Machine Learning - Optimization

E. Scornet

Fall 2018

Outline

Outline

Logistic regression

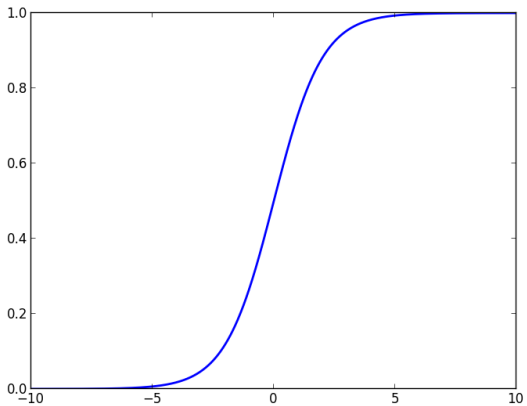
- By far the most widely used classification algorithm
- We want to explain the label y based on x, we want to "regress" y on x
- Models the distribution of $Y|X$

For $y \in \{-1, 1\}$, we consider the model

$$\mathbb{P}(Y = 1|X = x) = \sigma(\langle w, x \rangle + b)$$

where $w \in \mathbb{R}^d$ is a vector of model **weights** and $b \in \mathbb{R}$ is the **intercept**, and where σ is the **sigmoid** function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Logistic regression

- The sigmoid choice really is a **choice**. It is a **modelling choice**.
- It's a way to map $\mathbb{R} \to [0,1]$ (we want to model a probability)
- We could also consider

$$\mathbb{P}(Y = 1|X = x) = F(\langle w, x \rangle + b),$$

for any distribution function F. Another popular choice is the Gaussian distribution

$$F(z) = \mathbb{P}(N(0,1) \leqslant z),$$

which leads to another loss called **probit**

Logistic regression

- However, the sigmoid choice has the following nice interpretation: an easy computation leads to

$$\log\left(\frac{\mathbb{P}(Y=1|X=x)}{\mathbb{P}(Y=-1|X=x)}\right)=\langle w,x\rangle+b$$

  This quantity is called the **log-odd ratio**

- Note that

$$\mathbb{P}(Y=1|X=x)\geqslant\mathbb{P}(Y=-1|X=x)$$

  iff

$$\langle w,x\rangle+b\geqslant 0.$$

- This is a **linear classification** rule
- Linear with respect to the considered features x
- But, **you** choose the features: **features engineering**.

Logistic regression

Estimation of w and b

- We have a model for $Y|X$
- Data (x_i, y_i) is assumed i.i.d with the same distribution as (X, Y)
- Compute estimators $\hat{w}$ and $\hat{b}$ by **maximum likelihood estimation**
- Or equivalently, minimize the minus log-likelihood
- More generally, when a model is used

$$\text{Goodness-of-fit} = -\log \text{ likelihood}$$

- log is used mainly since averages are easier to study (and compute) than products

Logistic regression

Likelihood is given by

$$\begin{aligned}
&\prod_{i=1}^{n} \mathbb{P}(Y = y_i | X = x_i) \\
&= \prod_{i=1}^{n} \sigma(\langle w, x_i \rangle + b)^{\frac{1+y_i}{2}} \left(1 - \sigma(\langle w, x_i \rangle + b)\right)^{\frac{1-y_i}{2}} \\
&= \prod_{i=1}^{n} \sigma(\langle w, x_i \rangle + b)^{\frac{1+y_i}{2}} \sigma(-\langle w, x_i \rangle - b)^{\frac{1-y_i}{2}}
\end{aligned}$$

and the minus log-likelihood is given by

$$\sum_{i=1}^{n} \log(1 + e^{-y_i(\langle w, x_i \rangle + b)})$$

Logistic regression

Compute $\hat{w}$ and $\hat{b}$ as follows:

$$(\hat{w}, \hat{b}) \in \underset{w \in \mathbb{R}^d, b \in \mathbb{R}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \log(1 + e^{-y_i(\langle w, x_i \rangle + b)})$$

- It is an **average of losses**, one for each sample point
- It is a convex and smooth problem
- Many ways to find an approximate minimizer
- Convex optimization algorithms

If we introduce the **logistic loss** function

$$\ell(y, y') = \log(1 + e^{-yy'})$$

then

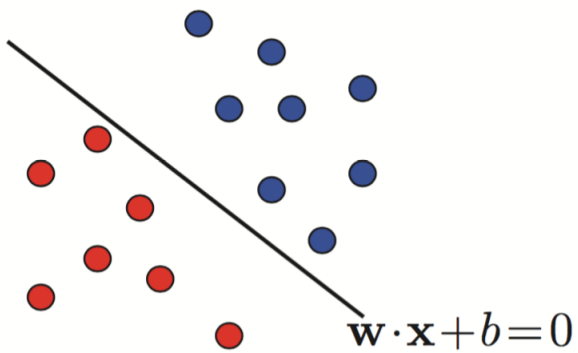
$$(\hat{w}, \hat{b}) \in \underset{w \in \mathbb{R}^d, b \in \mathbb{R}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \langle w, x_i \rangle + b)$$

Outline

Support Vector Machine

A dataset is **linearly separable** if we can find an hyperplane H that puts

- Points $x_i \in \mathbb{R}^d$ such that $y_i = 1$ on one side of the hyperplane
- Points $x_i \in \mathbb{R}^d$ such that $y_i = -1$ on the other
- H do not pass through a point x_i



An hyperplane

$$H = \{x \in \mathbb{R}^d : \langle w, x \rangle + b = 0\}$$

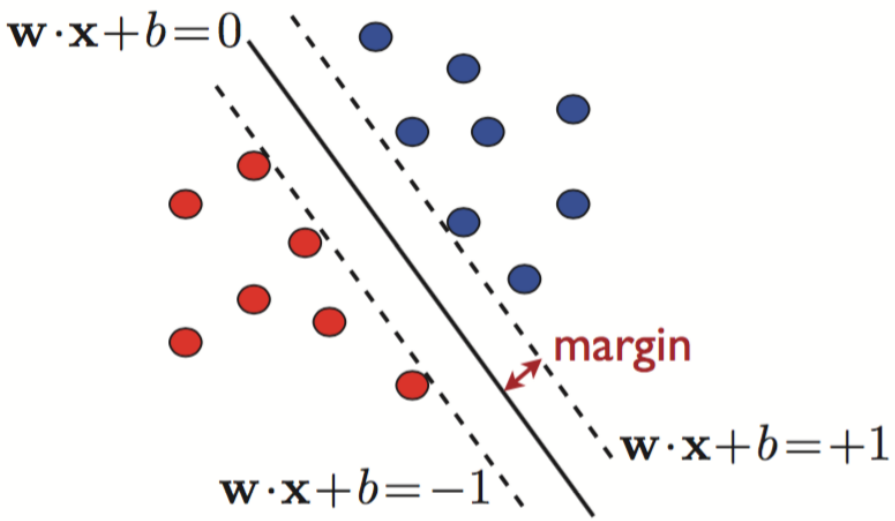
is a translation of a set of vectors orthogonal to w.

Support Vector Machine

The definition of H is invariant by multiplication of w and b by a non-zero scalar

If H do not pass through any sample point x_i, we can scale w and b so that

$$\min_{(x_i,y_i)\in\mathcal{D}_n} |\langle w, x_i\rangle + b| = 1$$

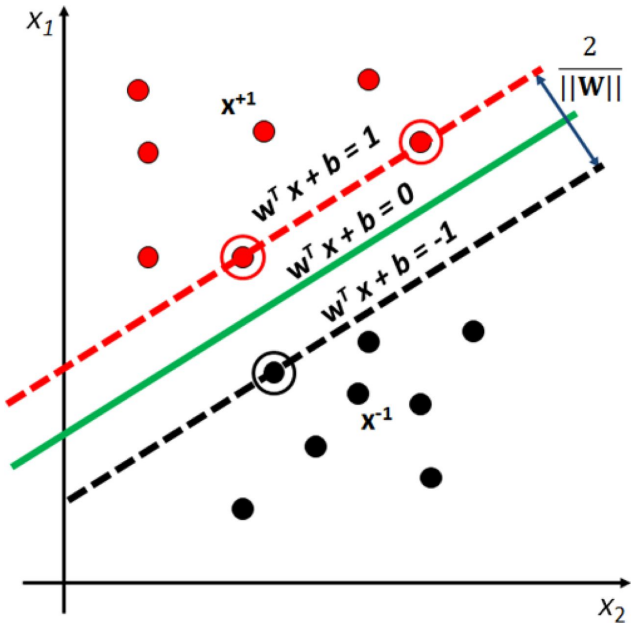


For such w and b, we call H the *canonical* hyperplane

Support Vector Machine

The distance of any point $x' \in \mathbb{R}^d$ to H is given by

$$\frac{|\langle w, x' \rangle + b|}{\|w\|}$$



So, if H is a canonical hyperplane, its **margin** is given by

$$\max_{(x_i, y_i) \in \mathcal{D}_n} \frac{|\langle w, x_i \rangle + b|}{\|w\|} = \frac{1}{\|w\|}.$$

In summary.

If $\mathcal{D}_n$ is strictly linearly separable, we can find a canonical separating hyperplane

$$H = \{x \in \mathbb{R}^d : \langle w, x \rangle + b = 0\}.$$

that satisfies

$$|\langle w, x_i \rangle + b| \geqslant 1 \text{ for any } i = 1, \dots, n,$$

which entails that a point x_i is correctly classified if

$$y_i(\langle w, x_i \rangle + b) \geqslant 1.$$

The margin of H is equal to $1/\|w\|$.

