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

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

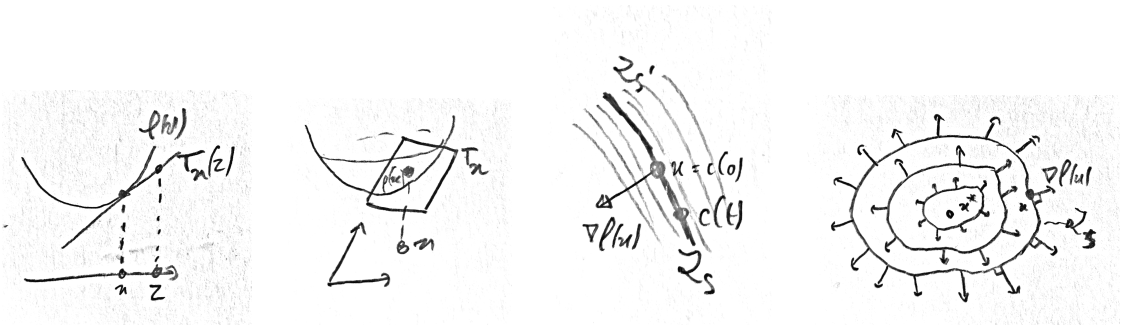


Figure 8: Left: First order Taylor expansion in 1-D and 2-D. Right: orthogonality of gradient and level sets and schematic of the proof.

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, lets consider around x inside $\mathcal{L}_s$ a smooth curve of the form $t \in \mathbb{R} \mapsto c(t)$ where $c(0) = x$. Then the function $h(t) \stackrel{\text{def.}}{=} f(c(t))$ is constant $h(t) = s$ since $c(t)$ belong to the level set. So $h'(t) = 0$. But at the same time, we can compute its derivate at $t = 0$ as follow

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

i.e. $h'(0) = \langle c'(0),\, \nabla f(x)\rangle = 0$, so that $\nabla f(x)$ is orthogonal to the tangent $c'(0)$ of the curve c, which lies in the tangent plane of $\mathcal{L}_s$ (as shown on Fig. 8, right). Since the curve c is arbitrary, the whole tangent plane is thus orthogonal to $\nabla f(x)$.

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

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

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 $\|u\| = r \to 0$, 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 use a fix step size $\tau_k = \tau$, or try to adapt τ_k at each iteration (see Fig. 9).

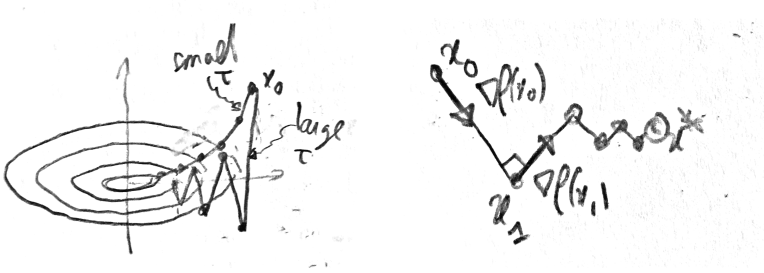


Figure 9: Influence of τ on the gradient descent (left) and optimal step size choice (right).

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.

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

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

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

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).

Remark 5 (Armijo rule). Instead of looking for the optimal τ, one can looks for an admissible τ which guarantees a large enough decay of the functional, in order to ensure convergence of the descent. Given some parameter $0 < \alpha < 1$ (which should be actually smaller than $1/2$ in order to ensure a sufficient decay), one consider a τ to be valid for a descent direction d_k (for instance $d_k = -\nabla f(x_k)$) if it satisfies

$$f(x_k + \tau d_k) \leqslant f(x_k) + \alpha\tau \langle d_k, \nabla f(x_k)\rangle \tag{14}$$

For small τ, one has $f(x_k + \tau d_k) = f(x_k) + \tau\langle d_k, \nabla f(x_k)\rangle$, so that, assuming d_k is a valid descent direction (i.e $\langle d_k, \nabla f(x_k)\rangle < 0$), condition (14) will always be satisfied for τ small enough (if f is convex, the set of allowable τ is of the form $[0, \tau_{\max}]$). In practice, one perform gradient descent by initializing τ very large, and decaying it $\tau \leftarrow \beta\tau$ (for $\beta < 1$) until (14) is satisfied. This approach is often called "backtracking" line search.

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

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

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

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

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

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

$$\|x_k - x^\star\| \leqslant \tilde{\rho}^\ell \|x_0 - x^\star\|. \tag{15}$$

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

$$\tau_\ell = \frac{2}{L+\mu} \quad \Longrightarrow \quad \tilde{\rho} \overset{\text{def.}}{=} \frac{L-\mu}{L+\mu} = 1 - \frac{2\varepsilon}{1+\varepsilon} \quad \text{where} \quad \varepsilon \overset{\text{def.}}{=} \mu/L. \tag{16}$$

Proof. One iterate of gradient descent reads

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

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

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

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

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

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

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

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

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

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

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

□

Note that when the condition number $\xi \overset{\text{def.}}{=} \mu/L \ll 1$ is small (which is the typical setup for ill-posed problems), then the contraction constant appearing in (16) scales like

$$\tilde{\rho} \sim 1 - 2\xi. \tag{17}$$

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

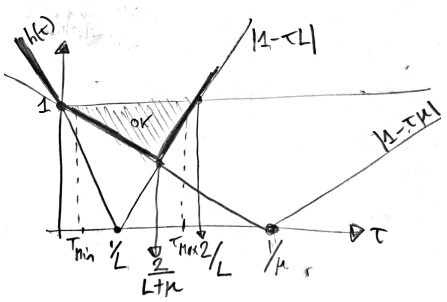


Figure 10: Contraction constant $h(\tau)$ for a quadratic function (right).

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

Proposition 5. *For $f(x) = \langle Cx, x\rangle - \langle b, x\rangle$, assuming the eigenvalue of C are bounded by L, then if $0 < \tau_k = \tau < 1/L$ is constant, then*

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

where

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

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

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

Now one has

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

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

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

Note also that

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

This shows that

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

This thus implies

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

where we have denoted

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

Since $x^\star$ can be chosen arbitrary, one can replace $\|x_0 - x^\star\|$ by $\mathrm{dist}(x_0, \operatorname{argmin} f)$. One has, for any ℓ, the following bound

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

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

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

Indeed, one has

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

□

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

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

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

$$f(x+\varepsilon) = f(x) + \langle \nabla f(x), \varepsilon \rangle + \frac{1}{2}\langle \partial^2 f(x)\varepsilon, \varepsilon \rangle + o(\|\varepsilon\|^2). \tag{18}$$

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 (18) 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

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

and thus

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

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

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

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

Remark 6 (Second order optimality condition). The first use of Hessian is to decide wether a point $x^\star$ with $\nabla f(x^\star)$ is a local minimum or not. Indeed, if $\partial^2 f(x^\star)$ is a positive matrix (i.e. its eigenvalues are strictly positive), then $x^\star$ is a strict local minimum. Note that if $\partial^2 f(x^\star)$ is only non-negative (i.e. some its eigenvalues might vanish) then one cannot deduce anything (such as for instance x^3 on $\mathbb{R}$). Conversely, if $x^\star$ is a local minimum then $\partial^2 f(x)$

Remark 7 (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 $H_k = \tau_k \mathrm{Id}_p$, and Newton's algorithm corresponds to using the inverse of the Hessian $H_k = [\partial^2 f(x_k)]^{-1}$. Note that

