Mathematical Foundations of Data Sciences

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

March 19, 2020

Chapter 11

Gradient Descent Methods

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

11.1 Motivation in Machine Learning

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

$$\inf_{x \in \mathbb{R}^p} f(x), \tag{11.1}$$

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

$$\min_{x \in \mathbb{R}^p} f(x). \tag{11.2}$$

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

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

$$f(x) = \frac{1}{2} \sum_{i=1}^n (y_i - \langle x, a_i \rangle)^2 = \frac{1}{2} \|Ax - y\|^2, \tag{11.3}$$

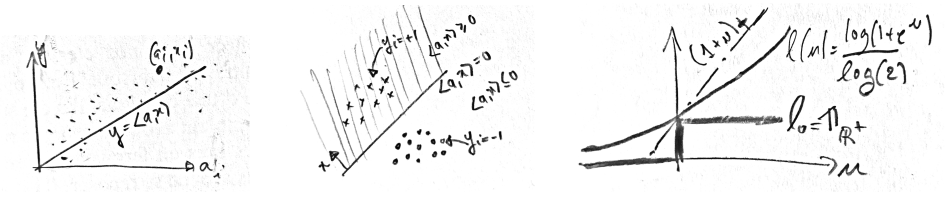


Figure 11.1: Left: linear regression, middle: linear classifier, right: loss function for classification.

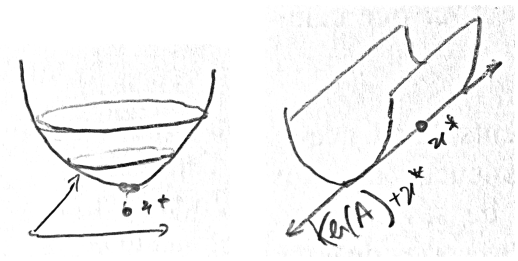


Figure 11.2: Coercivity condition for least squares.

is the least square quadratic risk function (see Fig. 11.1). Here $\langle u,\, v\rangle = \sum_{i=1}^p u_i v_i$ is the canonical inner product in $\mathbb{R}^p$ and $\|\cdot\|^2 = \langle \cdot,\, \cdot\rangle$.

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

$$f(x) = \sum_{i=1}^n \ell(-y_i \langle x,\, a_i\rangle) = L(-\operatorname{diag}(y)Ax) \tag{11.4}$$

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. 11.1, right). Here the separable loss function $L = \mathbb{R}^n \to \mathbb{R}$ is, for $z \in \mathbb{R}^n$, $L(z) = \sum_i \ell(z_i)$.

Coercivity. In general, there might be no solution to the optimization (12.1). This is of course the case if f is unbounded by bellow, for instance $f(x) = -x^2$ in which case the value of the minimum is $-\infty$. But this might also happens if f does not grow at infinity, for instance $f(x) = e^{-x}$, for which $\min f = 0$ but there is no minimizer.

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

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

which is bounded because of coercivity, and closed because f is continuous.

Example 3 (Least squares). For instance, for the quadratic loss function $f(x) = \frac{1}{2}\|Ax - y\|^2$, coercivity holds if and only if $\ker(A) = \{0\}$ (this corresponds to the overdetermined setting). Indeed, if $\ker(A) \neq \{0\}$ if $x^\star$ is a solution, then $x^\star + u$ is also solution for any $u \in \ker(A)$, so that the set of minimizer is unbounded. On contrary, if $\ker(A) = \{0\}$, we will show later that the set of minimizer is unique, see Fig. 11.2. If ℓ is strictly convex, the same conclusion holds in the case of classification.

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

$$\forall\, t \in [0, 1], \quad f((1-t)x + ty) \leqslant (1-t)f(x) + tf(y) \tag{11.5}$$

which means that the function is bellow its secant (and actually also above its tangent when this is well defined), see Fig. 11.3. If $x^\star$ is a local minimizer of a convex f, then $x^\star$ is a global minimizer, i.e. $x^\star \in \operatorname{argmin} f$.

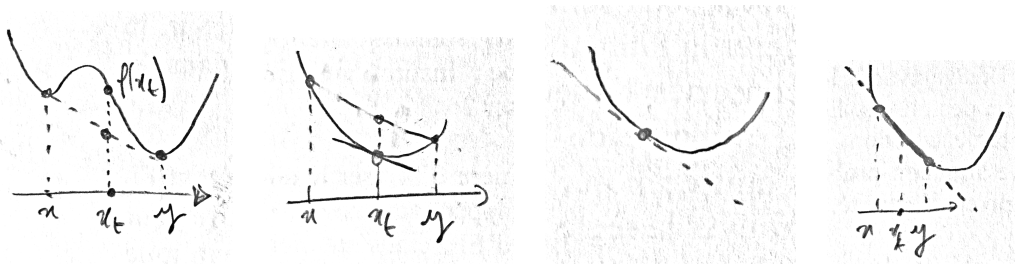


Figure 11.3: Convex vs. non-convex functions ; Strictly convex vs. non strictly convex functions.

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!). 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 (11.3) 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 (11.4)Â is itself convex.

In general, minimizers $x^\star$ might be non-unique **[ToDo: figure]**. When f is convex, one can strengthen the condition (11.5) and impose that the inequality is strict for $t \in]0,1[$ (see Fig. 11.3, right), i.e.

$$\forall\, t \in]0,1[, \quad f((1-t)x + ty) < (1-t)f(x) + tf(y). \tag{11.6}$$

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

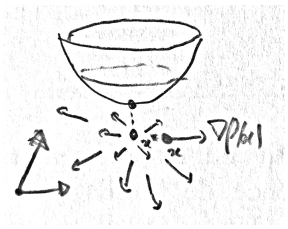
Example 4 (Least squares). For the quadratic loss function $f(x) = \frac{1}{2}\|Ax - y\|^2$, strict convexity is equivalent to $\ker(A) = \{0\}$ **[ToDo: show this]**.

11.2 Derivative and gradient

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

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

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



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

$$f(x + \varepsilon) = f(x) + \langle \varepsilon,\, \nabla f(x) \rangle + o(\|\varepsilon\|). \tag{11.7}$$

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

Also, $\nabla f(x)$ is the only vector such that the relation (11.7). 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 $\nabla f(x) = g$.

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

Figure 11.4: Function with local maxima/minima (left), saddle point (middle) and global minimum (right).

Proposition 35. *If $x^\star$ is a local minimum of the function f (i.e. that $f(x^\star) \leqslant f(x)$ for all x in some ball around $x^\star$) then*

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

Proof. One has for ε small enough and u fixed

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

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

Note that the converse is not true in general, since one might have $\nabla f(x) = 0$ but x is not a local mininimum. For instance $x = 0$ for $f(x) = -x^2$ (here x is a maximizer) or $f(x) = x^3$ (here x is neither a maximizer or a minimizer, it is a saddle point), see Fig. 11.4. 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 gradient-based algorithm will converge to point with $\nabla f(x^\star) = 0$ that are local minimizer. The following proposition shows that a much strong result holds if f is convex.