Linear SVM: separable case

From that, we deduce that a way of classifying $\mathcal{D}_n$ with maximum margin is to solve the following problem:

$$\min_{w \in \mathbb{R}^d, b \in \mathbb{R}} \frac{1}{2} \|w\|_2^2$$

$$\text{subject to} \quad y_i(\langle w, x_i \rangle + b) \geqslant 1 \quad \text{for all} \quad i = 1, \dots, n$$

Note that:

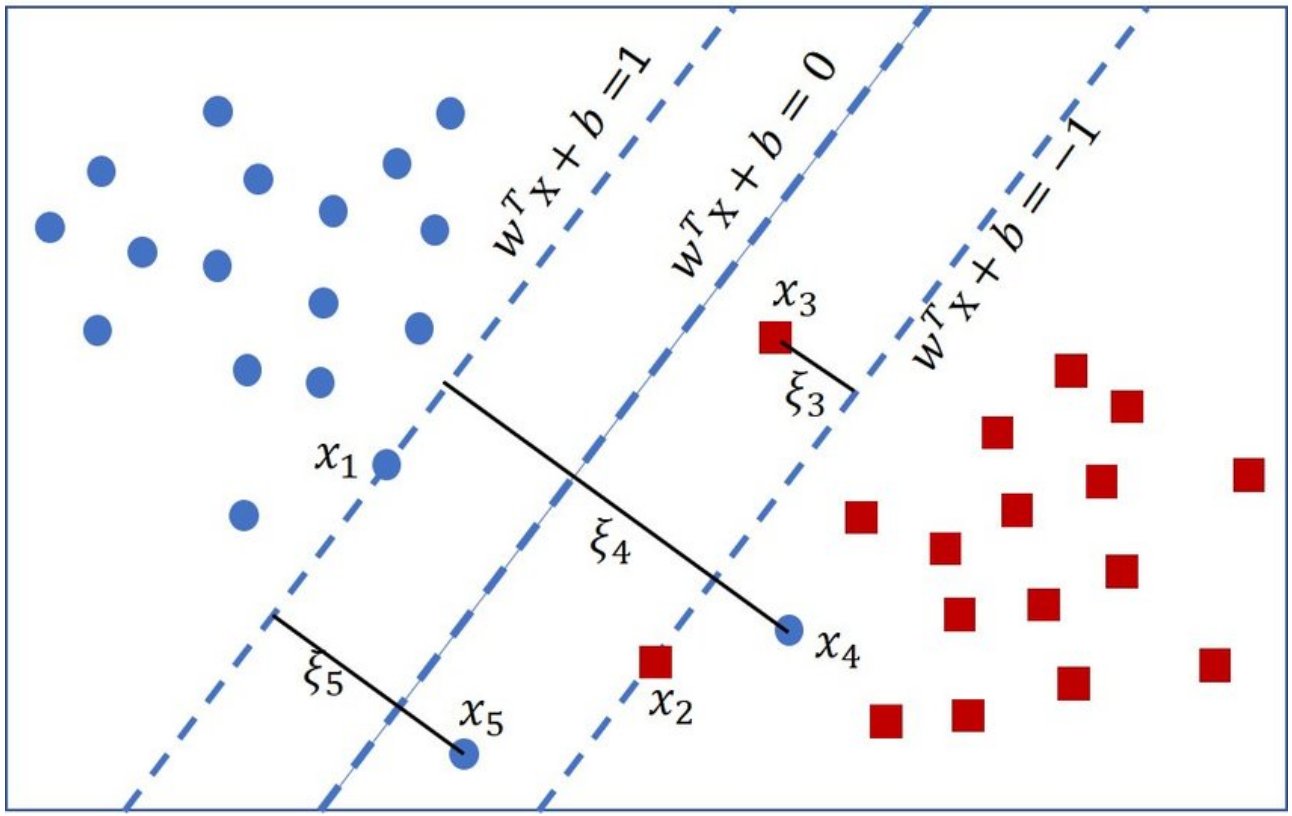
- This problem admits a **unique** solution
- It is a "quadratic programming" problem, which is easy to solve numerically
- Dedicated optimization algorithms can solve this on a large scale very efficiently

SVM for the non linearly separable case

Introducing slack variables $\xi_i \geqslant 0$.

Modeling potential errors

$$(x_i, y_i) \begin{cases} \text{no error:} & y_i(\langle w, x_i \rangle + b) \geqslant 1 \Rightarrow \xi_i = 0 \\ \text{error:} & y_i(\langle w, x_i \rangle + b) < 1 \Rightarrow \xi_i = 1 - y_i(\langle w, x_i \rangle + b) > 0 \end{cases}$$



New optimization problem

$$\min_{w,b,\xi} \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{subject to, for all } i = 1,\dots,n,$$

$$y_i(\langle w, x_i\rangle + b) \geqslant 1 - \xi_i$$

$$\xi_i \geqslant 0.$$

Introducing the hinge loss $\ell(y, y') = \max(0, 1 - yy')$, the optimization can be rewritten as

SVM with hinge loss

$$\min_{w,b} \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n}\ell(y_i, \hat{y}_i).$$

Outline

General optimization problem

We have seen a lot of problems of the form

$$\operatorname*{argmin}_{w \in \mathbb{R}^d} f(w) + g(w)$$

with f a goodness-of-fit function

$$f(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \langle w, x_i \rangle)$$

where ℓ is some loss and

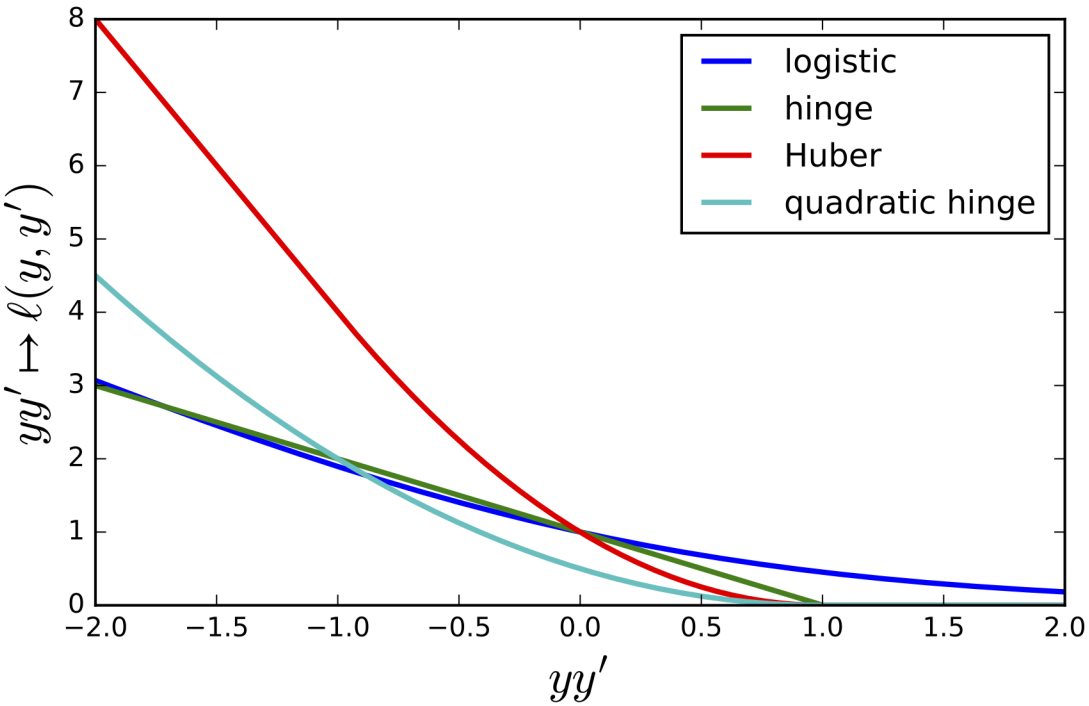
$$g(w) = \lambda \mathrm{pen}(w)$$

where $\mathrm{pen}(\cdot)$ is some penalization function, examples being

- $\mathrm{pen}(w) = \|w\|_2^2$ (ridge)
- $\mathrm{pen}(w) = \|w\|_1$ (Lasso)

Different losses for classification

- Logistic loss, $\ell(y, y') = \log(1 + e^{-yy'})$
- Hinge loss, $\ell(y, y') = (1 - yy')_+$
- Quadratic hinge loss, $\ell(y, y') = \frac{1}{2}(1 - yy')_+^2$
- Huber loss $\ell(y, y') = -4yy' \mathbb{1}_{yy' < -1} + (1 - yy')_+^2 \mathbb{1}_{yy' \geqslant -1}$



- These losses can be understood as a convex approximation of the 0/1 loss $\ell(y, y') = \mathbb{1}_{yy' \leqslant 0}$

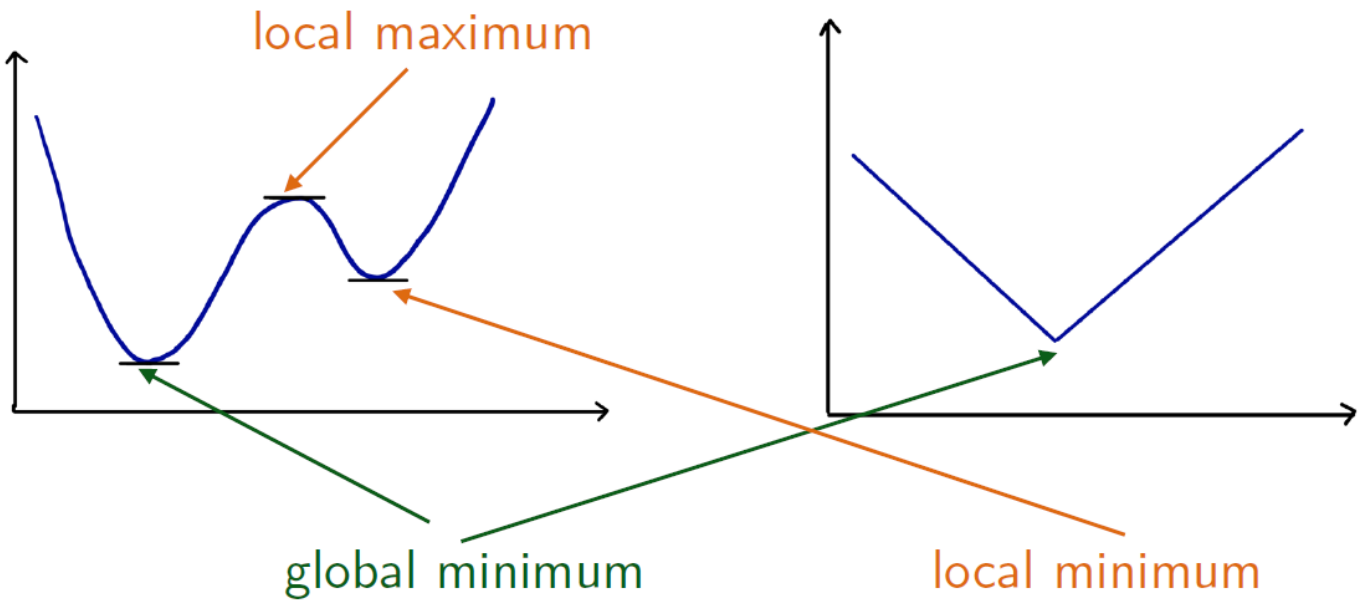
Outline

Outline

Minimization problems

Aim: minimizing a function $h : \mathbb{R}^d \to \mathbb{R}$

d: dimension of the search space.