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

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

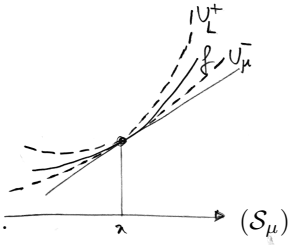
The last use of Hessian, that we explore next, is to study theoretically the convergence of the gradient descent. One simply needs to replace the boundedness of the eigenvalue of C of a quadratic function by a boundedness of the eigenvalues of $\partial^2 f(x)$ for all x. Roughly speaking, the theoretical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

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

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

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



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

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

$$\forall\, (x,x'), \quad f(x') + \langle \nabla f(x), x' - x\rangle + \frac{\mu}{2}\|x - x'\|^2 \leqslant f(x) \leqslant f(x') + \langle \nabla f(x'), x' - x\rangle + \frac{L}{2}\|x - x'\|^2. \tag{19}$$

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

$$\forall\, x, \quad \mu \mathrm{Id}_p \preceq \partial^2 f(x) \preceq L\mathrm{Id}_p \tag{20}$$

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

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

Proof. We prove (19), using Taylor expansion with integral remain

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

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

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

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

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

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

The relation (19) shows that a smooth (resp. strongly convex) functional is bounded by below (resp. above) by a quadratic tangential majorant (resp. minorant).

Condition (20) thus reads that the singular values of $\partial^2 f(x)$ should be contained in the interval $[\mu, L]$. The upper bound is also equivalent to $\|\partial^2 f(x)\|_{\text{op}} \leqslant L$ where $\|\cdot\|_{\text{op}}$ is the operator norm, i.e. the largest singular value. In the special case of a quadratic function of the form $\langle Cx,\, x\rangle - \langle b,\, x\rangle$ (recall that necessarily C is semi-definite symmetric positive for this function to be convex), $\partial^2 f(x) = C$ is constant, so that $[\mu, L]$ can be chosen to be the range of the eigenvalues of C.

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

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

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

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

$$f(x_k) - f(x^\star) \leqslant \frac{C}{\ell + 1}. \tag{21}$$

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

Proof. In the case where f is not strongly convex, we only prove (21) since the proof that x_k converges is more technical. Note indeed that if the minimizer $x^\star$ is non-unique, then it might be the case that the iterate x_k "cycle" while approaching the set of minimizer, but actually convexity of f prevents this kind of pathological behavior. For simplicity, we do the proof in the case $\tau_\ell = 1/L$, but it extends to the general case. The L-smoothness property imply (19), which reads

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

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

$$f(x_{k+1}) \leqslant f(x_k) - \frac{1}{L}\|\nabla f(x_k)\|^2 + \frac{1}{2L}\|\nabla f(x_k)\|^2 \leqslant f(x_k) - \frac{1}{2L}\|\nabla f(x_k)\|^2 \tag{22}$$

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

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

and plugging this in (22) shows

$$f(x_{k+1}) \leqslant f(x^\star) - \langle \nabla f(x_k),\, x^\star - x_k \rangle - \frac{1}{2L}\|\nabla f(x_k)\|^2 \tag{23}$$

$$= f(x^\star) + \frac{L}{2}\left(\|x_k - x^\star\|^2 - \|x_k - x^\star - \frac{1}{L}\nabla f(x_k)\|^2 \right) \tag{24}$$

$$= f(x^\star) + \frac{L}{2}\left(\|x_k - x^\star\|^2 - \|x^\star - x_{k+1}\|^2 \right). \tag{25}$$

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

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

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

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

which gives (21) for $C \stackrel{\text{def.}}{=} L\|x_0 - x^\star\|^2/2$.

If we now assume f is μ-strongly convex, then, using $\nabla f(x^\star) = 0$, one has $\frac{\mu}{2}\|x^\star - x\|^2 \leqslant f(x) - f(x^\star)$ for all x. Re-manipulating (25) gives

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

and hence

$$\|x_{k+1} - x^\star\| \leqslant \sqrt{\frac{L}{L+\mu}}\|x_{k+1} - x^\star\|, \tag{26}$$

which is the desired result. □

Note that in the low conditioning setting $\varepsilon \ll 1$, one retrieve a dependency of the rate (26) similar to the one of quadratic functions (17), indeed

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

5.3 Acceleration

The previous analysis shows that for L-smooth functions (i.e. with a hessian uniformly bounded by L, $\|\partial^2 f(x)\|_{\text{op}} \leqslant L$), the gradient descent with fixed step size converges with a speed on the function value $f(x_k) - \min f = O(1/k)$. Even using various line search strategies, it is not possible to improve over this rate. A way to improve this rate is by introducing some form of "momentum" extrapolation and rather consider a pair of variables (x_k, y_k) with the following update rule, for some step size s (which should be smaller than $1/L$)

$$\begin{cases} x_{k+1} = y_k - s\nabla f(y_k) \\ y_{k+1} = x_{k+1} + \beta_k(x_{k+1} - x_k) \end{cases}$$

where the extrapolation parameter satisfies $0 < \beta_k < 1$. The case of a fixed $\beta_k = \beta$ corresponds to the so-called "heavy-ball" method. In order for the method to bring an improvement for the $1/k$ "worse case"

rate (which does not means it improves for all possible case), one needs to rather use increasing momentum $\beta_k \to 1$, one popular choice being

$$\beta_k \frac{k-1}{k+2} \sim 1 - \frac{1}{k}.$$

This corresponds to the so-called "Nesterov" acceleration (although Nesterov used a slightly different choice, with the similar $1 - 1/k$ asymptotic behavior).

When using $s \leqslant 1/L$, one can show that $f(x_k) - \min f = O(\frac{\|x_0 - x^\star\|}{sk^2})$, so that in the worse case scenario, the convergence rate is improved. Note however that in some situation, acceleration actually deteriorates the rates. For instance, if the function is strongly convex (and even on the simple case $f(x) = \|x\|^2$), Nesterov acceleration does not enjoy linear convergence rate.

A way to interpret this scheme is by looking at a time-continuous ODE limit when $s \to 0$. On the contrary to the classical gradient descent, the step size here should be taker as $\tau = \sqrt{s}$ so that the time evolves as $t = \sqrt{s}k$. The update reads

$$\frac{x_{k+1} - x_k}{\tau} = (1 - 3/k)\frac{x_k - x_{k-1}}{\tau} - \nabla f(y_k)$$

which can be re-written as

$$\frac{x_{k+1} + x_{k-1} - 2x_k}{\tau^2} - \frac{3}{k\tau}\frac{x_k - x_{k-1}}{\tau} + \tau \nabla f(y_k) = 0.$$

Assuming $(x_k, y_k) \to (x(t), y(t))$, one obtains in the limit the following second order ODE

$$x''(t) + \frac{3}{t}x'(t) + \nabla f(x(t)) = 0 \quad \text{with} \quad \begin{cases} x(0) = x_0, \\ x'(0) = 0. \end{cases}$$

This corresponds to the movement of a ball in the potential field f, where the term $\frac{3}{t}x'(t)$ plays the role of a friction which vanishes in the limit. So for small t, the method is similar to a gradient descent $x' = -\nabla f(x)$, while for large t, it ressemble a Newtonian evolution $x'' = -\nabla f(x)$ (which keeps oscillating without converging). The momentum decay rate $3/t$ is very important, it is the only rule which enable the speed improvement from $1/k$ to $1/k^2$.

6 Mirror Descent and Implicit Bias

6.1 Bregman Divergences

We consider a smooth strictly convex "entropy" function ψ such that $\|\nabla\psi(x)\|$ goes to $+\infty$ as $x \to \partial\,\mathrm{dom}(\psi)$. We denote

$$\psi^*(u) \stackrel{\text{def.}}{=} \sup_{x \in \mathrm{dom}(\psi)} \langle u,\, x\rangle - \psi(x)$$

its Legendre transform. In this case of "Legendre-type" entropy function, $\nabla\psi : \mathrm{dom}(\psi) \to \mathrm{dom}(\psi^\star)$ and $\nabla\psi^*$ are bijection reciprocal one from the other.