Proposition 36. *If f is differentiable and convex,*

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

Proof. **[ToDo: write me]** □

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

Example: Least squares. The most important gradient formula is the one of the square loss (11.3), 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. $A^\top = (A_{j,i})_{i=1,\ldots,n}^{j=1,\ldots,p}$, but the way it should be remember and used is that it obeys the following swapping rule of the inner product,

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

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

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

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

$$x^\star = (A^\top A)^{-1} A^\top y. \tag{11.8}$$

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

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

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

Example: Classification We can do a similar computation for the gradient of the classification loss (11.4). Assuming that L is differentiable, and using the Taylor expansion (11.7) 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 $+1$).

11.3 Gradient Descent

Steepest descent direction. The Taylor expansion (11.7) computes an affine approximation of the function f near x, since it can be written as

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

see Fig. 11.5. 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.

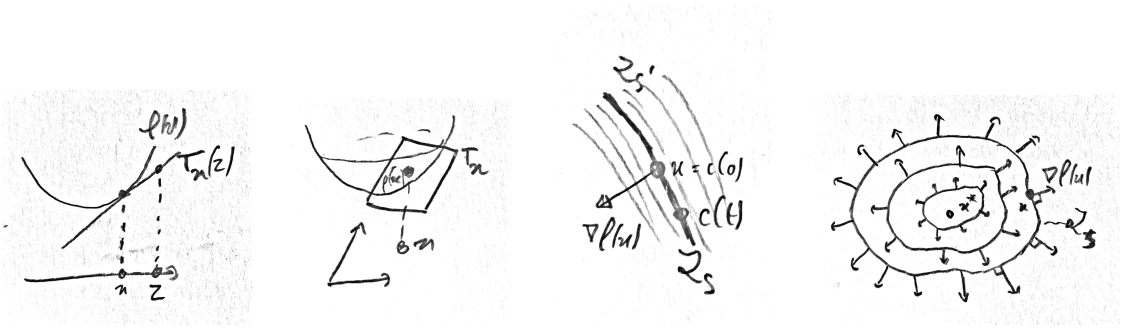


Figure 11.5: Left: First order Taylor expansion in 1-D and 2-D. Right: orthogonality of gradient and level sets and schematic of the proof.

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

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

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. 11.5 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 $h(t) \overset{\text{def.}}{=} f(c(t))$ 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. 11.5, 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,

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

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

$$x_{k+1} \overset{\text{def.}}{=} x_k - \tau_k \nabla f(x_k) \tag{11.9}$$

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. 11.6).

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

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

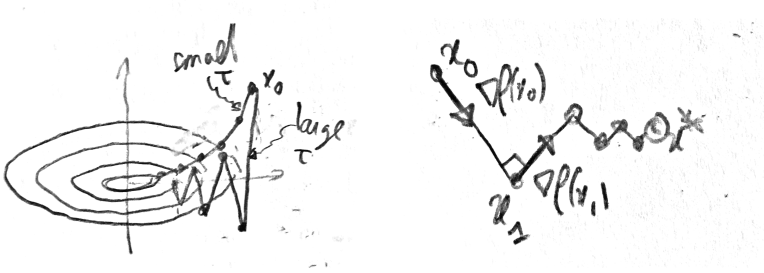


Figure 11.6: Influence of τ on the gradient descent (left) and optimal step size choice (right).

One note that at $\tau = \tau_k$, $\nabla f(x_k - \tau \nabla f(x_k)) = \nabla f(x_{k+1})$ by definition of x_{k+1} in (12.2). 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. 11.6).

11.4 Convergence Analysis for the Quadratic Case

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

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

We already saw that in (11.8)Â 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. **[ToDo: detail this]**

Proposition 37. *For $f(x) = \langle Cx, x\rangle - \langle b, x\rangle$ (C being symmetric semi-definite positive) with the eigenvalues of C upper-bounded by L and lower-bounded by $\mu > 0$, assuming there exists $(\tau_{\min}, \tau_{\max})$ such that*

$$0 < \tau_{\min} \leqslant \tau_\ell \leqslant \tilde{\tau}_{\max} < \frac{2}{L} \tag{11.10}$$

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

$$\|x_k - x^\star\| \leqslant \tilde{\rho}^\ell \|x_0 - x^\star\|. \tag{11.11}$$

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

$$\tau_\ell = \frac{2}{L+\mu} \quad \Longrightarrow \quad \tilde{\rho} \overset{\text{def.}}{=} \frac{L-\mu}{L+\mu} = 1 - \frac{2\varepsilon}{1+\varepsilon} \quad \text{where} \quad \varepsilon \overset{\text{def.}}{=} \mu/L. \tag{11.12}$$

Proof. One iterate of gradient descent reads

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

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

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

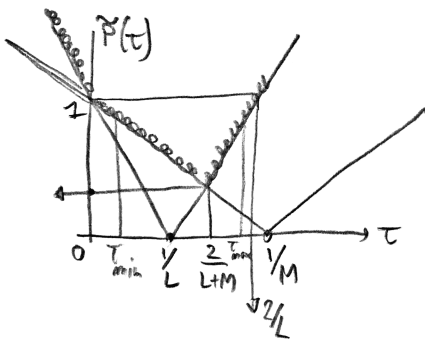


Figure 11.7: Contraction constant $h(\tau)$ for a quadratic function (right).

One thus has to study the contractance ratio of the linear map $\mathrm{Id}_p - \tau_\ell C$, i.e. its largest singular value, which reads

$$h(\tau) \overset{\text{def.}}{=} \|\mathrm{Id}_p - \tau C\|_2 = \sigma_{\max}(\mathrm{Id}_p - \tau C) = \max(|1 - \tau_\ell \sigma_{\max}(C)|, |1 - \tau \sigma_{\min}(C)|).$$

For a quadratic function, one has $\sigma_{\min}(C) = \mu, \sigma_{\max}(C) = L$. Figure 11.7, right, shows a display of $h(\tau)$. One has that for $0 < \tau < 2/L$, $h(\tau) < 1$. □

Note that when the condition number $\varepsilon \overset{\text{def.}}{=} \mu/L \ll 1$ is small (which is the typical setup for ill-posed problems), then the contraction constant appearing in (11.12) scales like

$$\tilde{\rho} \sim 1 - 2\varepsilon. \tag{11.13}$$

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

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

Proposition 38. *For $f(x) = \langle Cx, x\rangle - \langle b, x\rangle$, assuming the eigenvalue of C are bounded by L, then if $0 < \tau_\ell = \tau < 2/L$ is constant, then*

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

where

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

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

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

Now one has

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

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

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

where we have used the fact that **[ToDo: detail this]**

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

This thus implies

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

where we have denoted

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

One has

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

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

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

Indeed, one has

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

□

11.5 Convergence Analysis for the General Case

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

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

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

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

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

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

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

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

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

and thus

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

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

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

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

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

$$\forall\,(x,x'), \quad f(x') + \langle \nabla f(x), x'-x\rangle + \frac{\mu}{2}\|x-x'\|^2 \leqslant f(x) \leqslant f(x') + \langle \nabla f(x'), x'-x\rangle + \frac{L}{2}\|x-x'\|^2. \tag{11.14}$$

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

$$\forall\, x, \quad \mu \mathrm{Id}_p \preceq \partial^2 f(x) \preceq L\mathrm{Id}_p \tag{11.15}$$

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

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

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

Condition (11.15) 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 $\mathcal{Q}$ of the form (8.24), $\partial^2 f(x) = A$ is constant, so that $[\mu, L]$ can be chosen to be the range of the singular values of A.