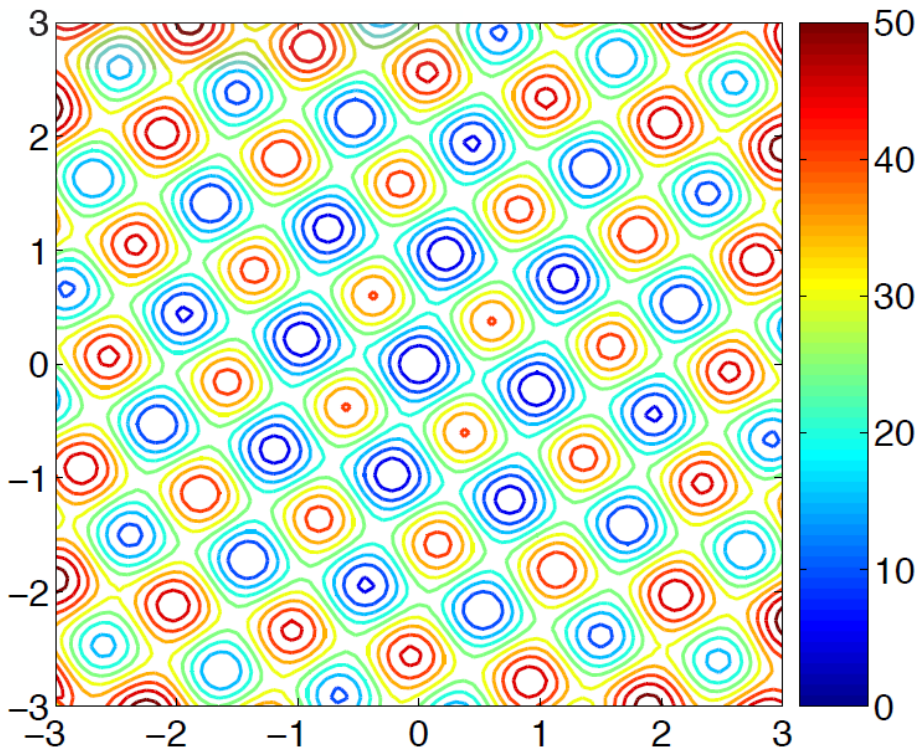


Level sets

One-dimensional (1-D) representations are often misleading, we therefore often represent level-sets of functions

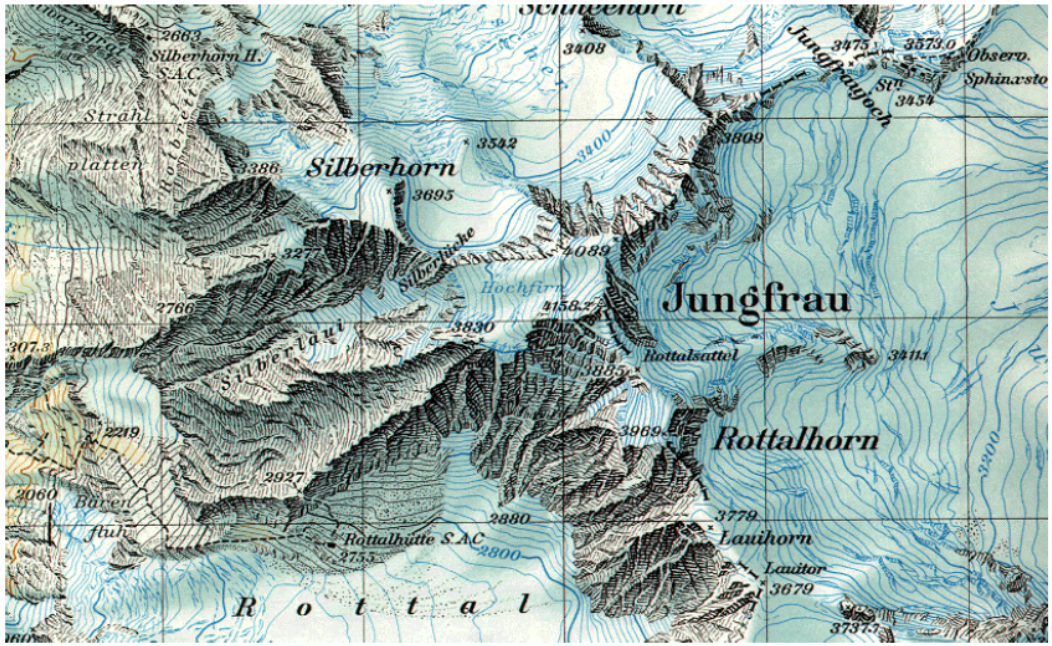
$$\mathcal{C}_c = \{\mathbf{x} \in \mathbb{R}^d, f(\mathbf{x}) = c\}.$$

Example of level sets in dimension two



Topographic sets

The function is the altitude!



See `https://mathinsight.org/applet/directional_derivative_mountain`

Exhaustive search

Consider the problem

$$w^\star \in \operatorname*{argmin}_{w \in [0,1]^d} f(w).$$

One can optimize this problem on a grid of $[0,1]^d$. For example, if the function f is regular enough, in dimension 1, to achieve a precision of ε we need $\lfloor 1/\varepsilon \rfloor$ evaluation of f.

Exhaustive search

Consider the problem

$$w^\star \in \underset{w \in [0,1]^d}{\operatorname{argmin}} f(w).$$

One can optimize this problem on a grid of $[0,1]^d$. For example, if the function f is regular enough, in dimension 1, to achieve a precision of ε we need $\lfloor 1/\varepsilon \rfloor$ evaluation of f. In dimension d, we need $\lfloor 1/\varepsilon \rfloor^d$ evaluations.

For example, evaluating the expression

$$f(x) = \sum_{i=1}^{n} x_i^2,$$

to obtain a precision of $\varepsilon = 10^{-2}$ requires:

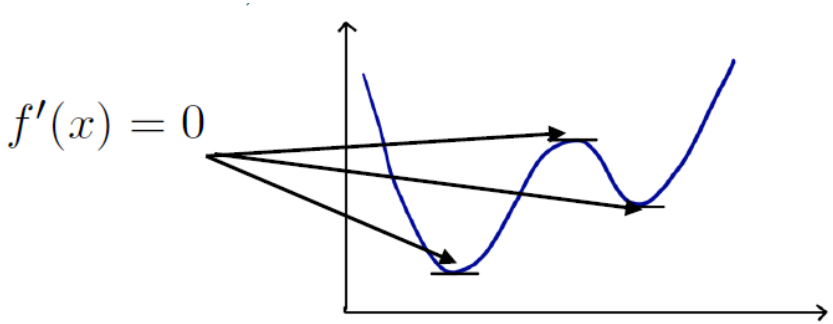
- $1,75.10^{-3}$ seconds in dimension 1
- $1,75.10^{15}$ seconds in dimension 10, i.e., nearly 32 millions years.

$\rightarrow$ Prohibitive in high dimensions (curse of dimensionality, term introduced by Richard **bellman2013dynamic**)

Necessary condition

First order necessary condition

- In dimension one.
  Let $f : \mathbb{R} \to \mathbb{R}$ be a differentiable function. If $x^\star$ is a local extremum (minimum/maximum) then $f'(x^\star) = 0$.



- Generalization for $d > 1$.
  Let $f : \mathbb{R}^d \to \mathbb{R}$ be a differentiable function. If $x^\star$ is a local extremum then $\nabla f(x^\star) = 0$.

Remark.

- Points such that $\nabla f(x^\star) = 0$ are called critical points.
- Critical points are not always extrema (consider $x \mapsto x^3$)

Gradient - Definition

The gradient of a function $f : \mathbb{R}^d \to \mathbb{R}$ in x denoted as $\nabla f(x)$ is the vector of partial derivatives

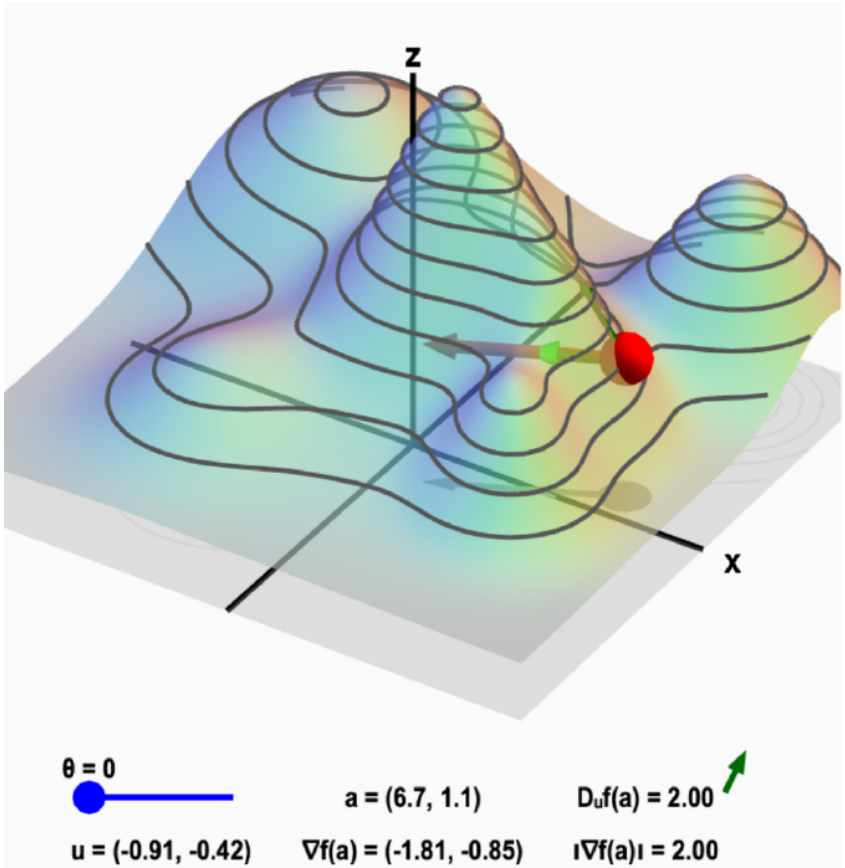
$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{pmatrix}$$

Exercise

- If $f : \mathbb{R} \to \mathbb{R}$, $\nabla f(x) = f'(x)$
- $f(x) = \langle a, x \rangle$: $\nabla f(x) = a$
- $f(x) = x^T A x$: $\nabla f(x) = (A + A^T)x$
- Particular case: $f(x) = \|x\|^2$, $\nabla f(x) = 2x$.

Gradient - Level sets

The gradient is orthogonal to level sets.



Heuristic: why gradient descent works?

For a function $f : \mathbb{R}^d \to \mathbb{R}$, define the level sets:

$$\mathcal{C}_c = \{\mathbf{x} \in \mathbb{R}^d, f(\mathbf{x}) = c\}.$$

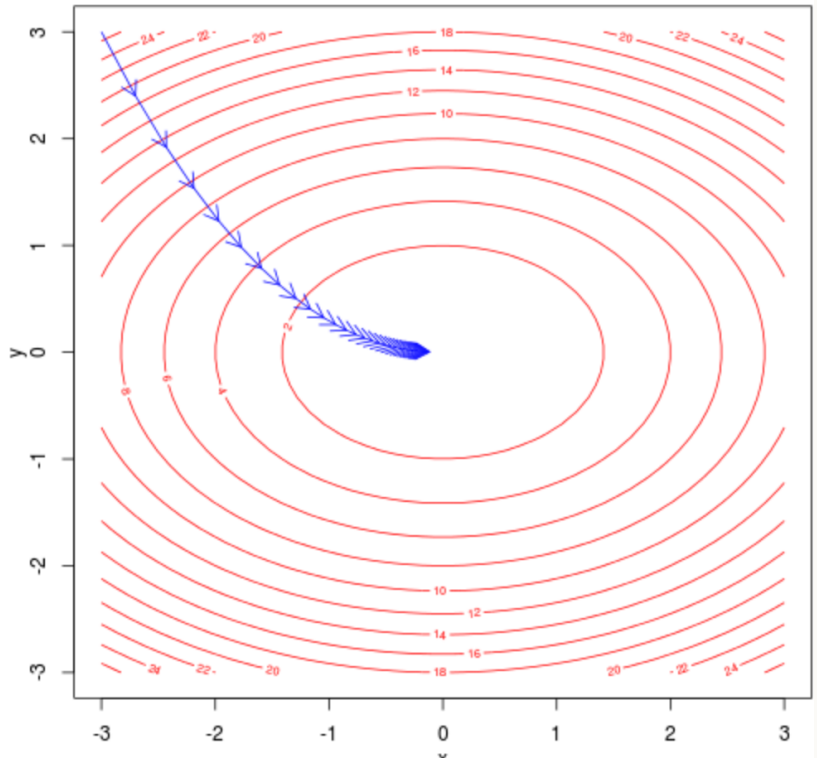


Figure: Gradient descent for function $f : (x, y) \mapsto x^2 + 2y^2$

Exercise:

1. The gradient is orthogonal to level sets.
2. The gradient is a good direction to follow, if step size is small enough.