One then defines the associated Bregman divergence

$$\mathrm{D}_\psi(x|y) \triangleq \psi(x) - \psi(y) - \langle \nabla\psi(y),\, x - y\rangle.$$

It is positive, convex in x (but not necessarily in y), not necessarily symmetric, and "distance-like".

For $\psi = \|\cdot\|^2$ one has $\nabla\psi = \nabla\psi^* = \mathrm{Id}$, and one recovers the Euclidean distance. For $\psi_{\mathrm{KL}}(x) = \sum_i x_i \log(x_i) - x_i + 1$ one has $\nabla\psi = \log$ and $\nabla\psi^* = \exp$, and one obtains the relative entropy, also known as Kullback-Leibler

$$\mathrm{D}_{\psi_{\mathrm{KL}}}(x|y) = \sum_i x_i \log(x_i/y_i) - x_i + y_i.$$

When $\psi_{\mathrm{Burg}}(x) = \sum_i -\log(x_i) + x_i - 1$ on $\mathbb{R}_+^d$, $\nabla\psi_{\mathrm{Burg}}(x) = \nabla\psi^*(x) = -1/x$ and associated divergence

$$\mathrm{D}_{\psi_{\mathrm{Burg}}}(x|y) = \sum_i -\log(y_i/x_i) - x_i/y_i + 1. \tag{27}$$

These examples can be generalized to power entropies

$$\psi_\alpha(x) \triangleq \sum_i \frac{|x_i|^\alpha - \alpha(x_i - 1) - 1}{\alpha(\alpha - 1)} \tag{28}$$

with special cases

$$\psi_1(x) \triangleq \psi_{\mathrm{KL}} = \sum_i x_i \log(x_i) - x_i + 1 \quad \text{and} \quad \psi_0(x) \triangleq \psi_{\mathrm{Burg}} = \sum_i -\log(x_i) + x_i - 1.$$

They are defined on $\mathbb{R}^d$ if $\alpha > 1$ and $\mathbb{R}_+^d$ if $\alpha \leqslant 1$.

Remark 8 (Matricial divergences). Given an entropy function $\psi_0(x)$ on vectors $x \in \mathbb{R}^d$ which is invariant under permutation of the indices, one lifts it to symmetric matrices $X \in \mathbb{R}^{d\times d}$ as

$$\psi(X) \triangleq \psi_0(\Lambda(X)) \quad \text{where} \quad X = U_X \operatorname{diag}(\Lambda(X)) U_X^\top$$

is the eigen-decomposition of X, where $\Lambda(X) = (\lambda_i(X))_{i=1}^d \in \mathbb{R}^d$ are the eigenvalues. Typically, if $\psi_0(x) = \sum_i h(x_i)$ then $\psi(X) = \mathrm{tr}(h(X))$ where h is extended to matrices as $h(X) \triangleq U_X \operatorname{diag}(h(\lambda_i(X)))U_X^\top$. If ψ_0 is convex and smooth, so is ψ, and

$$\nabla\psi(X) = U_X \operatorname{diag}(\nabla\psi_0(\Lambda(X)))U_X^\top.$$

For instance, if $h(s) = s\log(x) - s + 1$ is the Shannon entropy, this defines the quantum Shannon entropy as

$$\mathrm{D}_\psi(X) = \mathrm{tr}(X\log(X) - X\log(Y) - X + Y)$$

and if $h(s) = -\log(s)$ then $\mathrm{D}_\psi(X) = -\log\det(X)$.

Remark 9 (Cizard divergences). When defined on $\mathbb{R}_+^d$, these divergence should not be confounded with Cizar divergences which reads

$$\mathrm{C}_\psi(x|y) \overset{\text{def.}}{=} \sum_i y_i \psi(x_i/y_i) + \psi'_\infty \sum_{y_i=0} x_i,$$

which are jointly convex in x and y. Only for $\psi = \psi_{\mathrm{KL}}$ one has $\mathrm{D}_{\psi_{\mathrm{KL}}} = \mathrm{C}_{\psi_{\mathrm{KL}}}$.

6.2 Mirror descent

We consider the following implicit stepping

$$x_{k+1} = \operatorname*{argmin}_{x\in\mathrm{dom}(\psi)} f(x) + \frac{1}{\tau}\mathrm{D}_\psi(x|x_k).$$

Its explicit version then reads by Taylor expanding f at x_k

$$\begin{aligned} x_{k+1} &= \operatorname*{argmin}_{x\in\mathrm{dom}(\psi)} f(x_k) + \langle x - x_k, \nabla f(x_k)\rangle + \frac{1}{\tau}\mathrm{D}_\psi(x|x_k), \\ &= \operatorname*{argmin}_{x\in\mathrm{dom}(\psi)} \langle x, \nabla f(x_k)\rangle + \frac{1}{\tau}\mathrm{D}_\psi(x|x_k). \end{aligned}$$

The fact that ψ is Legendre type allows to ignore the constraint, and the solution satisfies the following first order condition

$$\nabla f(x_k) + 1/\tau[\nabla\psi(x_{k+1}) - \nabla\psi(x_k)] = 0$$

so that it can be explicitly computed

$$x_{k+1} = (\nabla\psi^*)[\nabla\psi(x_k) - \tau\nabla f(x_k)] \tag{29}$$

For $\psi = \|\cdot\|^2/2$ one recovers the usual Euclidean gradient descent. For $\psi(x) = \sum_i x_i \log(x_i)$, this defines the multiplicative updates

$$x_{k+1} = x_k \odot \exp(-\tau\nabla f(x_k))$$

where $\odot$ is the entry-wise multiplication of vectors.

Note that introducing the "dual" variable $u_k \triangleq \nabla\psi(x_k)$, one has

$$u_{k+1} = u_k - \tau h(u_k) \quad \text{where} \quad h(u) \triangleq \nabla f(\nabla\psi^*(u)). \tag{30}$$

Note however that in general h is not a gradient field, so this is not in general a gradient flow.

Mirror flow. When $\tau \to 0$, one obtains the following expansion

$$x_{k+1} = (\nabla\psi^*)[\nabla\psi(x_k)] - \tau[\partial^2\psi^*](\nabla\psi(x_k)) \times \nabla f(x_k) + o(\tau)$$

so that defining $x(t) = x_k$ for $t = k\tau$ the limit is the following flow

$$\dot{x}(t) = -H(x(t))\nabla f(x(t)) \quad \text{where} \quad H(x) \triangleq [\partial^2\psi^*](\nabla\psi(x)) = [\partial^2\psi(x)]^{-1} \tag{31}$$

so that this is a gradient flow on a very particular type of manifold, of "Hessian type". Note that if $\psi = f$, then one recovers the flow associated to Newton's method.