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

$$0 < \tau_{\min} \leqslant \tau_\ell \leqslant \tau_{\max} < \frac{2}{L}, \tag{11.16}$$

then x_k converges to a solution $x^\star$ of (12.1) and there exists $C > 0$ such that

$$f(x_k) - f(x^\star) \leqslant \frac{C}{\ell + 1}. \tag{11.17}$$

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 (11.17) 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 (11.14), 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

$$f(x_{k+1}) \leqslant f(x_k) - \frac{1}{L}\|\nabla f(x_k)\|^2 + \frac{1}{2L}\|\nabla f(x_k)\|^2 \leqslant f(x_k) - \frac{1}{2L}\|\nabla f(x_k)\|^2 \tag{11.18}$$

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 (11.18) shows

$$\begin{aligned} f(x_{k+1}) &\leqslant f(x^\star) - \langle \nabla f(x_k),\, x^\star - x_k \rangle - \frac{1}{2L}\|\nabla f(x_k)\|^2 && (11.19)\\ &= f(x^\star) + \frac{L}{2}\left(\|x_k - x^\star\|^2 - \|x_k - x^\star - \frac{1}{L}\nabla f(x_k)\|^2 \right) && (11.20)\\ &= f(x^\star) + \frac{L}{2}\left(\|x_k - x^\star\|^2 - \|x^\star - x_{k+1}\|^2 \right). && (11.21) \end{aligned}$$

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

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

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

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

which gives (11.17) for $C \overset{\text{def.}}{=} L\|x_0 - x^\star\|^2/2$.

If we now assume f is μ-strongly convex, then, using $\nabla f(x^\star)=0$, one has $\frac{\mu}{2}\|x^\star-x\|^2 \leqslant f(x)-f(x^\star)$ for all x. Re-manipulating (11.21) 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

$$\|x_{k+1}-x^\star\| \leqslant \sqrt{\frac{L}{L+\mu}}\|x_{k+1}-x^\star\|, \tag{11.22}$$

which is the desired result. □

Note that in the low conditioning setting $\varepsilon \ll 1$, one retrieve a dependency of the rate (11.22) similar to the one of quadratic functions (11.13), indeed

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

11.6 Stochastic Optimization

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

11.6.1 Minimizing Sums and Expectation

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

$$\min_{x\in\mathbb{R}^p} f(x) \stackrel{\text{def.}}{=} \frac{1}{n}\sum_{i=1}^n f_i(x) \tag{11.23}$$

or even expectations of the form

$$\min_{x\in\mathbb{R}^p} f(x) \stackrel{\text{def.}}{=} \mathbb{E}_{\mathbf{z}\sim\pi}(f(x,\mathbf{z})) = \int_{\mathcal{Z}} f(x,z)\mathrm{d}\pi(z). \tag{11.24}$$

Problem (11.23) can be seen as a special case of (11.24), 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 (11.23) as a discretized "empirical" version of (11.24) when drawing $(z_i)_i$ i.i.d. according to $\mathbf{z}$ and defining $f_i(x)=f(x,z_i)$. In this setup, (11.23) converges to (11.24) as $n\to+\infty$.

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

$$f_i(x)=\ell(\langle a_i,\, x\rangle, y_i) \quad \text{and} \quad f(x,z)=\ell(\langle a,\, x\rangle, y) \tag{11.25}$$

for $z=(a,y)\in\mathcal{Z}=(\mathcal{A}=\mathbb{R}^p)\times\mathcal{Y}$ (typically $\mathcal{Y}=\mathbb{R}$ or $\mathcal{Y}=\{-1,2\}$ for regression and classification), where ℓ is some loss function. We illustrate bellow the methods on binary logistic classification, where

$$L(s,y) \stackrel{\text{def.}}{=} \log(1+\exp(-sy)), \tag{11.26}$$

see Section 15.4.2 for details. But this extends to arbitrary parametric models, and in particular deep neural networks as detailed in Section 16.2.

While some algorithms (in particular batch gradient descent) are specific to finite sums (11.23), the stochastic methods we detail next work verbatim (with the same convergence guarantees) in the expectation case (11.24). 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.

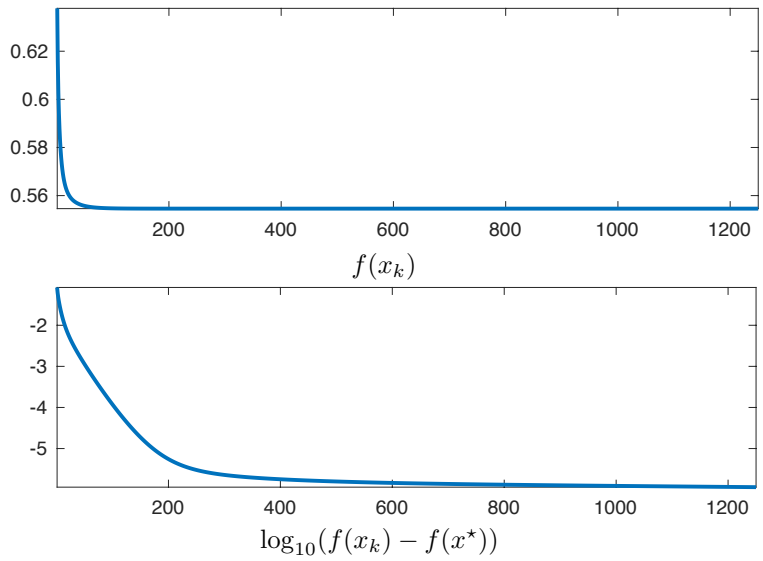


Figure 11.8: Evolution of the error of the BGD for logistic classification.

The general idea underlying stochastic optimization methods is *not* to have faster algorithms with respect to traditional optimization schemes such as those detailed in Chapter 12. In almost all cases, if n is not too large so that one afford the price of doing a few non-stochastic iterations, then deterministic methods are faster. But if n is so large that one cannot do even a single deterministic iteration, then stochastic methods allow one to have a fine grained scheme by breaking the cost of determinstic iterations in smaller chunks. Another advantage is that they are quite easy to parallelize.

