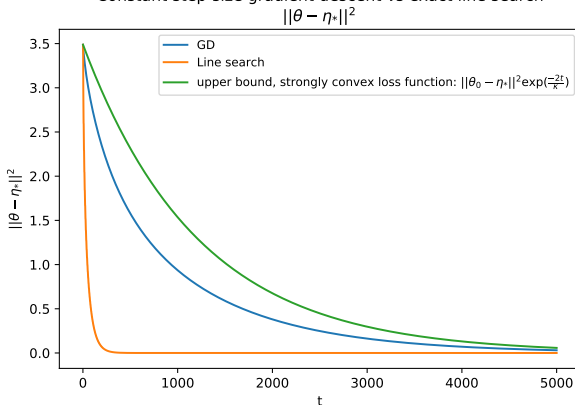


Machine learning I, supervised learning: gradient algorithms

Constant step-size gradient descent vs exact line search



Analytic minimum

What is the minimum of the function

$$f : x \rightarrow (x - 1)^2 + 3.5 \quad (1)$$

And for what value x is it obtained ?

Context

In machine learning, we often encounter problems in high dimension, where closed-form solutions to the **empirical risk minimization** problem are **not** available (e.g. for logistic regression), or where even if they are available, the necessary computation time is too large (OLS).

Context

In machine learning, we often encounter problems in high dimension, where closed-form solutions are not available, or where even if they are available, the necessary computation time is too large.

Example 1 : Computing the OLS estimator requires a matrix inversion, which is $\mathcal{O}(d^3)$.

$$\hat{\theta} = (X^T X)^{-1} X^T y \quad (2)$$

Context

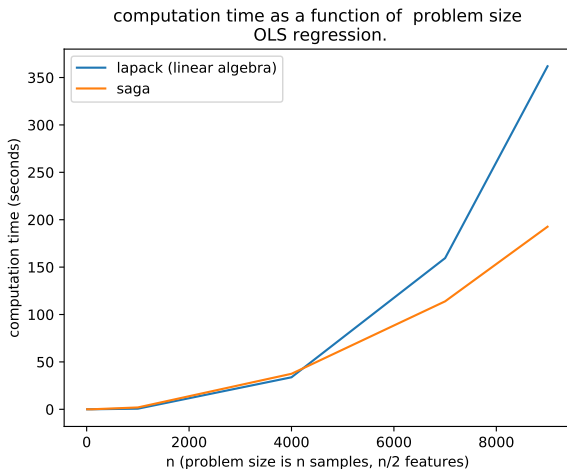
In machine learning, we often encounter problems in high dimension, where closed-form solutions are not available, or where even if they are available, the necessary computation time is too large.

Example 2 : The cancellation of the gradient of the objective function with logistic loss has no closed-form solution.

Context

Instead, we often use **iterative** algorithm such as Gradient descent (GD) or Stochastic gradient descent (SGD). SGD is the standard optimization algorithm for large-scale machine learning.

SGD vs Lapack



Derivation and variation : 1d function

- ▶ In the case a function $f : \mathbb{R} \rightarrow \mathbb{R}$, we can study its variations by computing its derivative f' , **if f is differentiable**
- ▶ If $f'(x) > 0$, the function grows around x .
- ▶ If $f'(x) < 0$, the function decreases around x .
- ▶ If x is a local extremum, $f'(x) = 0$

Derivation and variation

- ▶ In the case a function $f : \mathbb{R} \rightarrow \mathbb{R}$, we can study its variations by computing its derivative f' , **if it exists**
- ▶ If $f'(x) > 0$, the function grows around x .
- ▶ If $f'(x) < 0$, the function decreases around x .
- ▶ If x is a local extremum, $f'(x) = 0$
- ▶ Is the reciprocal true?

One dimensional functions

If $f : \mathbb{R} \mapsto \mathbb{R}$ is differentiable (dérivable) :
(Landau notation)

$$f(a + h) = f(a) + hf'(a) + o(h) \quad (3)$$

Physicist notation :

$$f(a + h) \simeq f(a) + hf'(a) \quad (4)$$

Gradient update

- In one dimension :

$$x \leftarrow x - \gamma f'(x) \quad (5)$$

Multiple-variable functions

If $f : \mathbb{R}^d \mapsto \mathbb{R}$ is differentiable (dérivable) :
(Landau notation)

$$f(a + h) = f(a) + \langle \nabla f(a) | h \rangle + o(h) \quad (6)$$

Physicist notation :

$$f(a + h) \simeq f(a) + \langle \nabla f(a) | h \rangle \quad (7)$$

Gradient descent

$$\theta \leftarrow \theta - \gamma \nabla_f(\theta) \quad (8)$$

γ must be carefully chosen.

- ▶ too large γ : the minimization might not work
- ▶ too small γ : the minimization will be too slow

Gradient descent

Consider a function f that has 2 parameters as inputs.

$$\nabla_f(x, y) = \left(\frac{\delta f}{\delta x}, \frac{\delta f}{\delta y} \right) \quad (9)$$

We want x to **minimise** f . We perform, until some criteria is satisfied :

$$x \leftarrow x - \gamma \nabla_f(x) \quad (10)$$

γ is a small parameter called the learning rate.