Gradient orthogonal to level sets

1. Locally near $w^{(0)}$,
$$f(w) = f(w^{(0)}) + \langle \nabla f(w^{(0)}), w - w^{(0)} \rangle + O(\|w - w^{(0)}\|^2).$$
Thus, for all $w \in \mathcal{C}_{f(w^{(0)})}$,
$$\lim_{w \to w^{(0)}, w \in \mathcal{C}_{f(w^{(0)})}} \langle \nabla f(w^{(0)}), \frac{w - w^{(0)}}{\|w - w^{(0)}\|} \rangle = 0.$$
2. Locally near $w^{(0)}$,
$$f(w) = f(w^{(0)}) + \langle \nabla f(w^{(0)}), w - w^{(0)} \rangle + O(\|w - w^{(0)}\|^2).$$
Thus, locally, minimizing $f(w)$ is equivalent to
$$\operatorname*{argmin}_{w \in B(w^{(0)}, \varepsilon)} f(w^{(0)}) + \langle \nabla f(w^{(0)}), w - w^{(0)} \rangle,$$
for ε small enough, that is
$$w - w^{(0)} = -\eta \nabla f(w^{(0)}),$$
for some $\eta > 0$. This gives the final gradient descent equation
$$w = w^{(0)} - \eta \nabla f(w^{(0)}),$$

Bad objective functions

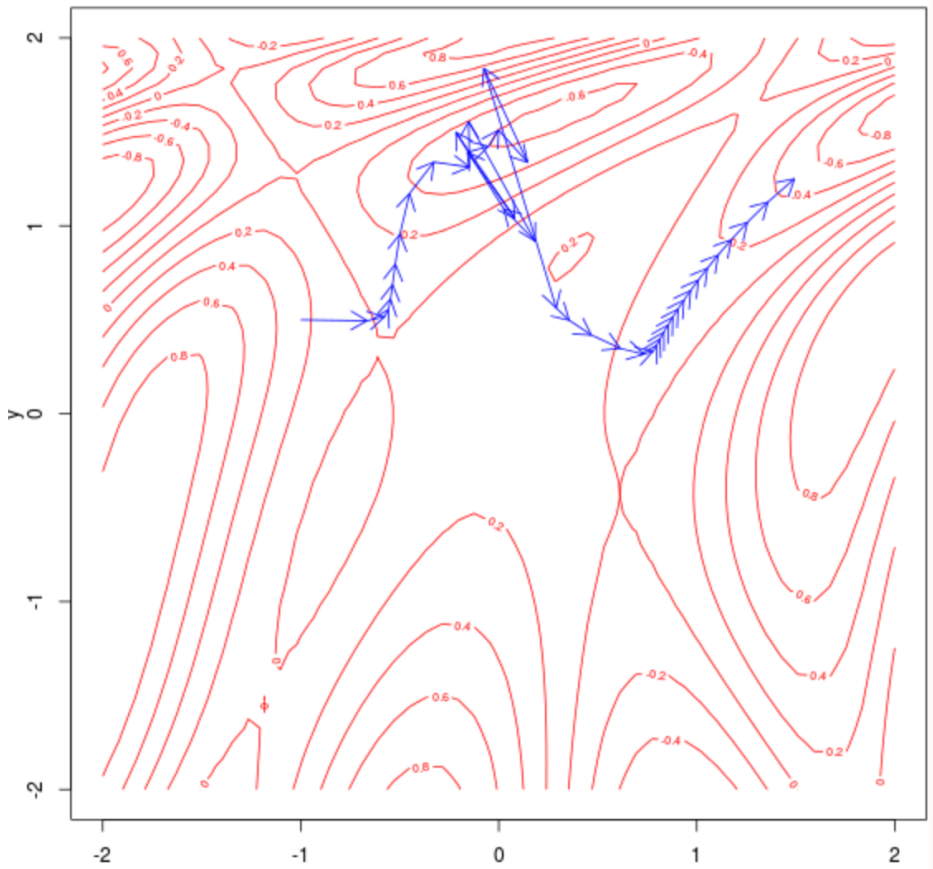


Figure: Gradient descent for $f : (x, y) \mapsto \sin(1/(2x^2) - 1/(4y^2) + 3)\cos(2x + 1 - \exp(y))$

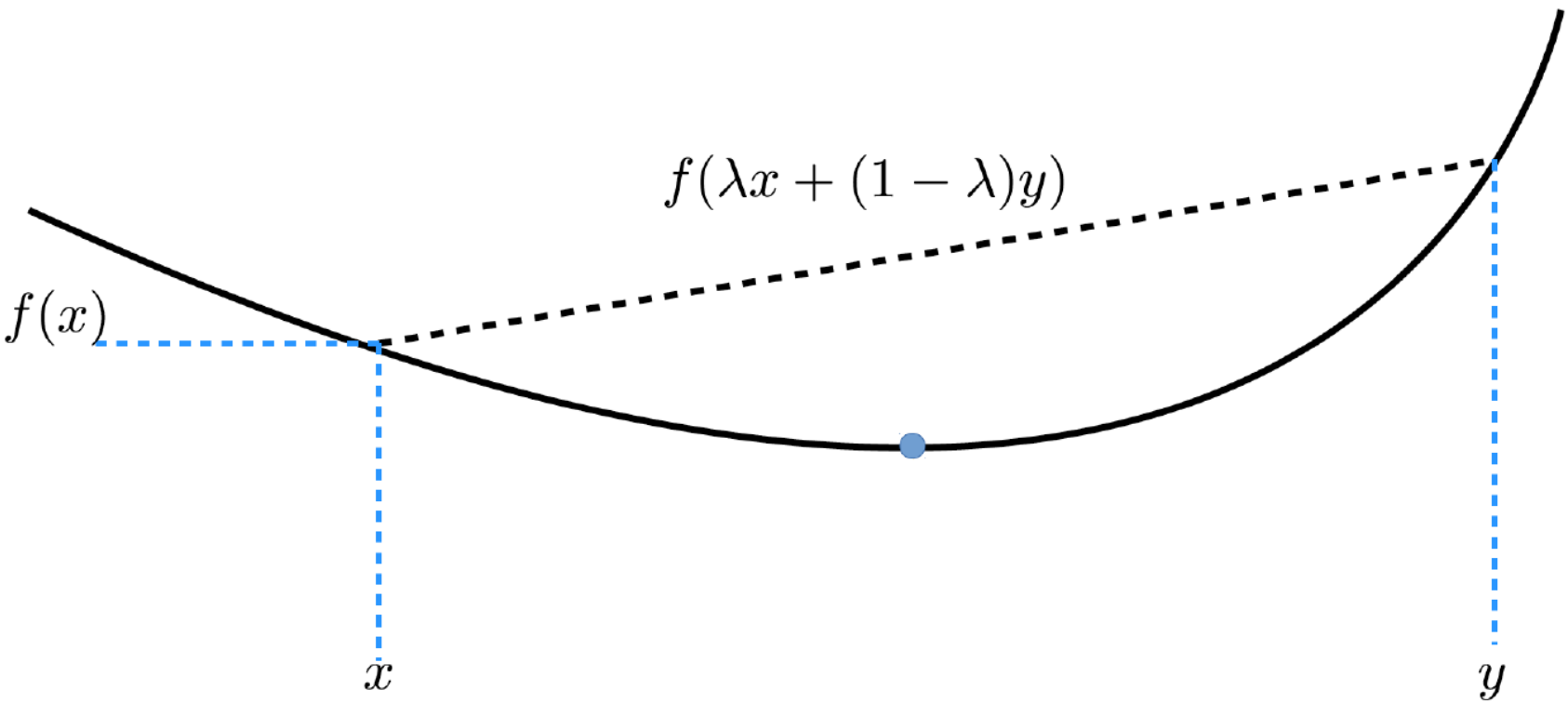
`http://vis.supstat.com/2013/03/gradient-descent-algorithm-with-r/`

Convexity

Convexity - Definition

We say that $f : \mathbb{R}^d \to \mathbb{R}$ is convex if ($\mathbb{R}^d$ is convex and if)

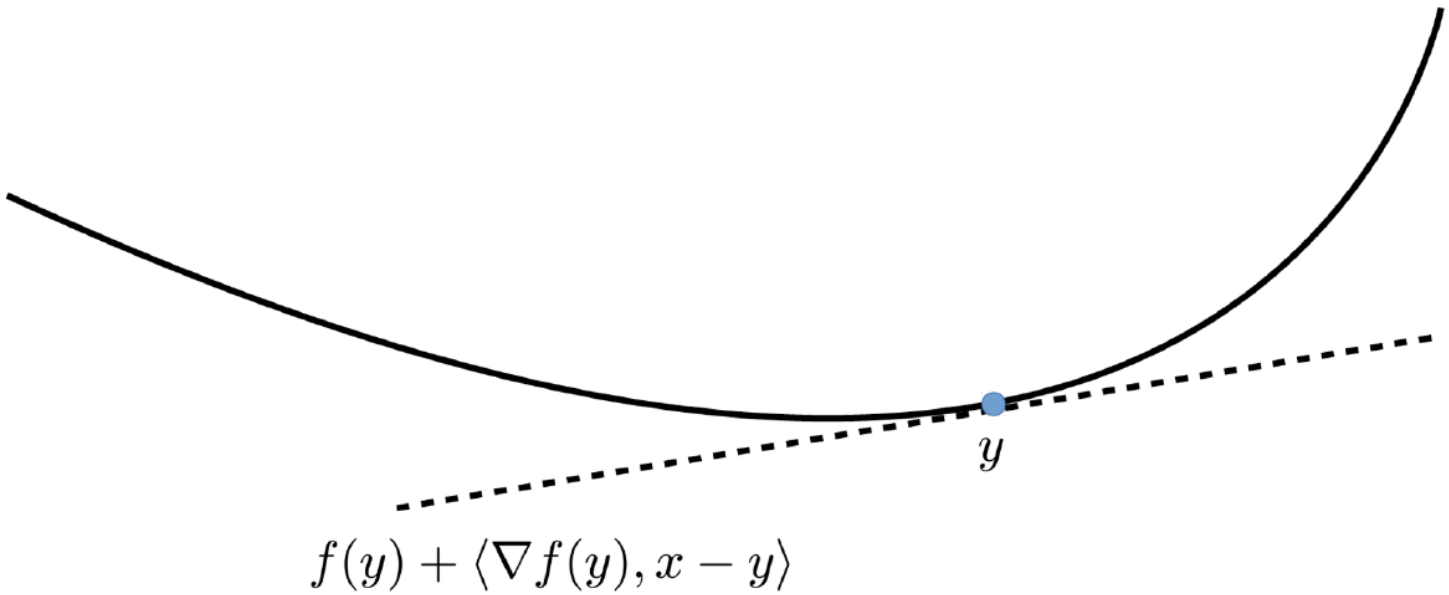
$$f(\lambda x + (1-\lambda)y) \leqslant \lambda f(x) + (1-\lambda)f(y), \quad \text{for all } x, y \in \mathbb{R}^d, \lambda \in [0,1].$$