Convergence. Convergence theory (ensuring convergence and rates) for mirror descent is the same as for the usual gradient descent, and one needs to consider relative L-smoothness, and if possible also relative μ-strong convexity,

$$\mu\,\mathrm{D}_\psi \leqslant \mathrm{D}_f \leqslant L\,\mathrm{D}_\psi \quad \Longleftrightarrow \quad \forall\, x,\ \mu\partial^2\psi(x) \leqslant \partial^2 f(x) \leqslant L\partial^2\psi(x).$$

If $L < +\infty$, then one has $f(x_k) - f(x^\star) \leqslant O(\mathrm{D}_\psi(x^\star|x_0)/k)$ while if both $0 < \mu \leqslant L < +\infty$, then $\mathrm{D}_\psi(x_k|x^\star) \leqslant O(\mathrm{D}_\psi(x^\star|x_0)(1-\mu/L)^k)$. The advantages of using Bregman geometry are two-fold: this can improves the conditioning μ/L (some function might be non-smooth for the Euclidean geometry but smooth for some Bregman geometry, and can avoid introducing constraint in the optimization problem) and this can also lower the radius of the domain $\mathrm{D}_\psi(x^\star|x_0)$. For instance, assuming the solution belongs to the simplex, and using $x_0 = \mathbb{1}_d/d$, then $\mathrm{D}_{\psi_{\mathrm{KL}}}(x^\star|x_0) \leqslant \log(d)$ whereas for the ℓ^2 Euclidean distance, one only has the bound $\|x^\star - x_0\|^2 \leqslant d$.

6.3 Re-parameterized flows

One can consider a change of variable $x = \varphi(z)$ where $\varphi : \mathbb{R}^p \mapsto \mathcal{X} \subset \mathbb{R}^d$ is a smooth map, and perform the gradient descent on the function $g(z) \triangleq f(\varphi(z))$. Then one has

$$\nabla g(z) = [\partial\varphi(z)]^\top \nabla f(x)$$

so that, denoting $z(t)$ the gradient flow $\dot{z} = -\nabla g(z)$ of g, and $x(t) \triangleq \varphi(z(t))$, one has $\dot{x}(t) = [\partial\varphi(z(t))]\dot{z}(t)$ and thus $x(t)$ solves the following equation

$$\dot{x} = -Q(z)\nabla f(x) \quad \text{with} \quad Q(z) \triangleq [\partial\varphi(z)][\partial\varphi(z)]^\top \in \mathcal{S}_+^{d\times d}$$

So unless φ is a bijection, this is not a gradient flow over the x variable. If φ is a bijection, then this is a gradient flow associated to the field of tensors ("manifold") $Q(\varphi^{-1}(x))$. The issue is that even in this case, in general H might fail to be a Hessian manifold, so this does not correspond to a mirror descent flow.

Dual parameterization If ψ is an entropy function, then the parametrization $x = \nabla\psi^*(z)$, i.e. $\varphi = \nabla\psi^*$, then $Q(z) = [\partial^2\psi^*(z)]^2$, i.e. $Q(\varphi^{-1}(x)) = [\partial^2\psi(x)]^{-2}$ is not of Hessian-type in general, but rather a squared-Hessian manifold. For instance, when $\psi^*(z) = \exp(z)$, then $Q(\varphi^{-1}(x)) = \mathrm{diag}(1/x_i^2)$, which surprisingly is the hessian metric associated to Burg's entropy $-\sum_i \log(x_i)$.

Example: power-type parameterization We consider power entropies (28), on $\mathbb{R}_+^d$, for $\alpha \leqslant 1$, for which

$$H(x) = [\partial^2\psi(x)]^{-1} \propto \mathrm{diag}(x_i^{2-\alpha}).$$

Remark than when using the parameterization $x = \varphi(z) = (z_i^b)_i$ then

$$Q(\varphi^{-1}(x)) = [\partial\varphi(z)][\partial\varphi(z)]^\top \propto \mathrm{diag}(z_i^{2(b-1)}) = \mathrm{diag}(x_i^{2(b-1)/b})$$

so if one selects $2(1-1/b) = 2-\alpha$ i.e. $2/b = \alpha$, the re-parameterized flow is equal to the flow on a Hessian manifold. For instance, when setting $b = 2$, $\alpha = 1$, i.e. using the parmeterization $x = z^2$, one retrieves the flow on the manifold for the Shannon entropy ("Fisher-Rao" geometry). Note that when $b \to +\infty$, one obtains $\alpha = 0$, i.e. the flow is the one of the Burg's entropy $\psi(x) = -\sum_i \log(x_i)$ (which we saw above as also being associated to the parameterization $x = \exp(z)$).

Counter-example: SDP matrices We now consider semi-definite symmetric matrices $X \in \mathcal{S}_+^{d\times d}$, together with the parameterization $X = \varphi(Z) = ZZ^\top$ for $Z \in \mathbb{R}^{d\times d}$. In this case, denoting $g(Z) = f(ZZ^\top)$, one has

$$\nabla g(Z) = [\nabla f(X) + \nabla f(X)^\top]Z$$

so that the flow $\dot{Z} = -\nabla g(Z)$ is equivalent to the following flow on symmetric (and it maintains positivity as well)

$$\dot{X} = X[\nabla_S f(X)] + [\nabla_S f(X)]X \tag{32}$$

where the symmetric gradient is

$$\nabla_S f(X) \triangleq [\nabla f(X)] + [\nabla f(X)]^\top$$

So most likely (32) cannot be written as a usual gradient flow on a manifold which would be a hessian of a convex function. To mimic the diagonal case (or vectorial case above), the most natural quantitate would have been the spectral entropy $\psi(X) \triangleq \mathrm{tr}(X\log(X) - X + \mathrm{Id})$, whose gradient is $\log(X)$, but there is no closed form expression for the derivative of the log unfortunately. Another simpler approach to mimic ψ_{-1} is to use $\psi(X) = -\mathrm{tr}(\log(X)) = -\log\det(X)$, because the Hessian and its inverse can be computed

$$\partial^2\psi(X) : S \mapsto -X^{-1}SX^{-1}.$$

6.4 Implicit Bias

We consider the problem

$$\min_{x\in\mathbb{R}^d} f(x) = L(Ax) \triangleq \sum_i \ell(\langle a_i, x\rangle, y_i),$$

where the loss is coercive such that $\ell(\cdot, y_i)$ has a unique minimizer at y_i. The typical example is $f(x) = \|Ax - y\|^2$ for $\ell(u, v) = (u - v)^2$. We do not impose that L is convex, and simply assumes convergence of the considered optimization method to the set of global minimizers. The set of global minimizers is thus the affine space

$$\mathrm{argmin}\, f = \{x \;;\; Ax = y\}.$$

The simplest optimization method is just gradient descent

$$x_{k+1} = x_k - \tau\nabla f(x_k) \quad \text{where} \quad \nabla f(x) = A^\top\nabla L(Ax).$$

As $\tau \to 0$, one defines $x(t) = x_k$ for $t = k\tau$ and consider the flow

$$\dot{x}(t) = -\nabla f(x(t)).$$

The implicit bias of the descent (and the flow) is given by the orthogonal projection.

Proposition 7. *If $x_k \to x^\star \in \operatorname{argmin} f$, then*

$$x^\star = \operatorname*{argmin}_{x \in \operatorname{argmin} f} \|x - x_0\|.$$

The following Proposition, whose proof can be found in [**?**] generalizes this proposition to the case of an arbitrary mirror flow.

Proposition 8. *If x_k defined by (29) (resp. $x(t)$ defined by (31)) is such that x_k (resp. $x(t)$) converges to $x^\star \in \operatorname{argmin} f$, then*

$$x^\star = \operatorname*{argmin}_{x \in \operatorname{argmin} f} \mathrm{D}_\psi(x|x_0). \tag{33}$$

Proof. From the dual variable evolution (30), since $\nabla f(x) \in \operatorname{Im}(A^\top)$, one has that $y_k - y_0 \in \operatorname{Im}(A^\top)$, so that in the limit

$$y^\star - y_0 = \nabla\psi(x^\star) - \nabla\psi(x_0) \in \operatorname{Im}(A^\top). \tag{34}$$

Note that $\nabla \mathrm{D}_\psi(x|x_0) = \nabla\psi(x) - \nabla\psi(x_0)$, and $\operatorname{Im}(A^\top) = \operatorname{Ker}(A)^\perp$ is the space orthogonal to $\operatorname{argmin} f$ so that (34) are the optimality conditions of the strictly convex problem (33). □

In particular, for the Shannon entropy (equivalently when using the $x = z^2$ parameterization), as $x_0 \to 0$, by doing the expansion of $\mathrm{KL}(x|x_0)$ one has

$$x^\star \to \operatorname*{argmin}_{x \in \operatorname{argmin} f, x \geqslant 0} \sum_i |\log((x_0)_i)| x_i,$$

which is a weighted ℓ^1 norm (so in particular it induces sparsity in the solution, it is a Lasso-type problem).