Gradient update

- ▶ In one dimension :

$$x \leftarrow x - \gamma f'(x) \quad (11)$$

- ▶ In more dimensions :

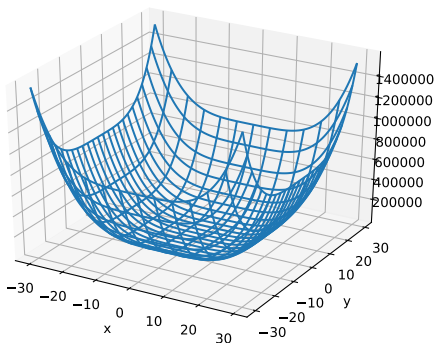
$$w \leftarrow w - \gamma \nabla_w(f) \quad (12)$$

- ▶ γ is the **learning rate**.

Gradient

Exercise 1 : Implementing the gradient algorithm

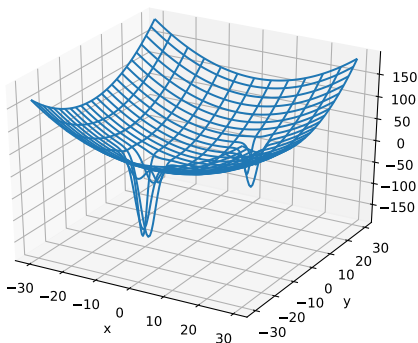
We will use the algorithm on two functions.



Gradient

Exercise 1 : Implementing the gradient algorithm

We will use the algorithm on two functions.

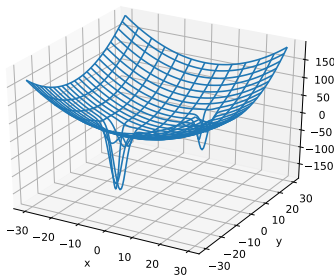


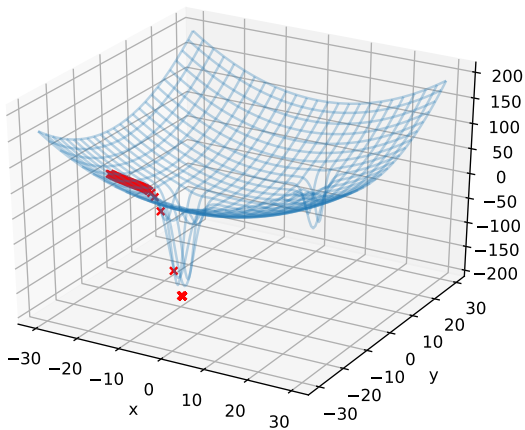
Gradient

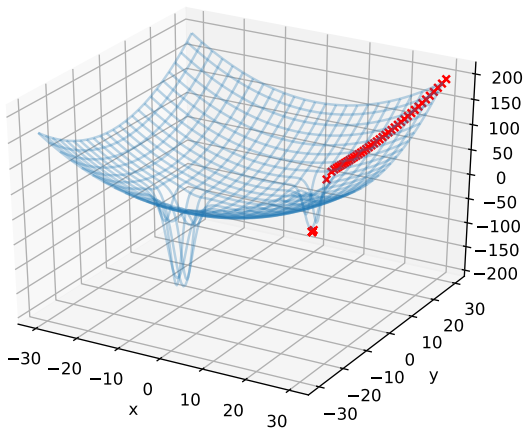
Exercise 1 : Implementing the gradient algorithm

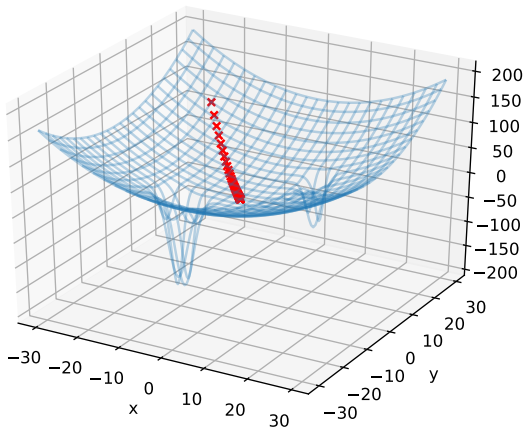
`cd ./gradient` and use the files `gradient_algo_1.py` and `gradient_algo_2.py` in order to implement the algorithm to find minima.

Experiment with all the parameters that you consider relevant (several are) to assess their impact on the algorithm.