11.6.2 Batch Gradient Descent (BGD)

The usual deterministic (batch) gradient descent (BGD) is studied in details in Section 12.1. Its iterations read

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

and the step size should be chosen as $0 < \tau_{\min} < \tau_k < \tau_{\max} \stackrel{\text{def.}}{=} 2/L$ where L is the Lipschitz constant of the gradient ∇f. In particular, in this determinstic 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

$$\nabla f(x) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x) \tag{11.27}$$

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

In the ERM setting (11.25), the gradient reads **[ToDo: detail here]**

$$\nabla f_i(x) = \ell'(\langle a_i, x \rangle, y_i) a_i, \tag{11.28}$$

where $\ell(y, y') \in \mathbb{R}$ is the derivative with respect to the first variable, i.e. the gradient of the map $y \in \mathbb{R} \mapsto L(y, y') \in \mathbb{R}$. For the logistic loss, it is simply

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

11.6.3 Stochastic Gradient Descent (SGD)

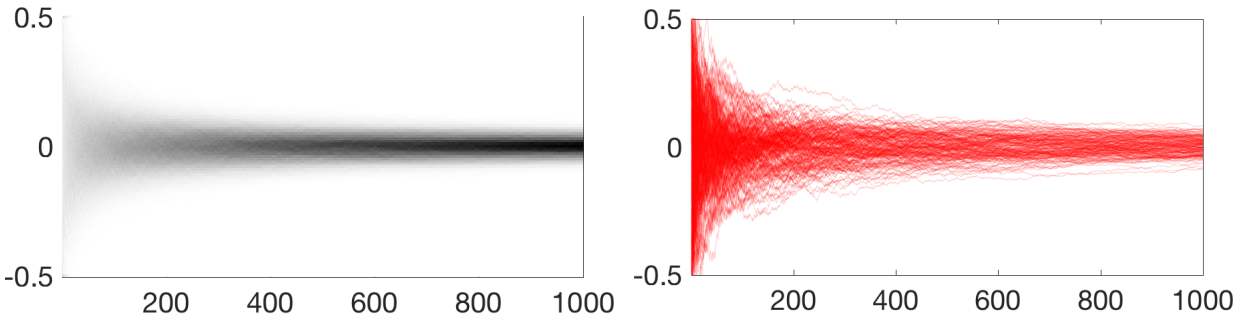


Figure 11.11: Display of a large number of trajectories $k \mapsto x_k \in \mathbb{R}$ generated by several runs of SGD. On the top row, each curve is a trajectory, and the bottom row displays the corresponding density.

For very large n, computing the full gradient ∇f as in (11.27) 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

$$\mathbb{E}_{\mathbf{i}} \nabla f_{\mathbf{i}}(x) = \nabla f(x) \tag{11.29}$$

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

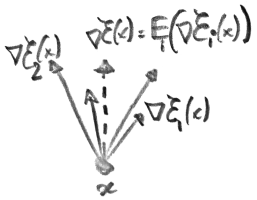


Figure 11.9: Unbiased gradient estimate

Starting from some $x^{(0)}$,the iterations of stochastic gradient descent (SGD) read

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

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

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

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

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

$$\tau_k \overset{\text{def.}}{=} \frac{\tau_0}{1 + k/k_0} \tag{11.30}$$

Figure 11.10: Schematic view of SGD iterates

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

Figure 11.11 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 expectration with a $1/\sqrt{k}$ rate on the objective.

Theorem 15. *We assume f is μ-strongly convex as defined in ($\mathcal{S}_\mu$) (i.e. $\mathrm{Id}_{N\times N} \preceq \partial^2 f(x)$ 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*

$$\mathbb{E}(\|x_k - x^\star\|^2) \leqslant \frac{R}{k+1} \quad \textit{where} \quad R = \max(\|x^{(0)} - x^\star\|, C^2/\mu^2), \tag{11.31}$$

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

$$\langle \nabla f(x_k) - \nabla f(x^\star),\, x_k - x^\star \rangle = \langle \nabla f(x_k),\, x_k - x^\star \rangle \geqslant \mu \|x_k - x^\star\|^2. \tag{11.32}$$

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 (11.29) that the gradient is unbiased. Taking now the full expectation with respect to all the other previous iterates, and using (11.32) one obtains

$$\mathbb{E}(\|x_{k+1} - x^\star\|^2) \leqslant \mathbb{E}(\|x_k - x^\star\|^2) - 2\mu\tau_k \mathbb{E}(\|x_k - x^\star\|^2) + \tau_k^2 C^2 = (1 - 2\mu\tau_k)\mathbb{E}(\|x_k - x^\star\|^2) + \tau_k^2 C^2. \tag{11.33}$$

We show by recursion that the bound (11.31) 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 (11.33) 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 14.

Figure 11.12 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.

11.6.4 Stochastic Gradient Descent with Averaging (SGA)

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

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

and output as estimated weight vector the Cesaro average

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

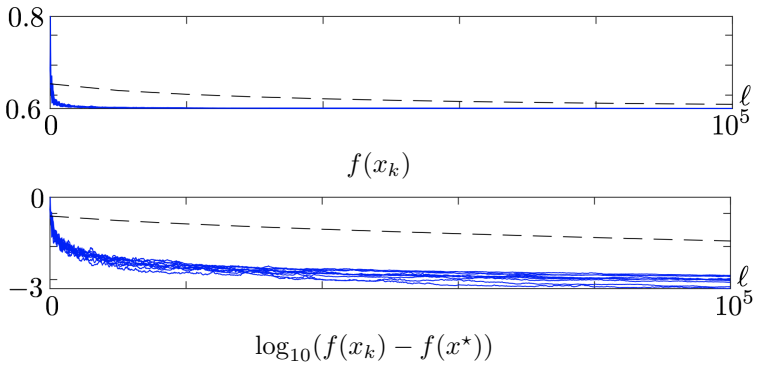


Figure 11.12: Evolution of the error of the SGD for logistic classification (dashed line shows BGD).

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

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

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

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

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

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

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

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

11.6.5 Stochastic Averaged Gradient Descent (SAG)

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

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

The algorithm reads

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

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.