Convexity - First derivative

A twice differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if and only if

$$f(x) \geqslant f(y) + \langle \nabla f(y), x - y \rangle, \quad \text{for all } x, y \in \mathbb{R}^d.$$



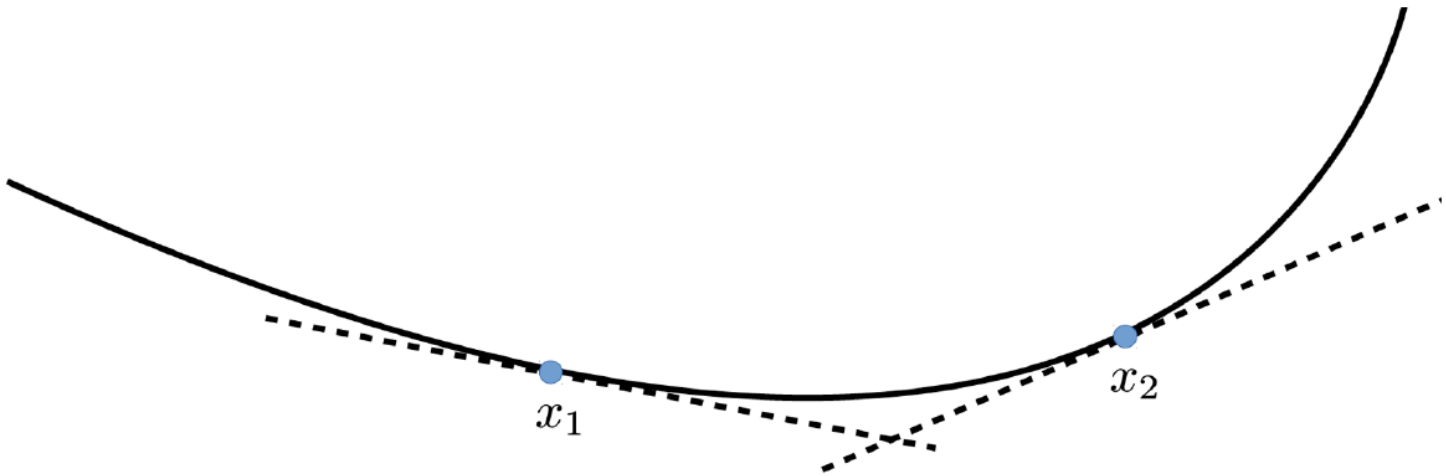
Convexity 3

Convexity - Hessian

A twice differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if and only if

$$\nabla^2 f(x) \succeq 0, \quad \text{for all } x,$$

that is $h^T \nabla^2 f(x) h \geqslant 0$, for all $h \in \mathbb{R}^d$.



$$x_1 \leq x_2 \quad \Rightarrow f'(x_1) \leq f'(x_2)$$

Hessian

If $f : \mathbb{R}^d \to \mathbb{R}$ is twice differentiable, the Hessian matrix in x denoted by $\nabla^2 f(x)$ is given by

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2}(x) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(x) & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_d}(x) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(x) & \frac{\partial^2 f}{\partial x_2^2}(x) & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_d}(x) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1}(x) & \frac{\partial^2 f}{\partial x_d \partial x_2}(x) & \dots & \frac{\partial^2 f}{\partial x_d^2}(x) \end{pmatrix}$$

The Hessian matrix is symmetric if f is twice continuously differentiable (C^2).

Optimality conditions: second order

Assume that f is twice continuously differentiable (C^2).

Necessary condition

If $x^\star$ is a local minimum, then $\nabla f(x^\star) = 0$ and $\nabla^2 f(x^\star)$ is positive semi-definite.

Sufficient condition

If $\nabla f(x^\star) = 0$ and $\nabla^2 f(x^\star)$ is positive definite then $x^\star$ is a strict local optimum.

Remark. For $d = 1$, this condition boils down to $f'(x^\star) = 0$ and $f''(x^\star) > 0$.

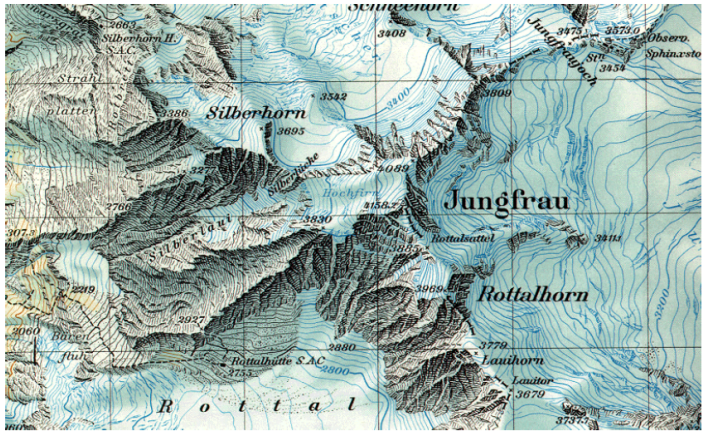
Classes of algorithms

In this lecture, we are going to study iterative algorithms. There are two classes of such algorithms, depending on the information that is used to compute the next iteration.

First-order algorithms that use f and ∇f. Standard algorithms when f is differentiable and convex.

Second-order algorithms that use $f, \nabla f$ and $\nabla^2 f$. They are useful when computing the Hessian matrix is not too costly.

Gradient descent algorithm



Gradient descent

Input: Function f to minimize, initial vector $w^{(0)}$, $k = 0$.

Parameters: step size $\eta > 0$.

While *not converge* do

- $w^{(k+1)} \leftarrow w^{(k)} - \eta \nabla f(w^{(k)})$
- $k \leftarrow k + 1$.

Output: $w^{(k)}$.

When does gradient descent converge?

Convex function

A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex on $\mathbb{R}^d$ if, for all $x, y \in \mathbb{R}^d$, for all $\lambda \in [0, 1]$,

$$f(\lambda x + (1 - \lambda)y) \leqslant \lambda f(x) + (1 - \lambda) f(y).$$

L-smooth function

A function f is said to be L-smooth if f is differentiable and if, for all $x, y \in \mathbb{R}^d$,

$$\|\nabla f(x) - \nabla f(y)\| \leqslant L\|x - y\|.$$

Exercise: If f is twice differentiable, this is equivalent to writing that for all $x \in \mathbb{R}^d$,

$$\lambda_{max}(\nabla^2 f(x)) \leqslant L.$$

Proof

Proposition

If f is twice differentiable, f is L-smooth if and only if for all $x \in \mathbb{R}^d$,

$$\lambda_{max}(\nabla^2 f(x)) \leqslant L.$$

Proof

Fix $x, y \in \mathbb{R}^d$ and $c > 0$. Let $g(t) = \nabla f(x + tcy)$. Thus, $g'(t) = [\nabla^2 f(x + tcy)](cy)$.
By the mean value theorem, there exists some constant $t_c \in [0, 1]$ such that

$$\nabla f(x + cy) - \nabla f(x) = g(1) - g(0) = g'(t_c) = [\nabla^2 f(x + t_c cy)](cy). \tag{1}$$

First implication

Taking the norm of both sides of (1) and applying the smoothness condition, we obtain

$$\|[\nabla^2 f(x + t_c cy)]y\| \leqslant L\|y\|.$$

By taking $c \to 0$ and using the fact that $t_c \in [0, 1]$ and $f \in C^2$, we have

$$\|[\nabla^2 f(x)]y\| \leqslant L\|y\|.$$

Then, $\lambda_{max}(\nabla^2 f(x)) \leqslant L$.

Proof

Second implication

Taking the norm of both sides of (1), we have

$$\|\nabla f(x+cy) - \nabla f(x)\|_2 = \|[\nabla^2 f(x+t_c cy)](cy)\|_2.$$

Note that, for any real-valued symmetric matrix A and any vector u,

$$\|Au\|_2^2 = u^T A^T Au = \langle A^T Au, u\rangle \leqslant \lambda_{max}(A)^2 \|u\|^2$$

Thus,

$$\|\nabla f(x+cy) - \nabla f(x)\|_2 \leqslant \lambda_{max}([\nabla^2 f(x+t_c cy)])\|(cy)\|_2 \leqslant L\|cy\|_2.$$

Convergence of GD

Theorem

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a L-smooth convex function. Let $w^\star$ be the minimum of f on $\mathbb{R}^d$. Then, Gradient Descent with step size $\eta \leqslant 1/L$ satisfies

$$f(w^{(k)}) - f(w^\star) \leqslant \frac{\|w^{(0)} - w^\star\|_2^2}{2\eta k}.$$

In particular, for $\eta = 1/L$,

$$L\|w^{(0)} - w^\star\|_2^2/2$$

iterations are sufficient to get an ε-approximation of the minimal value of f.

Descent Lemma

A key point: the descent lemma.

If f is L-smooth, then for any $w, w' \in \mathbb{R}^d$

$$f(w') \leqslant f(w) + \langle \nabla f(w), w' - w \rangle + \frac{L}{2}\|w - w'\|_2^2.$$

Assuming the descent Lemma holds, remark that

$$\begin{aligned} &\operatorname*{argmin}_{w \in \mathbb{R}^d} \left\{ f(w^k) + \langle \nabla f(w^k), w - w^k \rangle + \frac{L}{2}\|w - w^k\|_2^2 \right\} \\ &= \operatorname*{argmin}_{w \in \mathbb{R}^d} \left\| w - \left(w^k - \frac{1}{L}\nabla f(w^k) \right) \right\|_2^2 \end{aligned}$$

Hence, it is natural to choose

$$w^{k+1} = w^k - \frac{1}{L}\nabla f(w^k)$$

This is the basic **gradient descent** algorithm

Exercise: Prove the descent Lemma.

Proof - Descent Lemma for smooth functions

Using the fact that

$$\begin{aligned} f(w') &= f(w) + \int_0^1 \langle \nabla f(w + t(w' - w)), w' - w \rangle dt \\ &= f(w) + \langle \nabla f(w), w' - w \rangle \\ &\quad + \int_0^1 \langle \nabla f(w + t(w' - w)) - \nabla f(w), w' - w \rangle dt, \end{aligned}$$

so that

$$\begin{aligned} &|f(w') - f(w) - \langle \nabla f(w), w' - w \rangle| \\ &\leqslant \int_0^1 |\langle \nabla f(w + t(w' - w)) - \nabla f(w), w' - w \rangle dt| \\ &\leqslant \int_0^1 \|\nabla f(w + t(w' - w)) - \nabla f(w)\| \|w' - w\| dt \\ &\leqslant \int_0^1 Lt\|w' - w\|^2 dt = \frac{L}{2}\|w' - w\|^2, \end{aligned}$$

descent lemma is proved.

Faster rate for strongly convex function

Strong convexity

A function $f : \mathbb{R}^d \to R$ is μ-strongly convex if

$$x \mapsto f(x) - \frac{\mu}{2}\|x\|_2^2$$

is convex.