When using more general parameterizations of the form $x = z^b$ for $b > 0$, this corresponds to using the power entropy ψ_α for $\alpha = 2/b$, and one can check that the associated limit bias for small x_0 is still an ℓ^1, but with a different weighting scheme. For $x = \exp(z)$ (or $b \to +\infty$) one obtains Burg's entropy defined in (27) so that the limit bias is $\sum_i x_i/(x_0)_i$. The use of $x = z^2$ parameterization (which can be generalized to $x = u \odot v$ for signed vectors) was introduced in [**?**], and its associated implicit regularization is detailed in [**?**, **?**]. It is possible to analyze this sparsity-inducing behavior in a quantitative way, see for instance [**?**, Thm.2] One can generalize this parameterization to arbitrary (not only positive vector) by using $x = u^2 - v^2$ or $x = u \odot v$ and the same type of bias appears, with now rather a (weighted) ℓ^1 norm.

7 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.

7.1 Penalized Least Squares

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

$$\min_{x \in \mathbb{R}^p} f_\lambda(x) \overset{\text{def.}}{=} \frac{1}{2}\|Ax - y\|^2 + \lambda R(x) \tag{35}$$

where $R(x)$ is the regularizer and $\lambda \geqslant 0$ 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. $R(x) \to +\infty$ as $\|x\| \to +\infty$. 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 9. *If $(x_{\lambda_k})_k$ is a sequence of minimizers of f_λ, then this sequence is bounded, and any accumulation $x^\star$ is a solution of the constrained optimization problem*

$$\min_{Ax=y} R(x). \tag{36}$$

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

$$\frac{1}{2}\|Ax_{\lambda_k} - y\|^2 + \lambda_k R(x_{\lambda_k}) \leqslant \lambda_k R(x_0). \tag{37}$$

Since all the term are positive, one has $R(x_{\lambda_k}) \leqslant R(x_0)$ so that $(x_{\lambda_k})_k$ is bounded by coercivity of R. Then also $\|Ax_{\lambda_k} - y\| \leqslant \lambda_k R(x_0)$, and passing to the limit, one obtains $Ax^\star = y$. And passing to the limit in $R(x_{\lambda_k}) \leqslant R(x_0)$ one has $R(x^\star) \leqslant R(x_0)$ which shows that $x^\star$ is a solution of (36). □

7.2 Ridge Regression

Ridge regression is by far the most popular regularizer, and corresponds to using $R(x) = \|x\|^2_{\mathbb{R}^p}$. Since it is strictly convex, the solution of (35) is unique

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

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 formula.

Proposition 10. *One has*

$$x_\lambda = (A^\top A + \lambda \mathrm{Id}_p)^{-1} A^\top y, \tag{38}$$

$$= A^\top (AA^\top + \lambda \mathrm{Id}_n)^{-1} y. \tag{39}$$

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

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

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

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 (38) while if $n < p$ one should rather use (39).

Pseudo-inverse. As $\lambda \to 0$, then $x_\lambda \to x_0$ which is, using (36)

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

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

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

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

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

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 Values Decomposition (SVD). The matrix A^+ is often called the Moore-Penrose pseudo-inverse.

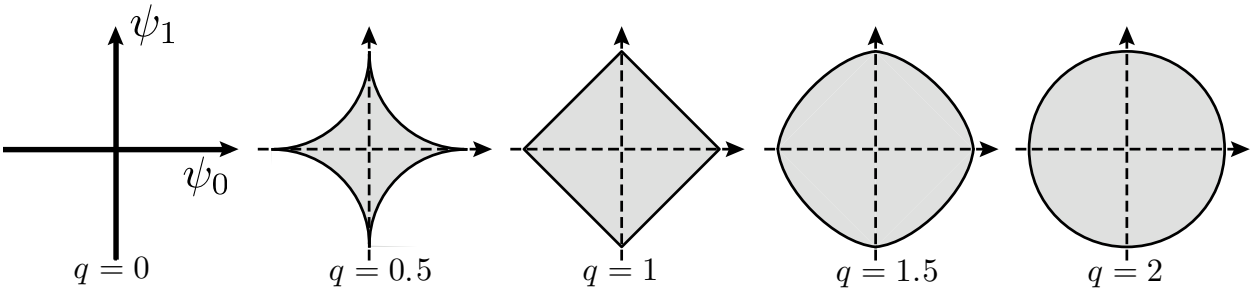


Figure 11: ℓ^q balls $\{x \,;\, \sum_k |x_k|^q \leqslant 1\}$ for varying q.

7.3 Lasso

The Lasso corresponds to using a ℓ^1 penalty

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

The underlying idea is that solutions x_λ of a Lasso problem

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

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 $q \to 0$ (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 7.4 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.

Proposition 11. *When $n = p$ and $A = \mathrm{Id}_n$, one has*

$$\operatorname*{argmin}_{x \in \mathbb{R}^p} \frac{1}{2}\|x - y\|^2 + \lambda \|x_1\| = S_\lambda(x) \quad \textit{where} \quad S_\lambda(x) = (\mathrm{sign}(x_k) \max(|x_k| - \lambda, 0))_k$$

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 $F'(x) = x - y + \lambda$ wich is 0 at $x = y - \lambda$. The minimum is at $x = y - \lambda$ for $\lambda \leqslant y$, and stays at 0 for all $\lambda > y$. The problem is symmetric with respect to the switch $x \mapsto -x$. □

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

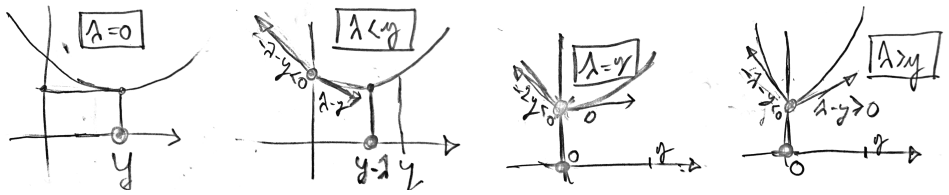


Figure 12: Evolution with λ of the function $F(x) \overset{\text{def.}}{=} \frac{1}{2}\|\cdot - y\|^2 + \lambda|\cdot|$.

7.4 Iterative Soft Thresholding

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

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

and we introduce for any fixed x' the function

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

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

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

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

$$x_{k+1} \overset{\text{def.}}{=} \operatorname*{argmin}_x f_\tau(x, x_k) \tag{40}$$

which by construction satisfies

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

Proposition 12. *The iterates x_k defined by (40) satisfy*

$$x_{k+1} = S_{\lambda\tau}\left(x_k - \tau A^\top(Ax_k - y)\right) \tag{41}$$

where $S_\lambda(x) = (s_\lambda(x_m))_m$ where $s_\lambda(r) = \mathrm{sign}(r)\max(|r| - \lambda, 0)$ 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 (11) shows that the minimizer of $f_\tau(x, x')$ is thus indeed $S_{\lambda\tau}(x' - \tau A^\top(Ax' - y))$ as claimed. □

Equation (41) 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.