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

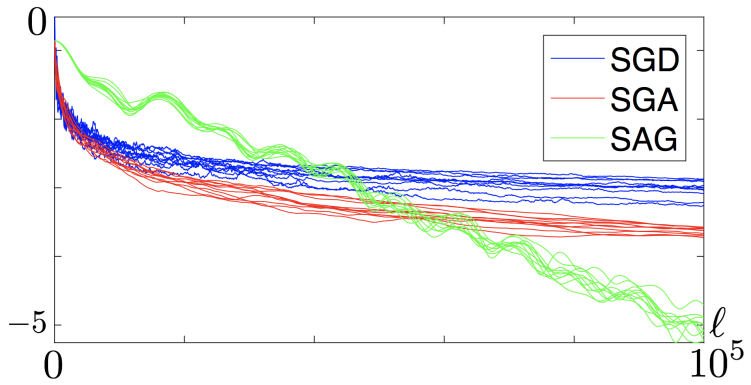


Figure 11.13: Evolution of $\log_{10}(f(x_k) - f(x^\star))$ for SGD, SGA and SAG.

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 (11.24)).

Figure 16.1 shows a comparison of SGD, SGA and SAG.

11.7 Automatic Differentiation

The main computational bottleneck of these gradient descent methods (batch or stochastic) is the evaluation of the elementary gradients $\nabla \mathcal{E}_i$. The gradient formula (11.28) shows that it requires to remap the gradient of the loss $\nabla L(f(x_i, \beta), y_i)$ through the adjoint of the Jacobian $\partial f(x_i, \beta)$. The general idea is that for complicated model this computation should be broken in simpler sub-computation, which ultimately should corresponds to elementary operators (binary operators such as + or * and unary operators such as exp, log, etc.) for which the differential are trivial to compute.

11.7.1 Reverse Differentiation on a Feedforward Graph

To give a concrete examples which is actually found in many practical situation (and in particular for simple deep architectures, as detailed in Section 16.2.1), if the functional to be differentiated has the form

$$\mathcal{E}(\beta) = \mathcal{L} \circ \mathcal{F}_{L-1} \circ \mathcal{F}_{L-2} \circ \ldots \circ \mathcal{F}_0(\beta) \tag{11.34}$$

(often called a "feedforward" model) where $\mathcal{F}_\ell : \mathbb{R}^{n_\ell} \to \mathbb{R}^{n_{\ell+1}}$ and $\mathcal{L} : \mathbb{R}^{n_L} \to \mathbb{R}$, then one can compute the gradient $\nabla \mathcal{E}(\beta)$ in two steps:

– A forward pass, where one evaluates the function itself and keep track of all the intermediate computations, i.e., initializing $\beta_0 = \beta$, one computes

$$\beta_{\ell+1} \stackrel{\text{def.}}{=} \mathcal{F}_\ell(\beta_\ell) \quad \text{and} \quad \mathcal{E}(\beta) = \mathcal{L}(\beta_L).$$

– A backward pass, where one use the chain rule together with the Jacobian transposition

$$\nabla \mathcal{E}(\beta) = [\partial \mathcal{F}_0(\beta_0)]^* \circ \ldots \circ [\partial \mathcal{F}_{L-1}(\beta_{L-1})]^* (\nabla \mathcal{L}(\beta_L)) \tag{11.35}$$

to define the following backward recursion, initialized by $h_L = \nabla \mathcal{L}(\beta_L)$, and then

$$h_{\ell-1} \stackrel{\text{def.}}{=} [\partial \mathcal{F}_{\ell-1}(\beta_{\ell-1})]^*(h_\ell) \quad \text{and} \quad \nabla \mathcal{E}(\beta) = h_0. \tag{11.36}$$

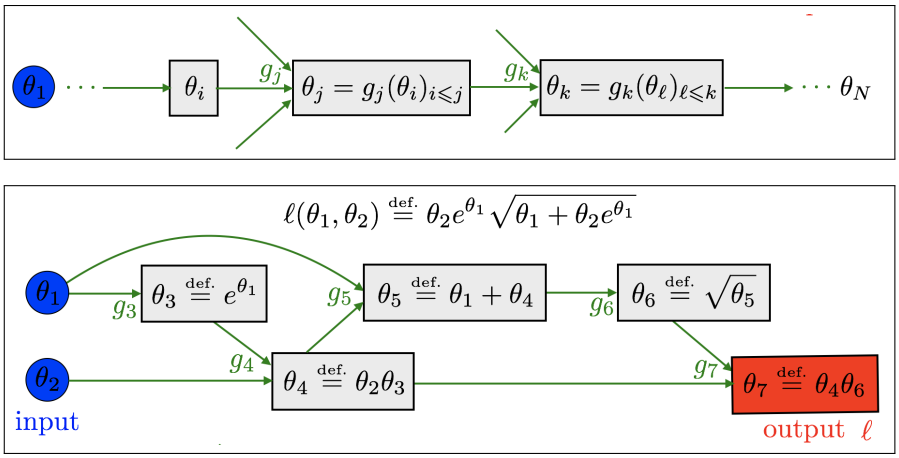


Figure 11.14: Top: elementary component of the DAG computational graph. Bottom: example of DAG computational graph.

The main issue here is when these adjoint Jacobian $[\partial \mathcal{F}_{\ell-1}(\beta_{\ell-1})]^*$ are difficult to apply (and it is out of question in most cases to actually store them on a computer, since it would occurs an enormous storage requirement and typical quadratic time scaling for the algorithms). We now detail a finer grained analysis which enable to tackle building blocks of arbitrary complexity.

The computation (11.35) should be compared with the "forward" accumulation

$$\nabla \mathcal{E}(\beta) = \partial \mathcal{L}(\beta_L) \circ \partial \mathcal{F}_{L-1}(\beta_{L-1}) \ldots \circ \partial \mathcal{F}_0(\beta_0).$$

Computing these matrix product would be extremely costly, although it would require no memory overhead because the computation would be caried over in parallel to the evaluation of the function.

11.7.2 Reverse Differentiation on a Generic Computational Graph

One can generalize the idea above to differentiate automatically any function which can be implemented on a computer. What is even more surprising is that the computational cost is the same as the one of evaluating the function itself. This fundamental computational fact (that gradient evaluation and function evaluation have the same computational cost) is not so well known, but is of paramount practical interest when it comes to differentiating complicated recursive functions. We will apply it in a very simple setup for deep-architectures, but it can be applied to much more involved computational architectures.

Note that this results only applies for function which output scalar values. For functions which output vector values, one can of course re-use this idea for each output, but this is in general vastly sub-optimal, because it ignore the redundancy between the computation of each output. The determination of optimal strategy in this case is known to be NP-hard. This include for instance the computation of the Hessian of a scalar valued function (since it corresponds to the differentiation of the gradient, which is itself a vector-valued function). Fortunately, for machine learning application, one is often interested in differentiating only empirical losses functions, which as scalar valued.