If f is differentiable it is equivalent to writing, for all $x \in \mathbb{R}^d$,

$$\lambda_{min}(\nabla^2 f(x)) \geqslant \mu.$$

This is also equivalent to, for all $x, y \in \mathbb{R}^d$,

$$f(y) \geqslant f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|y - x\|_2^2.$$

Theorem

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a L-smooth, μ strongly convex function. Let $w^\star$ be the minimum of f on $\mathbb{R}^d$. Then, Gradient Descent with step size $\eta \leqslant 1/L$ satisfies

$$f(w^{(k)}) - f(w^\star) \leqslant \Big(1 - \eta\mu\Big)^k \|f(w^{(0)}) - f(w^\star)\|_2^2.$$

In practice, how to choose η?

Do not set $\eta = 1/L$, it corresponds to the worst case scenario.

Exact line search
Instead, at each step, choose the best η by optimizing

$$\eta^{(k)} = \underset{\eta>0}{\operatorname{argmin}}\, f(w - \eta \nabla f(w)).$$

$\rightarrow$ Too costly!

Backtracking line search
First, fix a parameter $0 < \beta < 1$, then at each iteration, start with $t = 1$ and while

$$f(w - t\nabla f(w)) > f(w) - \frac{t}{2}\|\nabla f(w)\|^2,$$

update $t \leftarrow \beta t$.

$\rightarrow$ Simple and work pretty well in practice.

Backtracking line search

[armijo1966minimization armijo1966minimization]

First, fix a parameter $0 < \beta < 1$, then at each iteration, start with $\eta_k = 1$ and while

$$f(w^{(k)} - \eta_k \nabla f(w^{(k)})) - f(w^{(k)}) > -\frac{\eta_k}{2} \|\nabla f(w^{(k)})\|^2,$$

update $\eta_k \leftarrow \beta \eta_k$.

$\rightarrow$ Simple and work pretty well in practice.

Indeed, for $\eta > 0$ small enough,

$$f(w^{(k)} - \eta_k \nabla f(w^{(k)})) - f(w^{(k)}) = -\eta_k \|\nabla f(w^{(k)})\|^2 + o(\eta_k).$$

Theorem

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a L-smooth convex function. Let $w^\star$ be the minimum of f on $\mathbb{R}^d$. Then, Gradient Descent with backtracking line search satisfies

$$f(w^{(k)}) - f(w^\star) \leqslant \frac{\|w^{(0)} - w^\star\|_2^2}{2k \min(1, \beta/L)}.$$

Comparison of rates

Gradient descent uses iterations

$$w^{(k+1)} \leftarrow w^{(k)} - \eta \nabla f(w^{(k)})$$

- For L smooth convex function

$$f(w^{(k)}) - f(w^\star) \leqslant \frac{\|w^{(0)} - w^\star\|_2^2}{2\eta k}.$$

- For L smooth, μ strongly convex function

$$f(w^{(k)}) - f(w^\star) \leqslant \left(1 - \frac{\mu}{L}\right)^k \|f(w^{(0)}) - f(w^\star)\|_2^2.$$

Condition number $\kappa = L/\mu \geqslant 1$ stands for the difficulty of the learning problem.

Outline

Condition number

Condition number $\kappa = L/\mu \geqslant 1$

- Assuming that $\kappa = 1$, $\mu = L$, then, for all $x \in \mathbb{R}^d$

$$\nabla^2 f(x) = \mu I.$$

In that case, level sets of f are circles (in dimension two).

$\rightarrow$ Very easy optimization problem: gradient is directed to the global minimum of the function.

- Assuming that

$$f : (x, y) \mapsto \alpha_1 x^2 + \alpha_2 y^2,$$

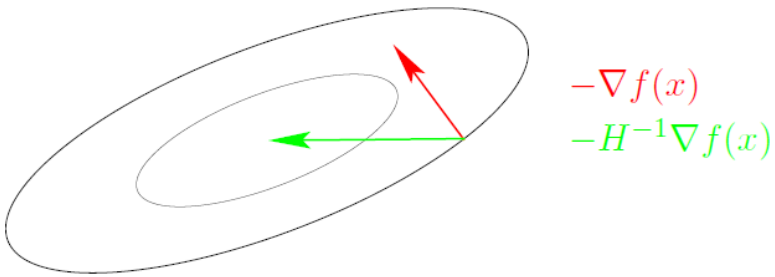
$\kappa \gg 1$ means that the level sets of f are ellipses where $\alpha_1 \gg \alpha_2$ or the opposite.

$\rightarrow$ Optimization is much more difficult because of the step size which is the same for both direction.

Ill-conditioned problems

Ill-conditioned problems are defined by a high condition number $\kappa = L/\mu \gg 1$, typically of order up to 10^{10} in real-world applications.

If level sets are ellipsoid, it means that there is a large ratio between the largest and smallest axis.



On ill-conditioned problems, the gradient descent algorithm is slow!

A better descent direction is given by

$$-H^{-1}\nabla f(x).$$

Newton algorithm

Take as descent direction, the Newton step:

$$d_k = -[\nabla^2 f(x_k)]^{-1} \nabla f(x_k)$$

The Newton's direction minimizes the best locally quadratic approximation of f. Indeed, by Taylor's expansion, we can approximate f locally around x by:

$$f(x+h) \simeq f(x) + \nabla f(x)^T h + \frac{1}{2} h^T \nabla^2 f(x) h.$$

Minimizing $f(x+h)$ with respect to h yields

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

Newton's Algorithm for Logistic regression

In the very specific case of logistic regression, we can have an explicit expression of the Newton's step and Newton's algorithm turns into the Iterative Reweighted Least Squares (IRWLS) that was presented in the logistic regression session.

Quasi-Newton's methods

In quasi-Newton's methods, the Newton direction is approximated by using only first order information (gradient).

Key idea: successive iterates and gradients ield second order information

$$q_k \simeq \nabla^2 f(x_{k+1}) p_k,$$

where

$$\begin{aligned} p_k &= x_{k+1} - x_k, \\ q_k &= \nabla f(x_{k+1}) - \nabla f(x_k). \end{aligned}$$

Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm

BFGS algorithm

B_k approximates the Hessian matrix at iteration k.

$$\begin{aligned} d_k &= -B_k^{-1}\nabla f(x_k), \\ x_{k+1} &= x_k + \sigma_k d_k \quad (\text{find } \sigma_k \text{ via line-search}) \\ y_k &= \nabla f(x_{k+1}) - \nabla f(x_k) \\ B_{k+1} &= B_k + \frac{y_k y_k^T}{y_k^T \sigma_k d_k} - \frac{B_k d_k d_k^T B_k}{d_k^T B_k d_k}. \end{aligned}$$

$\rightarrow$ Efficient update to compute the inverse of B_k.

Considered as the state-of-the-art quasi-Newton's algorithm!

Outline

Full gradients...

We say that these methods are based on **full gradients**, since at each iteration we need to compute

$$\nabla f(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w),$$

which depends on the whole dataset

Question. If n is large, computing $\nabla f(w)$ is long: need to pass on the whole data before doing a step towards the minimum!

Idea. Large datasets make your modern computer look old

Go back to "old" algorithms.

Stochastic Gradient Descent (SGD)

Stochastic gradients

If I choose uniformly at random $I \in \{1, \dots, n\}$, then

$$\mathbb{E}[\nabla f_I(w)] = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w) = \nabla f(w)$$

$\nabla f_I(w)$ is an **unbiased** but very noisy estimate of the full gradient $\nabla f(w)$

Computation of $\nabla f_I(w)$ only requires the I-th line of data ($O(d)$ and smaller for sparse data)

Stochastic Gradient Descent (SGD)

[robbins1985stochastic robbins1985stochastic]

Stochastic gradient descent algorithm

Input: starting point $w^{(0)}$, steps (learning rates) η_k
For $t = 1, 2, \ldots$ until *convergence* do

- Pick at random (uniformly) i_k in $\{1, \ldots, n\}$
- compute

$$w^{(k)} = w^{(k-1)} - \eta_k \nabla f_{i_k}(w^{(k-1)})$$

Return last $w^{(k)}$

Remarks

- Each iteration has complexity $O(d)$ instead of $O(nd)$ for full gradient methods
- Possible to reduce this to $O(s)$ when features are s-sparse using **lazy-updates**.

Convergence rate of SGD

Consider the stochastic gradient descent algorithm introduced previously but where each iteration is projected into the ball $B(0, R)$ with $R > 0$ fixed.

Let

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

Theorem

Assume that f is convex and that there exists $b > 0$ satisfying, for all $x \in B(0, R)$,

$$\|\nabla f_i(x)\| \leqslant b.$$

Besides, assume that all minima of f belong to $B(0, R)$. Then, setting $\eta_k = 2R/(b\sqrt{k})$,

$$\mathbb{E}\left[f\left(\frac{1}{k} \sum_{j=1}^{k} w^{(j)}\right)\right] - f(w^\star) \leqslant \frac{3Rb}{\sqrt{k}}$$

Convergence rate of SGD

Consider the stochastic gradient descent algorithm introduced previously but where each iteration is projected into the ball $B(0, R)$ with $R > 0$ fixed.

Let

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

Theorem

Assume that f is μ strongly convex and that there exists $b > 0$ satisfying, for all $x \in B(0, R)$,

$$\|\nabla f_i(x)\| \leqslant b.$$

Besides, assume that all minima of f belong to $B(0, R)$. Then, setting $\eta_k = 2/(\mu(k+1))$,

$$\mathbb{E}\Big[f\Big(\frac{2}{k(k+1)} \sum_{j=1}^{k} j w^{(j-1)}\Big)\Big] - f(w^\star) \leqslant \frac{2b^2}{\mu(k+1)}.$$

Comparison of GD and SGD

Full gradient descent

$$w^{(k+1)} \leftarrow w^{(k)} - \eta_k \left(\frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w^{(k)}) \right)$$

- $O(nd)$ iterations
- Upper bound $O((1 - (\mu/L))^k)$
- Numerical complexity $O(n \frac{L}{\mu} \log(\frac{1}{\varepsilon})))$

Stochastic gradient descent

$$w^{(k+1)} \leftarrow w^{(k)} - \eta_k \nabla f_{i_k}(w^{(k)}).$$

- $O(d)$ iterations
- Upper bound $O(1/(\mu k))$
- Numerical complexity $O(\frac{1}{\mu \varepsilon})$

It does not depend on n for SGD !

Comparison GD versus SGD

Under strong convexity, GD versus SGD is

$$O\Big(\frac{nL}{\mu}\log\big(\frac{1}{\varepsilon}\big)\Big) \quad \text{versus} \quad O\Big(\frac{1}{\mu\varepsilon}\Big)$$

GD leads to a more accurate solution, but what if n is very large?

Recipe

- SGD is extremely fast in the early iterations (first two passes on the data)
- But it fails to converge accurately to the minimum

Beyond SGD

- Bottou and LeCun (2005),
- Shalev-Shwartz et al (2007, 2009),
- Nesterov et al. (2008, 2009),
- Bach et al. (2011, 2012, 2014, 2015),
- T. Zhang et al. (2014, 2015).