8 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.

8.1 Minimizing Sums and Expectation

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

$$\min_{x\in\mathbb{R}^p} f(x) \stackrel{\text{def.}}{=} \frac{1}{n}\sum_{i=1}^{n} f_i(x) \tag{42}$$

or even expectations of the form

$$\min_{x\in\mathbb{R}^p} f(x) \stackrel{\text{def.}}{=} \mathbb{E}_{\mathbf{z}\sim\pi}(f(x,\mathbf{z})) = \int_{\mathcal{Z}} f(x,z)\mathrm{d}\pi(z). \tag{43}$$

Problem (42) can be seen as a special case of (43), 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 (42) as a discretized "empirical" version of (43) when drawing $(z_i)_i$ i.i.d. according to $\mathbf{z}$ and defining $f_i(x) = f(x,z_i)$. In this setup, (42) converges to (43) as $n \to +\infty$.

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

$$f_i(x) = \ell(\langle a_i,\, x\rangle, y_i) \quad \text{and} \quad f(x,z) = \ell(\langle a,\, x\rangle, y) \tag{44}$$

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 below the methods on binary logistic classification, where

$$L(s,y) \stackrel{\text{def.}}{=} \log(1+\exp(-sy)). \tag{45}$$

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 (42), the stochastic methods we detail next work verbatim (with the same convergence guarantees) in the expectation case (43). 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 notÂ 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 determinstic iterations in smaller chunks. Another advantage is that they are quite easy to parallelize.

8.2 Batch Gradient Descent (BGD)

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

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

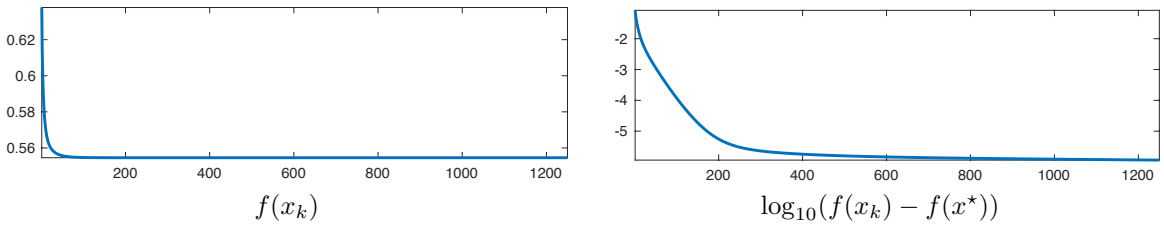


Figure 13: Evolution of the error of the BGD for logistic classification.

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

$$\nabla f(x) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x) \tag{46}$$

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

For ERM-type functions of the form (44), one can do the Taylor expansion of f_i

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

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

$$\nabla f_i(x) = \ell'(\langle a_i,\, x\rangle, y_i) a_i. \tag{47}$$

For the logistic loss, one has

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

8.3 Stochastic Gradient Descent (SGD)

For very large n, computing the full gradient ∇f as in (46) is prohibitive. The idea of SGD is to trade this exact full gradient by an inexact proxy using a single functional f_i where i is drawn uniformly at random. The main idea that makes this work is that this sampling scheme provides an unbiased estimate of the gradient, in the sense that

$$\mathbb{E}_{\mathbf{i}} \nabla f_{\mathbf{i}}(x) = \nabla f(x) \tag{48}$$

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

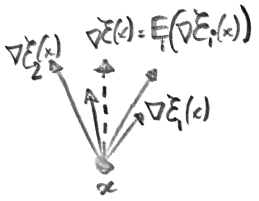


Figure 14: Unbiased gradient estimate

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

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

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

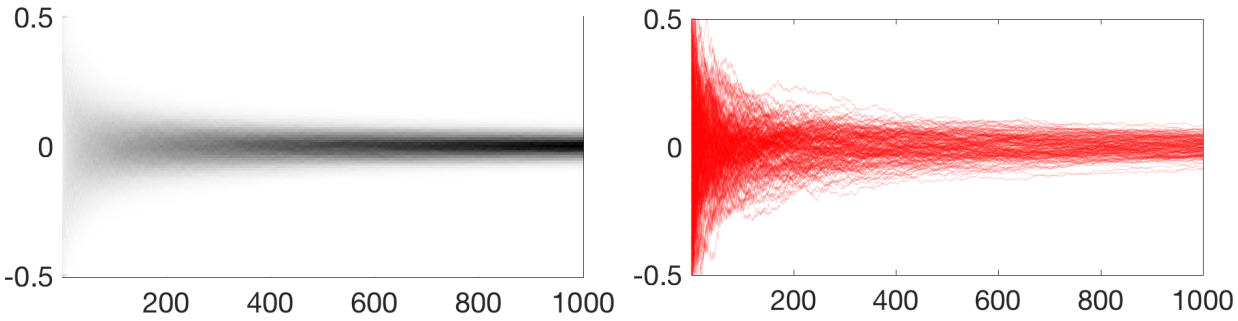


Figure 16: Display of a large number of trajectories $k \mapsto x_k \in \mathbb{R}$ generated by several runs of SGD. On the top row, each curve is a trajectory, and the bottom row displays the corresponding density.

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

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

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

$$\tau_k \overset{\text{def.}}{=} \frac{\tau_0}{1 + k/k_0} \tag{49}$$

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

Figure 15: Schematic view of SGD iterates

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 $x^\star = 0$. Here the distribution of x_0 is uniform on $[-1/2, 1/2]$.

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

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

$$\mathbb{E}(\|x_k - x^\star\|^2) \leqslant \frac{R}{k+1} \quad \textit{where} \quad R = \max(\|x_0 - x^\star\|, C^2/\mu^2), \tag{50}$$

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

Proof. By strong convexity, one has

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

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

$$\langle \nabla f(x_k) - \nabla f(x^\star),\, x_k - x^\star \rangle = \langle \nabla f(x_k),\, x_k - x^\star \rangle \geqslant \mu \|x_k - x^\star\|^2. \tag{51}$$

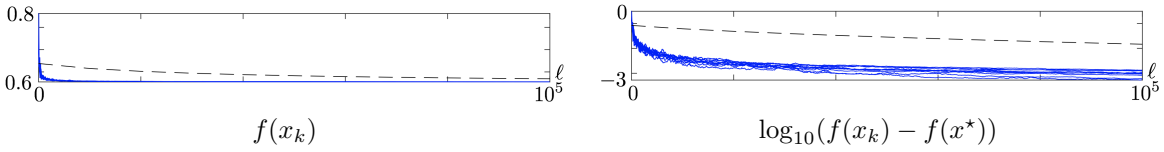


Figure 17: Evolution of the error of the SGD for logistic classification (dashed line shows BGD).

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

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

where we used the fact (48) that the gradient is unbiased. Taking now the full expectation with respect to all the other previous iterates, and using (51) one obtains

$$\mathbb{E}(\|x_{k+1} - x^\star\|^2) \leqslant \mathbb{E}(\|x_k - x^\star\|^2) - 2\mu\tau_k \mathbb{E}(\|x_k - x^\star\|^2) + \tau_k^2 C^2 = (1 - 2\mu\tau_k)\mathbb{E}(\|x_k - x^\star\|^2) + \tau_k^2 C^2. \quad (52)$$

We show by recursion that the bound (50) holds. We denote $\varepsilon_k \overset{\text{def.}}{=} \mathbb{E}(\|x_k - x^\star\|^2)$. Indeed, for $k = 0$, this it is true that

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

We now assume that $\varepsilon_k \leqslant \frac{R}{k+1}$. Using (52) in the case of $\tau_k = \frac{1}{\mu(k+1)}$, one has, denoting $m = k + 1$

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

□

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

Figure 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.