Forward pass as a DAG traversal. The crux of this idea is that the computational flow of any computable function ℓ can be represented as a directed acyclic graph. We denote $(\theta_i)_{i=1}^R$ the set of all scalar variables (input, output and intermediary) manipulated by the computational program. Without loss of generality, we impose that the first variable $(\theta_1, \ldots, \theta_M)$ are the M input variables, while the last θ_R is the output variable. The function to be computed is thus of the form

$$\theta_R = \ell(\theta_1, \ldots, \theta_M)$$

where $\ell : \mathbb{R}^M \to \mathbb{R}$ is broken in $R - M + 1$ intermediate steps corresponding to all the remaining variable $(\theta_i)_{M<i<R}$. The successive execution of the program defines an ordering of all the intermediate variables, so that, after initializing the input variables $(\theta_1, \ldots, \theta_M)$, the forward pass computes the value of θ_r for $r = M+1, \ldots, R$ as

$$\theta_r = g_r(\theta_{\pi(r)})$$

for some scalar valued function $g_r : \mathbb{R}^{|\pi(r)|} \to \mathbb{R}$, where $\pi(r) \subset \{1, \ldots, r-1\}$ is the set of "parent" node of r in a directed acyclic graph (DAG). Figure 11.14 shows an example of such a computational DAG.

From a symbolic computation point of view, variables θ_j (for $j > M$) in the graph can be interpreted either as variables (i.e. which can be assigned scalar values) and functions depending on input variables θ_m for $m \leqslant M$. The beauty of this DAG representation is that one can also view θ_j as depending on any other intermediate variable θ_i as long as $i < j$.

Direct mode auto-diff. The goal is to compute the gradient vector, which reads

$$\nabla \ell = \left(\frac{\partial \theta_R}{\partial \theta_m} \right)_{m=1}^{M}.$$

The naive way to comput this gradient vector would thus be to compute for each of the M input variable θ_m the differential $\frac{\partial \theta_j}{\partial \theta_m}$ of all function θ_j with respect to θ_m. Without loss of generality, we consider $m = 1$. This can be achieved by using the standard chain rule

$$\frac{\partial \theta_j}{\partial \theta_1} = \sum_{i \in \pi(j)} \frac{\partial \theta_j}{\partial \theta_i} \frac{\partial \theta_i}{\partial \theta_1}. \tag{11.37}$$

Here the multipliers involved are actually differential of the elementary functions

$$\frac{\partial \theta_j}{\partial \theta_i} = \partial_i g_j$$

Note that this writing is abusive, since $\frac{\partial \theta_j}{\partial \theta_i}$ really means that in practice such a differential is evaluated assuming all the variable $(\theta_r)_{r<j}$ on which the function θ_j depends are defined to their respective value (which have been computed by the forward pass, which here can be run in parallel to the forward DAG traversal).

By traversing forward the DAG, iterating this formula compute all the derivative, and in particular $(\nabla \ell)_1 = \frac{\partial \theta_R}{\partial \theta_1}$. This approach, while being the most natural, is however vastly sub-optimal because its complexity is M times the one of the evaluation of the function.

Reverse mode auto-diff. Instead of computing the quantities $(\frac{\partial \theta_j}{\partial \theta_1})_j$, a radically different approach consists in rather computing the quantities $\frac{\partial \theta_L}{\partial \theta_j} = (\nabla \ell)_j$. In place of the "forward" chain rule, one needs to the backward one

$$\frac{\partial \theta_R}{\partial \theta_j} = \sum_{j \in \pi(k)} \frac{\partial \theta_R}{\partial \theta_k} \frac{\partial \theta_k}{\partial \theta_j}. \tag{11.38}$$

Note that here the summation is done over k which are "child" of j in the DAG. Here the multiplier appearing in the formula are differential of the elementary function since $\frac{\partial \theta_k}{\partial \theta_j} = \partial_j g_k$. The main interest of this reverse recursion (11.38) with respect to the direct one (11.38) is that it only needs to be run once, so that the overall complexity is the same as the one of the forward pass to compute the function itself.

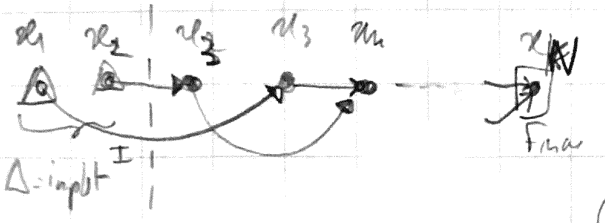
Figure 11.15 recaps the two passes of the reverse mode automatic differentiation method.

The main bottleneck of this backward automatic differentiation technic is the memory consumption. Indeed, since all intermediate results need to be computed and stored explicitly before applying the backward pass, memory grows proportionally to execution time. This can be unacceptable for very large machine learning model. Fortunately, it is possible to trade time vs. memory and only keep track of a fraction of

intermediate results, and retrieve the missing result locally by small forward passes. Doing this approach recursively allows to only have a logarithmic overhead in term of both time and memory, showing the vast superiority of automatic differentiation method with respect to any other alternative for differentiation. We could not insist more on the crucial importance and impact of this class of technics on modern data science.

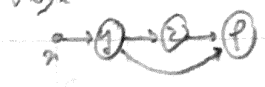
Automatic Differentiation

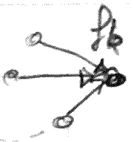
Math formula ⟷ several implementation ⟷ best (very hard)
Impl.



Example: $f(x) = \log x + \sqrt{\log x}$

$y = \log x$
$z = \sqrt{y}$
$f = y + z$





∂x_1 then ∂x_2

For $k = I+1 \dots N$

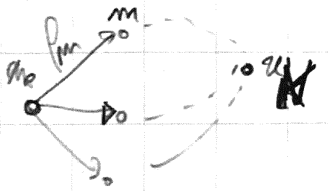
$x_k = f_k(x_1, \dots, x_{k-1})$

and

For $k = I+1 \dots N$

$$\frac{\partial x_k}{\partial x_1} = \sum_{\substack{l \in Father(k) \\ l < k}} \frac{\partial x_k}{\partial x_l} \times \frac{\partial x_l}{\partial x_1} = \sum_{l \in Father(k)} \frac{\partial f_k(x_1, \dots, x_{k-1})}{\partial x_l} \times \frac{\partial x_l}{\partial x_1}$$

Formula.