Improving stochastic gradient descent

The problem

- Put $X = \nabla f_I(w)$ with I uniformly chosen at random in $\{1, \dots, n\}$
- In SGD we use $X = \nabla f_I(w)$ as an approximation of $\mathbb{E}X = \nabla f(w)$
- How to reduce $\mathbb{V}X$?

Improving stochastic gradient descent

An idea

- Reduce it by finding C s.t. $\mathbb{E}C$ is "easy" to compute and such that C is highly correlated with X
- Put $Z_\alpha = \alpha(X - C) + \mathbb{E}C$ for $\alpha \in [0, 1]$. We have
$$\mathbb{E}Z_\alpha = \alpha\mathbb{E}X + (1 - \alpha)\mathbb{E}C$$
and
$$\mathbb{V}Z_\alpha = \alpha^2(\mathbb{V}X + \mathbb{V}C - 2\mathbb{C}(X, C))$$
- Standard variance reduction: $\alpha = 1$, so that $\mathbb{E}Z_\alpha = \mathbb{E}X$ (unbiased)

Improving stochastic gradient descent

Variance reduction of the gradient

In the iterations of SGD, replace $\nabla f_{i_t}(w^{(t-1)})$ by

$$\alpha(\nabla f_{i_t}(w^{(t-1)}) - \nabla f_{i_t}(\tilde{w})) + \nabla f(\tilde{w})$$

where $\tilde{w}$ is an "old" value of the iterate.

Several cases

- $\alpha = 1/n$: SAG (Bach et al. 2013)
- $\alpha = 1$: SVRG (T. Zhang et al. 2015, 2015)
- $\alpha = 1$: SAGA (Bach et al., 2014)

Important remark

- In these algorithms, the step-size η is kept **constant**
- Leads to **linearly convergent algorithms**, with a numerical complexity comparable to SGD!

Improving stochastic gradient descent

Stochastic Average Gradient

Input: starting point $w^{(0)}$, learning rate $\eta > 0$
For $k = 1, 2, \ldots$ until *convergence* do

- Pick uniformly at random i_k in $\{1, \ldots, n\}$
- Put

$$g_k(i) = \begin{cases} \nabla f_i(w^{(k-1)}) & \text{if } i = i_k \\ g_{k-1}(i) & \text{otherwise} \end{cases}$$

- Compute

$$w^{(k)} = w^{(k-1)} - \frac{\eta}{n} \sum_{i=1}^{n} g_k(i)$$

Return last $w^{(k)}$.

Improving stochastic gradient descent

Stochastic Variance Reduced Gradient

Input: starting point $\tilde{w}^{(0)}$, learning rate $\eta > 0$, phase size (typically $m = n$ or $m = 2n$).
For $k = 1, 2, \ldots$ to iterations do

- Compute $\nabla f(\tilde{w})$
- Put $w^{(0)} \leftarrow \tilde{w}$
- For $t = 0, \ldots,$ insideloop
  - Pick uniformly at random i_t in $\{1, \ldots, n\}$
  - Apply the step

$$w^{(t+1)} \leftarrow w^{(t)} - \eta(\nabla f_i(w^{(t)}) - \nabla f_i(\tilde{w}) + \nabla f(\tilde{w}))$$

- Set

$$\tilde{w} \leftarrow \frac{1}{m} \sum_{t=1}^{m} w^{(t)}$$

Return $\tilde{w}$.

Improving stochastic gradient descent

SAGA

Input: starting point $w^{(0)}$, learning rate $\eta > 0$
Compute $g_0(i) \leftarrow \nabla f_i(w^{(0)})$ for all $i = 1, \dots, n$
For $k = 1, 2, \dots$ until *convergence* do

- Pick uniformly at random i_k in $\{1, \dots, n\}$
- Compute $\nabla f_{i_k}(w^{(k-1)})$
- Apply

$$w^{(k)} \leftarrow w^{(k-1)} - \eta\left(\nabla f_{i_k}(w^{(k-1)}) - g_{k-1}(i_k) + \frac{1}{n}\sum_{i=1}^{n} g_{k-1}(i)\right)$$

- Store $g_k(i_k) \leftarrow \nabla f_{i_k}(w^{(k-1)})$

Return last $w^{(k)}$

Outline

Momentum algorithm

Aim: taking into account the previous update as additional velocity to avoid getting stuck into local minima.

Particularly useful for stochastic gradient descent.

https://distill.pub/2017/momentum/

Momentum algorithm

[polyak1964some polyak1964some]

Polyak's momentum algorithm - Heavy ball method

Input: starting point $w^{(0)}$, learning rate $\eta_k > 0$, initial velocity $v^{(0)} = 0$, momentum $\beta \in [0, 1]$ (default $\beta = 0.9$).
While *not converge* do

- $v^{(k)} = \beta(w^{(k)} - w^{(k-1)}) - \eta_k \nabla f(w^{(k)})$
- $w^{(k+1)} = w^{(k)} + v^{(k)}$
- $k \leftarrow k + 1$

Return last $w^{(k+1)}$.

If the step size $\eta_k = \eta$ is constant, the update equations can be written

$$w^{(k+1)} = w^{(k)} - \eta \sum_{t=1}^{k} \beta^{k-t} \nabla f(w^{(t)}).$$

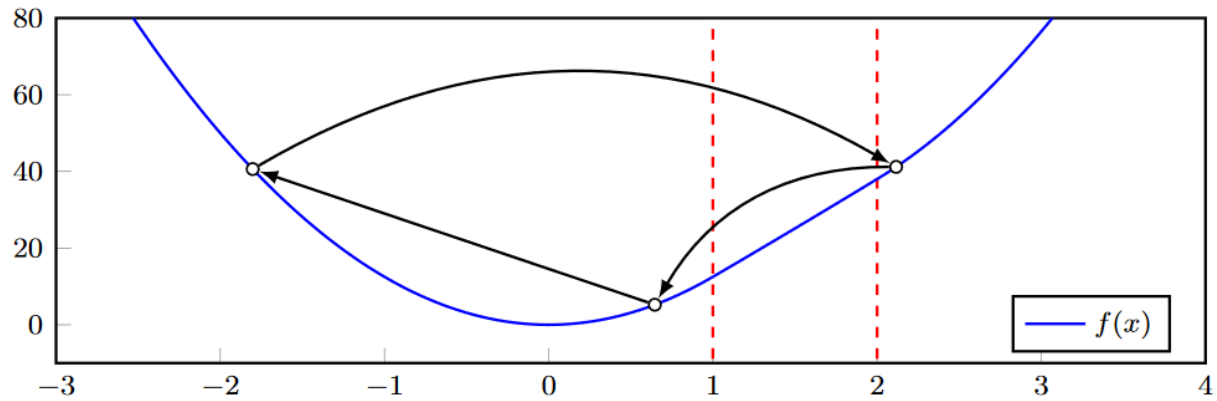
Polyak's momentum failure

[lessard2016analysis lessard2016analysis]

Polyak's momentum algorithm fails to converge in some specific cases, for instance:

$$\nabla f(x) = \begin{cases} 25x \text{ if } x < 1 \\ x + 24 \text{ if } 1 \leqslant x < 2 \\ 25x - 24 \text{ if } x \geqslant 2 \end{cases}$$

In that case, f is μ strongly convex and L-smooth with $(\mu, L) = (1, 25)$. However, iterations given by Polyak's algorithm cycles.



Improving Polyak's momentum

Nesterov Accelerated Gradient Descent

Input: starting point $w^{(0)}$, learning rate $\eta_k > 0$, initial velocity $v^{(0)} = 0$, momentum $\beta_k \in [0, 1]$.
While *not converge* do

- $v^{(k+1)} = w^{(k)} - \eta \nabla f(w^{(k)})$
- $w^{(k+1)} = v^{(k+1)} + \beta_{k+1}(v^{(k+1)} - v^{(k)})$
- $k \leftarrow k + 1$

Return last $w^{(k+1)}$.

Rate of convergence of Nesterov accelerated gradient (NAG)

Theorem

Assume that f is a L-smooth, convex function whose minimum is reached at $w^\star$. Then, if $\beta_{k+1} = k/(k+3)$,

$$f(w^{(k)}) - f(w^\star) \leqslant \frac{2\|w^{(0)} - w^\star\|_2^2}{\eta(k+1)^2}.$$

Theorem

Assume that f is a L-smooth, μ strongly convex function whose minimum is reached at $w^\star$. Then, if

$$\beta_k = \frac{1 - \sqrt{\mu/L}}{1 + \sqrt{\mu/L}},$$

we have

$$f(w^{(k)}) - f(w^\star) \leqslant \frac{\|w^{(0)} - w^\star\|_2^2}{\eta}\left(1 - \sqrt{\frac{\mu}{L}}\right)^k.$$

Optimal bounds

Assumption 1 An iterative method $\mathcal{M}$ generates a sequence of test points $\{w^{(k)}\}$ such that

$$w^{(k)} \in w^{(0)} + \mathrm{Span}(\nabla f(w^{(0)}), \dots, \nabla f(w^{(k-1)})).$$

Theorem

For any k satisfying $1 \leqslant k \leqslant (d-1)/2$, and any $w^{(0)} \in \mathbb{R}^d$, there exists a L-smooth convex function f such that for any first order method $\mathcal{M}$ satisfying Assumption 1, we have

$$f(w^{(k)}) - f(w^\star) \geqslant \frac{3L\|w^{(0)} - w^\star\|_2^2}{32(k+1)^2}.$$

Here, we consider an infinite dimension space $\ell_2 = \{(u_j)_{j=1\dots}, \|u\|_2^2 < \infty\}$.

Theorem

For any $w^{(0)} \in \ell_2$, there exists a L-smooth, μ strongly convex function f such that for any first order method $\mathcal{M}$ satisfying Assumption 1, we have

$$f(w^{(k)}) - f(w^\star) \geqslant \frac{\mu}{2}\left(\frac{1-\sqrt{\mu/L}}{1+\sqrt{\mu/L}}\right)^{2k}\|w^{(0)} - w^\star\|_2^2.$$

Outline

Coordinate Gradient Descent

Another approach: **coordinate descent**

- Received a lot of attention in machine learning and statistics the last 10 years
- It is state-of-the-art on several machine learning problems, when possible
- This is what is used in many `R` packages and for `scikit-learn` Lasso / Elastic-net and `LinearSVC`

Idea. Minimize one coordinate at a time (keeping all others fixed)

Coordinate Gradient Descent

Lemma

Given $f : \mathbb{R}^d \to \mathbb{R}$ convex and smooth if we have

$$f(w + ze_j) \geqslant f(w) \text{ for all } z \in \mathbb{R} \text{ and } j = 1, \dots, d$$

(where $e_j = j$-th canonical vector of $\mathbb{R}^d$) then we have

$$f(w) = \min_{w' \in \mathbb{R}^d} f(w')$$

Proof. $f(w + ze_j) \geqslant f(w)$ for all $z \in \mathbb{R}$ implies that

$$\frac{\partial f}{\partial w^j}(w) = 0$$

which entails $\nabla f(w) = 0$, so that w is a minimum for f convex and smooth

Coordinate Gradient Descent

Exact coordinate descent (CD)

- For $t = 1, \dots,$
- Choose $j \in \{1, \dots, d\}$
- Compute

$$w_j^{t+1} = \underset{z \in \mathbb{R}}{\operatorname{argmin}} f(w_1^t, \dots, w_{j-1}^t, z, w_{j+1}^t, \dots, w_d^t)$$

$$w_{j'}^{t+1} = w_{j'}^t \quad \text{for } j' \neq j$$

Remarks

- Cycling through the coordinates is arbitrary: uniform sampling, pick a permutation and cycle over it every each d iterations
- Only 1D optimization problems to solve, but a lot of them

Coordinate Gradient Descent

Theorem - Warga (1963)

If f is continuously differentiable and strictly convex, then exact coordinate descent converges to a minimum.