8.4 Stochastic Gradient Descent with Averaging (SGA)

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

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

and output as estimated weight vector the Cesaro average

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

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

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

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

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

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

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

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

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

8.5 Stochastic Averaged Gradient Descent (SAG)

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

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

The algorithm reads

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

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

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

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

for some $0 < \rho < 1$.

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 (43)).

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

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 presents methods to compute recursively in an optimal manner these gradients. The purpose of this approach is to automatize this computational step.

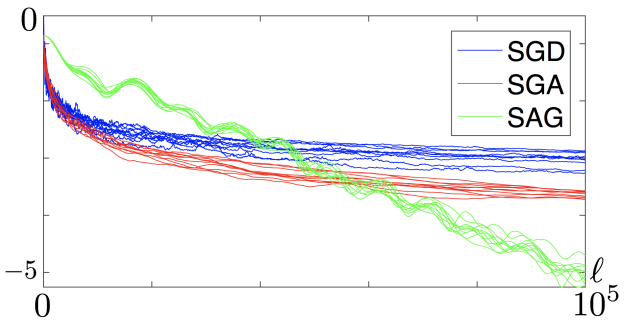


Figure 18: Evolution of $\log_{10}(f(x_k) - f(x^\star))$ for SGD, SGA and SAG.

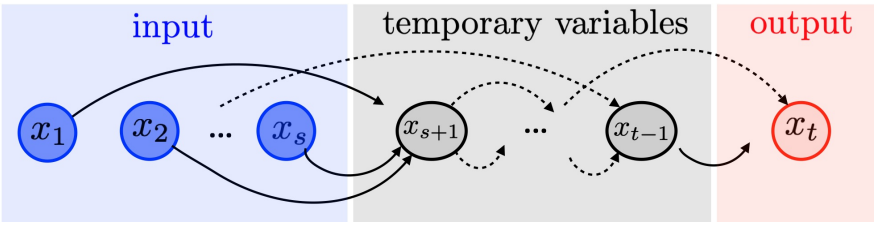


Figure 19: A computational graph.

9.1 Finite Differences and Symbolic Calculus

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

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

requires $p + 1$ evaluations of f, where we denoted $\delta_k = (0, \ldots, 0, 1, 0, \ldots, 0)$ 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 takes 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 $x = (x_1, \ldots, x_s)$ are the input variables. We assume that f is implemented in an algorithm, with intermediate variable $(x_{s+1}, \ldots, x_t)$ where t is the total number of variables. The output is x_t, and we thus denote $x_t = f(x)$ this function. We denote $x_k \in \mathbb{R}^{n_k}$ the dimensionality of the variables. The goal is to compute the derivatives $\frac{\partial f(x)}{\partial x_k} \in \mathbb{R}^{n_t \times n_k}$ for $k = 1, \ldots, s$. For the sake of simplicity, one can assume in what follows that $n_k = 1$ so that all the involved quantities are scalar (but if this is not the case, beware that the order of multiplication of the matrices of course matters).

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

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

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 node of this graph are thus conveniently ordered by their indexing, and the directed edges only link a variable to another

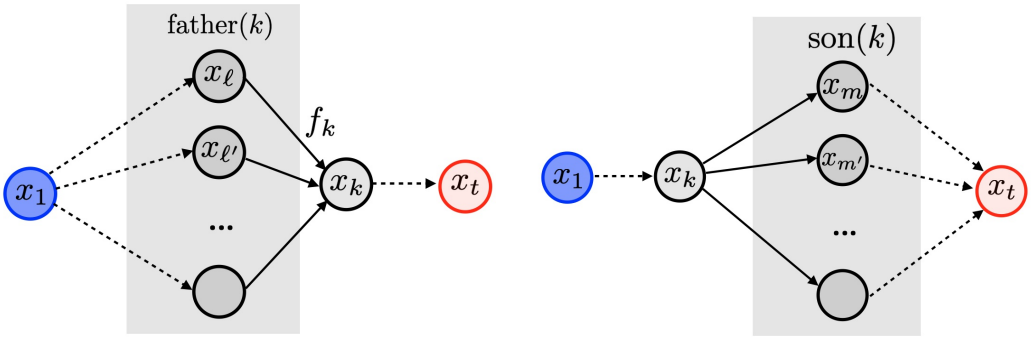


Figure 20: Relation between the variable for the forward (left) and backward (right) modes.

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 $x_1, x_2, \ldots, x_p$ (we only write thing for the first variable, the method being of course the same with respect to the other ones).

The method initialize the derivative of the input nodes

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

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

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

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 $[\ldots]$ an informal notation, since here $\frac{\partial x_k}{\partial x_\ell}$ should be interpreted not as a numerical variable but needs to be interpreted as derivative of the function f_k, which can be evaluated on the fly (we assume that the derivative of the function involved are accessible in closed form).

Assuming all the involved functions $\frac{\partial f_k}{\partial x_k}$ have the same complexity (which is likely to be the case if all the n_k are for instance scalar or have the same dimension), and that the number of 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 $\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_p}$). For a large p, this is prohibitive.

Simple example. We consider the fonction

$$f(x, y) = y \log(x) + \sqrt{y \log(x)} \tag{53}$$

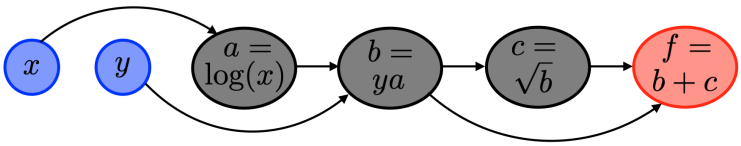


Figure 21: Example of a simple computational graph.

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

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

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 $f(x_1, \ldots, x_s)$ is a function implemented using these overloaded basic functions, one has

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

and this evaluation is equivalent to applying the forward mode of automatic differentiation to compute $\frac{\partial f}{\partial x_1}(x_1, \ldots, x_s)$ (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 the all the inner nodes.

The method initialize the derivative of the final node

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

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

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

The notation "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 follow

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

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

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

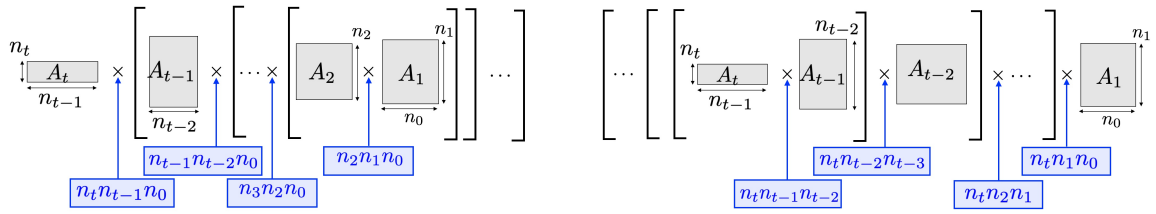


Figure 22: Complexity of forward (left) and backward (right) modes for composition of functions.

Simple example. We consider once again the fonction $f(x)$ of (53), 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 more necessitates two passes.

9.5 Feed-forward Compositions

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

$$f = f_t \circ f_{t-1} \circ \ldots \circ f_2 \circ f_1 \tag{54}$$

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

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

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

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

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

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

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

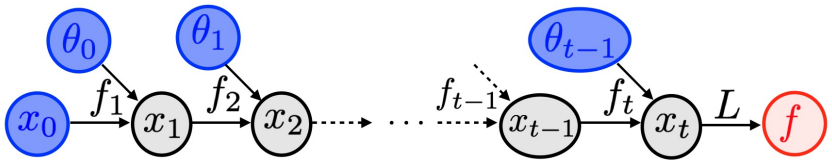


Figure 23: Computational graph for a feedforward architecture.