For $k = N-1 \dots I$

$$\frac{\partial x_N}{\partial x_k} = \sum_{\substack{m \in Son(k) \\ m > k}} \frac{\partial x_N}{\partial x_m} \times \left(\frac{\partial x_m}{\partial x_k} \right) = \sum_{m \in Son(k)} \frac{\partial x_N}{\partial x_m} \times \frac{\partial f_m}{\partial x_k}(x_1, \dots, x_{m-1})$$

Need to run first the algo once

For gradient, $x \in \mathbb{R}^n$: $f(x_1, \dots, x_I)$.

$$\nabla_{x_k} f = \left[\frac{\partial x_N}{\partial x_k} \right]^T$$

$$\nabla_{x_k} f = \sum_{m \in Son(k)} \left[\frac{\partial f}{\partial x_k}(x_1, \dots, x_{m-1}) \right]^T \times \nabla_{x_m} f$$

→ Simple example $\mathbb{R} \to \mathbb{R}$.
→ feed Forward $\mathbb{R}^n \to \mathbb{R}$ (matrix mult)
→ MLP example
→ Res Net

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

$x \to y = \log x \to z = \sqrt{y} \to f = y + z$

FWD

$\frac{\partial x}{\partial x} = 1$

$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial x} \cdot \frac{\partial x}{\partial x} = \frac{1}{x} \cdot 1$

$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x} = \frac{1}{2\sqrt{y}} \cdot \frac{\partial y}{\partial x}$

$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial x} + \frac{\partial f}{\partial y} \cdot \frac{\partial y}{\partial x} = 1 \cdot \frac{\partial z}{\partial x} + 1 \cdot \frac{\partial y}{\partial x}$

BWD

$\frac{\partial f}{\partial f} = 1$

$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f} \cdot \frac{\partial f}{\partial z} = 1 \cdot 1$

$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial y} + \frac{\partial f}{\partial f} \cdot \frac{\partial f}{\partial y} = \frac{\partial f}{\partial z} \cdot \frac{1}{2\sqrt{y}} + 1 \cdot 1$

$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} \cdot \frac{\partial y}{\partial x} = \frac{\partial f}{\partial y} \cdot \frac{1}{x}$

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

$x \to y = x^2 \to z = \sqrt{y+1} \to n = \log(x+z) \to t = n^3 \to u = \frac{t}{z} \to f = s - u$

$n \to s = \frac{n}{y}$

$\frac{\partial x}{\partial x} = 1$ (x)

$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial x} \cdot \frac{\partial x}{\partial x} = 2x \cdot \frac{\partial x}{\partial x}$ (x → y)

$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x} = \frac{1}{2\sqrt{y+1}} \cdot \frac{\partial y}{\partial x}$ (y → z)

$\frac{\partial n}{\partial x} = \frac{\partial n}{\partial x} \frac{\partial x}{\partial x} + \frac{\partial n}{\partial z} \frac{\partial z}{\partial x} = \frac{1}{x+z} \frac{\partial x}{\partial x} + \frac{1}{x+z} \frac{\partial z}{\partial x}$ (x, z → n)

$\frac{\partial s}{\partial x} = \frac{\partial s}{\partial n} \frac{\partial n}{\partial x} + \frac{\partial s}{\partial y} \frac{\partial y}{\partial x} = \frac{1}{y} \frac{\partial n}{\partial x} - \frac{n}{y^2} \frac{\partial y}{\partial x}$ (n, y → s)

$\frac{\partial t}{\partial x} = \frac{\partial t}{\partial n} \cdot \frac{\partial n}{\partial x} = 3n^2 \frac{\partial n}{\partial x}$ (n → t)

$\frac{\partial u}{\partial x} = \frac{\partial u}{\partial t} \frac{\partial t}{\partial x} + \frac{\partial u}{\partial z} \frac{\partial z}{\partial x} = \frac{1}{z} \frac{\partial t}{\partial x} - \frac{t}{z^2} \frac{\partial z}{\partial x}$ (t, z → u)

$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial s} \frac{\partial s}{\partial x} + \frac{\partial f}{\partial u} \cdot \frac{\partial u}{\partial x} = 1 \frac{\partial s}{\partial x} - 1 \frac{\partial u}{\partial x}$ (s, u → f)

FWD mode

$\frac{\partial f}{\partial f} = 1$

$\frac{\partial f}{\partial u} = \frac{\partial f}{\partial f} \cdot \frac{\partial f}{\partial u} = \frac{\partial f}{\partial f} \cdot (-1)$ (u → f)

$\frac{\partial f}{\partial t} = \frac{\partial f}{\partial u} \cdot \frac{\partial u}{\partial t} = \frac{\partial f}{\partial u} \cdot \frac{1}{z}$ (t → u)

$\frac{\partial f}{\partial s} = \frac{\partial f}{\partial f} \cdot \frac{\partial f}{\partial s} = \frac{\partial f}{\partial f} (+1)$ (s → f)

$\frac{\partial f}{\partial n} = \frac{\partial f}{\partial t} \cdot \frac{\partial t}{\partial n} + \frac{\partial f}{\partial s} \cdot \frac{\partial s}{\partial n} = \frac{\partial f}{\partial t} 3n^2 + \frac{\partial f}{\partial s} \frac{1}{y}$ (n → t, s)

$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial u} \cdot \frac{\partial u}{\partial z} + \frac{\partial f}{\partial n} \frac{\partial n}{\partial z} = \frac{\partial f}{\partial u} \frac{-t}{z^2} + \frac{\partial f}{\partial n} \frac{1}{x+z}$ (z → u, n)

$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial z} \frac{\partial z}{\partial y} + \frac{\partial f}{\partial s} \frac{\partial s}{\partial y} = \frac{\partial f}{\partial z} \frac{1}{2\sqrt{y+1}} + \frac{\partial f}{\partial s} \cdot \frac{-x}{y^2}$ (y → z, s)

$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} \cdot \frac{\partial y}{\partial x} + \frac{\partial f}{\partial n} \cdot \frac{\partial n}{\partial x} = \frac{\partial f}{\partial y} 2x + \frac{\partial f}{\partial n} \cdot \frac{1}{x+z}$ (x → y, n)

BWD mode

Feed Fawad : $x_1 \xrightarrow{f_1} x_2 \xrightarrow{f_2} x_3 \to \cdots \xrightarrow{f_{N-1}} x_N = f(x_1)$ ($x_1 \in \mathbb{R}^{n_1}$, $x_2 \in \mathbb{R}^{n_2}$, $x_N \in \mathbb{R}^{n_N}$)

ie. $f = f_{N-1} \circ f_{N-2} \circ \cdots \circ f_2 \circ f_1$ $\qquad x_{k+1} = f_k(x_k)$

Jacobian : $\partial f(x_1) = \underbrace{\partial f_{N-1}(x_{N-1})}_{A_{N-1}} \times \underbrace{\partial f_{N-2}(x_{N-2})}_{A_{N-2}} \times \cdots \times \underbrace{\partial f_2(x_2)}_{A_2} \cdot \underbrace{\partial f_1(x_1)}_{A_1}$

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

$A_{N-1} \in \mathbb{R}^{n_N \times n_{N-1}}$, $A_{N-2} \in \mathbb{R}^{n_{N-1} \times n_{N-2}}$, $A_2 \in \mathbb{R}^{n_3 \times n_2}$, $A_1 \in \mathbb{R}^{n_2 \times n_1}$

FWD mode:

$A \times B \rightsquigarrow$ cplxity mpq ($A \in \mathbb{R}^{m \times p}$, $B \in \mathbb{R}^{p \times q}$)

$A_{N-1} \times \left(A_{N-2} \times \left(\cdots \left(A_3 \times \left(A_2 \times A_1 \right) \right) \cdots \right) \right)$