Remarks.

- A 1D optimization problem to solve at each iteration: cheap for least-squares, but **can be expensive for other problems**
- Let's solve it approximately, since we have many iterations left
- Replace exact minimization w.r.t. one coordinate by a single gradient step in the 1D problem

Coordinate Gradient Descent

Coordinate gradient descent (CGD)

- For $t = 1, \ldots,$
- Choose $j \in \{1, \ldots, d\}$
- Compute

$$w_j^{t+1} = w_j^t - \eta_j \nabla_{w_j} f(w^t)$$
$$w_{j'}^{t+1} = w_{j'}^t \quad \text{for } j' \neq j$$

Note that

- $\eta_j =$ the step-size for coordinate j, can be taken as $\eta_j = 1/L_j$ where L_j is the Lipchitz constant of

$$f^j(z) = f(w + ze_j) = f(w_1, \ldots, w_{j-1}, z, w_{j+1}, \ldots, w_d)$$

- Cool. Let's try it...
- Wow! Coordinate gradient descent is much faster than GD and AGD! But why ?

Rate of Coordinate Gradient Descent

Theorem - Nesterov (2012)

Assume that f is convex and smooth and that each f^j is L_j-smooth.
Consider a sequence $\{w^t\}$ given by CGD with $\eta_j = 1/L_j$ and coordinates $j_1, j_2, \dots$ chosen at random: i.i.d and uniform distribution in $\{1, \dots, d\}$. Then

$$\mathbb{E} f(w^{t+1}) - f(w^\star) \leqslant \frac{n}{n+t}\Big((1 - \frac{1}{n})(f(w^0) - f(w^\star)) + \frac{1}{2}\|w^0 - w^\star\|_L^2 \Big),$$

with $\|w\|_L^2 = \sum_{j=1}^d L_j w_j^2$.

Remark.

- Bound in expectation, since coordinates are taken at random.
- For cycling cordinates $j = (t \mod d) + 1$ the bound is much worse.

Comparison of Gradient Descent and Coordinate Gradient Descent

- GD achieves ε-precision with

$$\frac{L\|w^0 - w^\star\|_2^2}{2\varepsilon}$$

iterations. A single iteration for GD is $O(nd)$

- CGD achieves ε-precision with

$$\frac{d}{\varepsilon}\left(\left(1 - \frac{1}{n}\right)(f(w^0) - f(w^\star)) + \frac{1}{2}\|w^0 - w^\star\|_L^2\right)$$

iterations. A single iteration for CGD is $O(n)$

- Note that

$$f(w^0) - f(w^\star) \leqslant \frac{L}{2}\|w^0 - w^\star\|_2^2,$$

but typically

$$f(w^0) - f(w^\star) \ll \frac{L}{2}\|w^0 - w^\star\|_2^2.$$

Outline

Outline

ADAGRAD

First order method.

[**duchi2011adaptive duchi2011adaptive**]

ADAptive GRADient algorithm

Input: starting point w^0, learning rate $\eta > 0$, momentum α.
For $t = 1, 2, \ldots$ until *convergence* do

- For all $k = 1, \ldots, d$, apply the step

$$w_k^{t+1} \leftarrow w_k^t - \frac{\eta}{\sqrt{\sum_{\tau=1}^{t} (\nabla f(w^\tau))_k^2}} (\nabla f(w^t))_k$$

Return last w^t

ADAGRAD

Update equation for ADAGRAD

$$w_k^{t+1} \leftarrow w_k^t - \frac{\eta}{\sqrt{\sum_{\tau=1}^{t} (\nabla f(w^\tau))_k^2}} (\nabla f(w^t))_k$$

Pros:

- Different dynamic rates on each coordinate
- Dynamic rates grow as the inverse of the gradient magnitude:
  1. Large/small gradients have small/large learning rates
  2. The dynamic over each dimension tends to be of the same order
  3. Interesting for neural networks in which gradient at different layers can be of different order of magnitude.
- Accumulation of gradients in the denominator act as a decreasing learning rate.

Cons:

- Very sensitive to initial condition: large initial gradients lead to small learning rates.
- Can be fought by increasing the learning rate thus making the algorithm sensitive to the choice of the learning rate.

Outline

Improving upon AdaGrad: AdaDelta

Input: starting point w^0, decay rate $\rho > 0$, constant ε, window parameter $p \in \mathbb{N}^\star$.
Initialization: $\overline{(\nabla f)^2}^0 = 0, \overline{(\Delta x)^2}^0 = 0$

Adadelta algorithm

For $t = 1, 2, \ldots$ until *convergence* do

- For all $j = 1, \ldots, d$,
  1. Compute the accumulated gradient
  $$\overline{(\nabla f)^2}^t = \rho \overline{(\nabla f)^2}^{t-1} + (1-\rho)(\nabla f(w^t))^2$$
  2. Compute the update
  $$w_j^{t+1} = w_j^t - \frac{\sqrt{\overline{(\Delta w)_j^2}^{t-1} + \varepsilon}}{\sqrt{\overline{(\nabla f)_j^2}^t + \varepsilon}} (\nabla f(w^t))_j$$
  3. Compute the aggregated update
  $$\overline{(\Delta w)^2}^t = \rho \overline{(\Delta w)^2}^{t-1} + (1-\rho)(w^{t+1} - w^t)^2$$

Return last w^t

Here $\bar{u}^t = \frac{1}{m} \sum_{p=0}^{m-1} u^{t-p}$ is the mobile mean taken at time t over the last m iterations.

ADADELTA

[zeiler2012adadelta zeiler2012adadelta]

Created as a response to ADAGRAD: less sensitivity to initial parameters.

Second order methods: make use of the Hessian matrix or approximate it.
→ Often costly!

Update equation for adadelta

$$w_j^{t+1} = w_j^t - \frac{\sqrt{\overline{(\Delta w)_j^2}^{t-1} + \varepsilon}}{\sqrt{\overline{(\nabla f)_j^2}^{t} + \varepsilon}} (\nabla f(w^t))_j$$

Interpretation:

- The numerator keeps the size of the previous step in memory and enforce larger steps along directions in which large steps were made.
- The denominator keeps the size of the previous gradients in memory and acts as a decreasing learning rate. Weights are lower than in Adagrad due to the decay rate ρ.

Adadelta

Determining a good learning rate becomes more of an art than science for many problems.

M.D. Zeiler

Compute a dynamic learning rate per dimension based only on the gradient (first order method). Based on a second order method. Fundamental idea comes from studying units. In second order methods,

$$\Delta w \simeq (\nabla^2 f)^{-1} \nabla f.$$

Roughly,

$$\Delta w = \frac{\frac{\partial f}{\partial w}}{\frac{\partial^2 f}{\partial w^2}} \Leftrightarrow \frac{1}{\frac{\partial^2 f}{\partial w^2}} = \frac{\Delta w}{\frac{\partial f}{\partial w}}.$$

See also [schaul2013no schaul2013no]

Outline

RMSprop

Unpublished methode, from the course of Geoff Hinton

`http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`

RMSprop algorithm

Input: starting point w^0, learning rate $\eta > 0$ (default $\eta = 0.001$), decay rate ρ (default $\rho = 0.9$).

For $t = 1, 2, \ldots$ until *convergence* do

- First, compute the accumulated gradient

$$\overline{(\nabla f)^2}^t = \rho \overline{(\nabla f)^2}^{t-1} + (1 - \rho)(\nabla f(w^t))^2$$

- Then, compute the update: for all $k = 1, \ldots, d$,

$$w_k^{t+1} \leftarrow w_k^t - \frac{\eta}{\sqrt{\overline{(\nabla f)^2}^t + \varepsilon}} (\nabla f(w^t))_k$$

Return last w^t

Outline

ADAM: ADAptive Moment estimation

[kingma2014adam kingma2014adam]

General idea: store the estimated first and second moment of the gradient and use them to update the parameters.

Equations - first and second moment

Let m_t be an exponentially decaying average over the past gradients

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla f(w^{(t)})$$

Similarly, let v_t be an exponentially decaying average over the past square gradients

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla f(w^{(t)}))^2.$$

Initialization: $m_0 = v_0 = 0$.

With this initialization, estimates m_t and v_t are biased towards zero in the early steps of the gradient descent.

Final equations

$$\tilde{m}_t = \frac{m_t}{1-\beta_1^t} \quad \tilde{v}_t = \frac{v_t}{1-\beta_2^t}.$$

$$w^{(k+1)} = w^{(k)} - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon}\hat{m}_t.$$

Adam

Adam algorithm

Inputs: stepsize η (default $\eta = 0.001$), exponential decay rates for the moment estimates $\beta_1, \beta_2 \in [0, 1)$ (default: $\beta_1 = 0.9$, $\beta_2 = 0.999$), numeric constant ε (default $\varepsilon = 10^{-8}$).

Initialization: $m_0 = 0$ (Initialization of the first moment vector), $v_0 = 0$ (Initialization of the second moment vector), w_0 (initial vector of parameters).

While *not converge* do

- $k = k + 1$
- Compute first and second moment estimate
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla f(w^{(t)}) \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla f(w^{(t)}))^2.$$
- Compute their respective correction
$$\tilde{m}_t = \frac{m_t}{1 - \beta_1^t} \quad \tilde{v}_t = \frac{v_t}{1 - \beta_2^t}.$$
- Update the parameters accordingly
$$w^{(k+1)} = w^{(k)} - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon}\hat{m}_t.$$

Convergence results: **[kingma2014adam kingma2014adam]**, **[reddi2018convergence reddi2018convergence]**.

Outline

Variation on Adam: Adamax

[kingma2014adam kingma2014adam]

Adamax algorithm

Inputs: stepsize η (default $\eta = 0.002$), exponential decay rates for the moment estimates $\beta_1, \beta_2 \in [0, 1)$ (default: $\beta_1 = 0.9$, $\beta_2 = 0.999$), numeric constant ε (default $\varepsilon = 10^{-8}$).

Initialization: $m_0 = 0$ (Initialization of the first moment vector), $v_0 = 0$ (Initialization of the second moment vector), w_0 (initial vector of parameters).

While *not converge* do

- $k = k + 1$
- Compute first moment estimate and its correction

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla f(w^{(t)}), \qquad \tilde{m}_t = \frac{m_t}{1 - \beta_1^t}$$

- Compute the quantity

$$u_t = \max(\beta_2 u_{t-1}, |\nabla f(w^{(t)})|).$$

- Update the parameters accordingly

$$w^{(k+1)} = w^{(k)} - \frac{\eta}{u_t}\hat{m}_t.$$