We note that the computation of the product $A \times B$ of $A \in \mathbb{R}^{n \times p}$ with $B \in \mathbb{R}^{p \times q}$ necessitates npq operations. As shown on Figure 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

$$\forall\, k = 1, \ldots, t, \quad x_k = f_k(x_{k-1}, \theta_{k-1}) \tag{55}$$

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

$$f(\theta) \overset{\text{def.}}{=} L(x_t) \tag{56}$$

where $L : \mathbb{R}^{n_t} \to \mathbb{R}$ 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

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

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

$$z_{k-1} = [\partial_x f_k(x_{k-1}, \theta_{k-1})]^\top z_k \quad \text{and} \quad \nabla_{\theta_{k-1}} f = [\partial_\theta f_k(x_{k-1}, \theta_{k-1})]^\top (\nabla_{x_k} f) \tag{57}$$

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

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

$$\forall\, x_{k-1} \in \mathbb{R}^{n_{k-1}}, \quad f_k(x_{k-1}, \theta_{k-1}) = \rho(\theta_{k-1} x_{k-1}) \tag{58}$$

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 $\nabla_{x_k} f$)

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

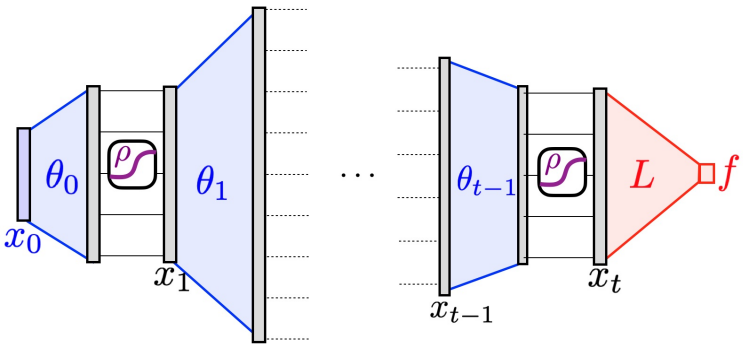


Figure 24: Multi-layer perceptron parameterization.

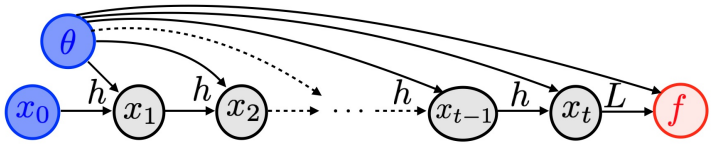


Figure 25: Computational graph for a recurrent architecture.

Link with adjoint state method. One can interpret (55) 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 $x_k \to x(k\tau)$ when $k \to +\infty$ and $k\tau \to t$. Imposing then the structure

$$f_k(x_{k-1}, \theta_{k-1}) = x_{k-1} + \tau u(x_{k-1}, \theta_{k-1}, k\tau) \tag{59}$$

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

$$\dot{x}(t) = u(x(t), \theta(t), t) \tag{60}$$

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

Denoting $z(t) = \nabla_{x(t)} f(\theta)$ the "adjoint" vector field, the discrete equations (62) 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 $\frac{1}{\tau} \nabla_{\theta_{k-1}} f \to \nabla_{\theta(t)} f(\theta)$

9.7 Recurrent Architectures

Parametric recurrent functions are obtained by using the same parameter $\theta = \theta_k$ and $f_k = h$ recursively in (58), so that

$$\forall\, k = 1, \ldots, t, \quad x_k = h(x_{k-1}, \theta). \tag{61}$$

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 (56)). Figure 25, bottom, displays the associated computational graph.

The back-propagation then operates as

$$\nabla_{x_{k-1}} f = [\partial_x h(x_{k-1}, \theta)]^\top \nabla_{x_k} f \quad \text{and} \quad \nabla_\theta f = \nabla_\theta L(x_t, \theta) + \sum_k [\partial_\theta h(x_{k-1}, \theta)]^\top \nabla_{x_k} f. \tag{62}$$

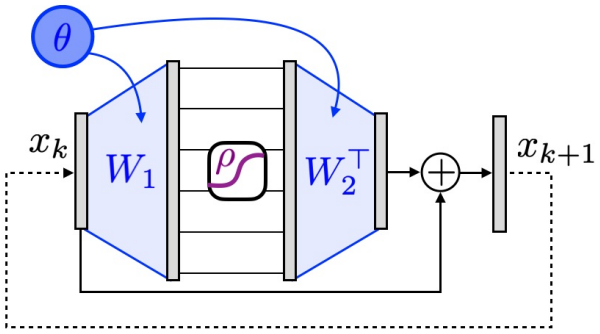


Figure 26: Recurrent residual perceptron parameterization.

Similarly, writing $h(x,\theta) = x + \tau u(x,\theta)$, letting $(k, k\tau) \to (+\infty, t)$, 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 $f(\theta) = L(x(T),\theta)$

$$\dot{z}(t) = -[\partial_x u(x(t),\theta)]^\top z(t) \quad \text{and} \quad \nabla_\theta f(\theta) = \nabla_\theta L(x(T),\theta) + \int_0^T [\partial_\theta f(x(t),\theta)]^\top z(t)\mathrm{d}t. \tag{63}$$

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 (65) 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 (63).

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 method perform this recursively in order to only require $\log(t)$ stored values and a $\log(t)$ increase on the numerical complexity.

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

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

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

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 (60) 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 equation 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 $x^\star(\theta)$ which is thus a fixed point

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

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 $f^\star(\theta) = L(x^\star(\theta), \theta)$. Indeed, one has

$$\nabla f^\star(\theta) = [\partial x^\star(\theta)]^\top (\nabla_x L(x^\star(\theta), \theta)) + \nabla_\theta L(x^\star(\theta), \theta). \tag{64}$$

Using the implicit function theorem one can compute the Jacobian as

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

In practice, one replace in these formulas $x^\star(\theta)$ 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 bypass 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

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

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

$$x_{k+1} = x_k - \tau \nabla \mathcal{E}(x_k, \theta). \tag{65}$$

This has the form (59) when using the vector field $u(x, \theta) = \nabla \mathcal{E}(x_k, \theta)$.

Using formula (64) in this case where $h = \nabla \mathcal{E}$, one obtains

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

In the special case where the function $f(\theta)$ is the minimized function itself, i.e. $f(\theta) = \mathcal{E}(x^\star(\theta), \theta)$, i.e. $L = \mathcal{E}$, then one can apply the implicit function theorem formula (64), which is much simpler since in this case $\nabla_x L(x^\star(\theta), \theta) = 0$ so that

$$\nabla f^\star(\theta) = \nabla_\theta L(x^\star(\theta), \theta). \tag{66}$$

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 $x_0 \overset{\text{def.}}{=} (u_0, v_0) = (1_n, 1_m)$

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

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

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

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

Denoting $\mathcal{K} \overset{\text{def.}}{=} \begin{pmatrix} 0 & K \\ K^\top & 0 \end{pmatrix} \in \mathbb{R}^{(n+m)\times(n+m)}$, one can re-write these iterations in the form (61) using

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

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 $x_k \to x^\star(\theta)$, one finds a minimizer of $\mathcal{E}$, so that $\nabla_x L(x^\star(\theta),\theta) = 0$ and thus the gradient of $f^\star(\theta) = \mathcal{E}(x^\star(\theta),\theta)$ can be computed using (66) i.e.

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