$n_N \times n_{N-1} \times n_1$; $n_{N-1} \times n_{N-2} \times n_1$; $n_4 n_3 n_1$; $n_3 n_2 n_1$; $n_3 \times n_1$

BWD mode

$\left(\left(\left(A_{N-1} \times A_{N-2} \right) \times A_{N-3} \right) \times \cdots \times A_3 \right) \times A_2 \times A_1 \Big)$

$n_N \times n_{N-1} \times n_{N-2}$; $n_N n_{N-2} n_{N-3}$; $n_N n_4 n_3$; $n_N n_3 n_2$; $n_N n_2 n_1$

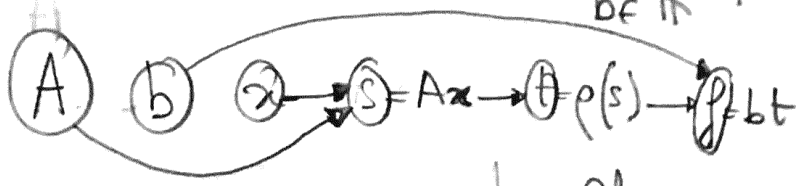
Complexity : $\underline{\text{FWD}} = n_1 \sum_{k=2}^{N-1} n_k n_{k+1}$ $\qquad \underline{\text{BWD}} = n_N \sum_{k=1}^{N-2} n_k n_{k+1}$

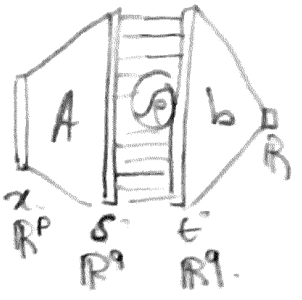
if $n_1 \ll n_N$ (i.e. $n_1 \ll n_N$) $\to$ FWD $\ll$ BWD (i.e. FWD $\ll$ BWD)

if $n_1 = 1$: FWD optimal (for any graph)

$n_N = 1$ (grad. compt$^{\text{c}}$) : BWD optimal (for any graph)

MLP: $f(x, A, b) = b\,\rho(Ax)$. $A \in \mathbb{R}^{q \cdot p}$, $b \in \mathbb{R}^{1 \cdot q}$





$\frac{\partial x}{\partial A} = 0 \quad \frac{\partial b}{\partial A} = 0 \quad \frac{\partial A}{\partial A} = Id_{(qp)(qp)}$

$\frac{\partial s}{\partial A} = \frac{\partial s}{\partial x}\frac{\partial x}{\partial A} + \frac{\partial s}{\partial A}\frac{\partial A}{\partial A} = R_x \cdot \frac{\partial A}{\partial A}$ ($\frac{\partial x}{\partial A} = 0$)

$B \in \mathbb{R}^{q \cdot p} \to Bx \in \mathbb{R}^q$

$\frac{\partial t}{\partial A} = \frac{\partial t}{\partial s}\frac{\partial s}{\partial A} = diag(\rho'(s))\frac{\partial s}{\partial A}$ (s → t)

$\frac{\partial f}{\partial A} = \frac{\partial f}{\partial b}\frac{\partial b}{\partial a} + \frac{\partial f}{\partial t}\cdot\frac{\partial t}{\partial A} = I_b \frac{\partial t}{\partial A}$ ($\frac{\partial b}{\partial a} = 0$) (b → f, t → f)

$t \in \mathbb{R}^q \to bt \in \mathbb{R}$

FWD (this is crazy ...)

$\frac{\partial f}{\partial f} = 1$

$\frac{\partial f}{\partial t} = \frac{\partial f}{\partial f}\cdot\frac{\partial f}{\partial t} = \frac{\partial f}{\partial f} b \xrightarrow{T} \nabla_t f = b^T$ (t → f)

$\frac{\partial f}{\partial s} = \frac{\partial f}{\partial t}\frac{\partial t}{\partial s} = \frac{\partial f}{\partial t}\cdot diag(\rho'(s)) \xrightarrow{T} \nabla_s f = diag(\rho'(s))\nabla_t f$ (s → t)

$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial s}\cdot\frac{\partial s}{\partial x} = \frac{\partial f}{\partial s}\cdot A \to \nabla_x f = A^T\frac{\partial f}{\partial s}$ (x → s)

$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial f}\cdot\frac{\partial f}{\partial b} = 1 \cdot t \rightsquigarrow \nabla_b f = t^T$ (b → f)

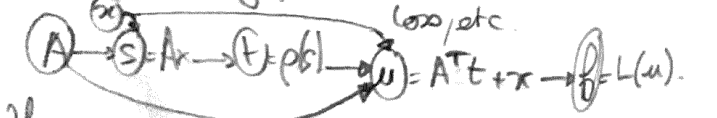
BWD

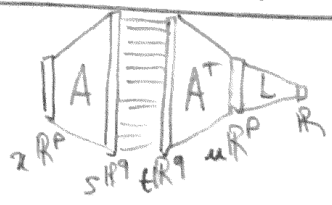
Directly: $f(x, A+\varepsilon, b) = b\rho(Ax + \varepsilon x) = b\rho(Ax) + b(\rho'(Ax)\odot\varepsilon x) = f(x,A,b) + \langle b^T, \rho'(Ax)\odot\varepsilon x\rangle$

$= f(x,A,b) + \langle \varepsilon, [\rho'(Ax)\odot b^T]x^T\rangle \Rightarrow \nabla_A f(x,A,b) = [\rho'(Ax)\odot b^T]x^T$

$f(x, A, b+\varepsilon) = (b+\varepsilon)\rho(Ax) = f(x,A,b) + \langle \varepsilon, \rho(Ax)^T\rangle \Rightarrow \nabla_b f(x,A,b) = \rho(Ax)^T$

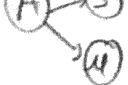
ResNet: $f(x, A) = L(A^T\rho(Ax) + x)$ (L ∞, etc)





$\frac{\partial f}{\partial f} = 1$; $\nabla_u L = \underline{init}$; $\nabla_t = A\nabla_u$; $\nabla_s = diag(\rho'(s))\nabla_t$

$\nabla_A = \nabla_s x^T + (\nabla_u)^T t$



forward

```
function ℓ(θ_1, ..., θ_M)
  for r = M + 1, ..., R
    θ_r = g_r(θ_Parents(r))
  return θ_R
```

backward

```
function ∇ℓ(θ_1, ..., θ_M)
  ∇_R ℓ = 1
  for r = R − 1, ..., 1
    ∇_r ℓ = Σ_{s∈Child(r)} ∂_r g_s(θ) ∇_s ℓ
  return (∇_1 ℓ, ..., ∇_M ℓ)
```

Figure 11.15: Recap of the two step of the automatic differentiation procedure.

Bibliography

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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