Convergence speed

For some problems, it is possible to have guarantees on the convergence speed of gradient descent. The results will depend on the following properties of the objective function :

- ▶ convexity or strong convexity
- ▶ smoothness (Lipshitz-continuous gradients) or non-smoothness
- ▶ condition number

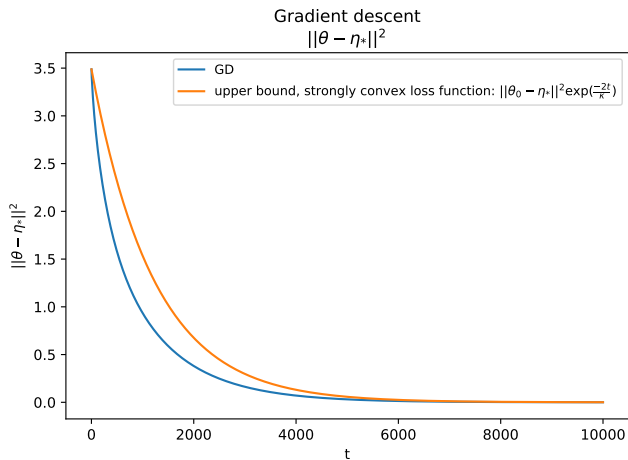
Smoothness (Example of theoretical criterion)

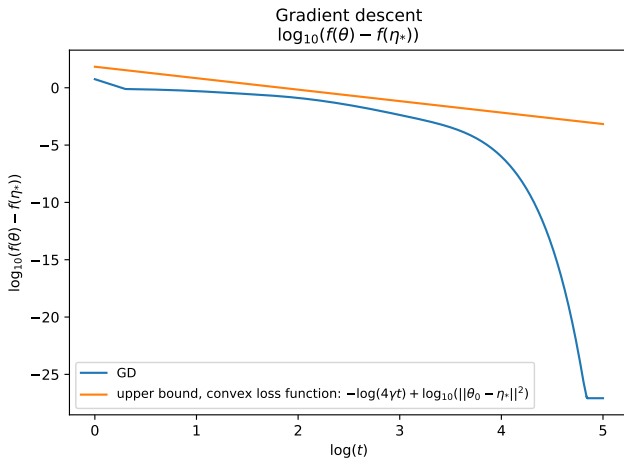
Définition

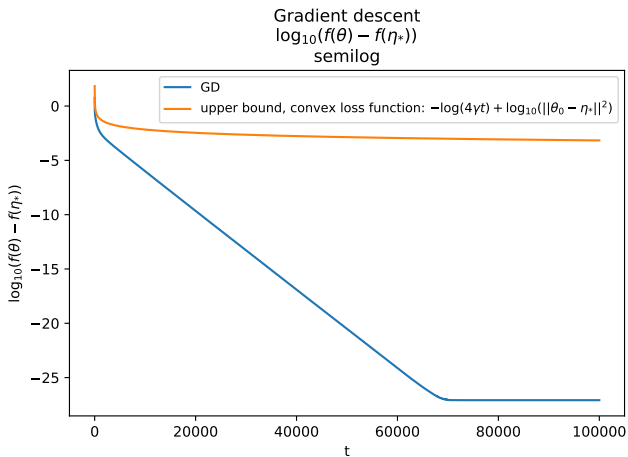
Smoothness

A differentiable function f with real values is said L -smooth if and only if

$$\forall x, y \in \mathbb{R}^d, |f(y) - f(x) - \nabla_x f(y - x)| \leq \frac{L}{2} \|y - x\|^2$$

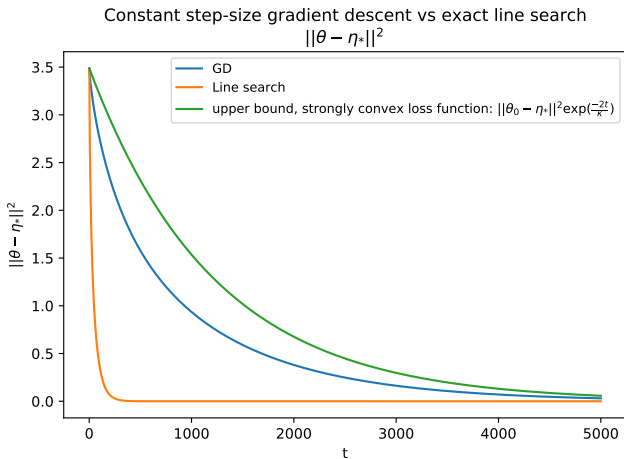






Extensions

- ▶ Line search
- ▶ Nesterov acceleration (optimal rates among algorithms that linearly combine gradients)



Stochastic gradient descent

In machine learning, we often consider an objective function of the form

$$f(\theta) = \frac{1}{n} \sum_{i=1}^n l(y_i, f_{\theta}(x_i)) + \Omega(\theta) \quad (13)$$

Batch gradient

In machine learning, we often consider an objective function of the form

$$f(\theta) = \frac{1}{n} \sum_{i=1}^n l(y_i, f_{\theta}(x_i)) + \Omega(\theta) \quad (14)$$

Computing the gradient of f requires at least n calculations, and each calculation also has a complexity that depends on the dimension d . When n and d are large, this can be quite slow.

Stochastic gradient descent

We consider an objective function of the form

$$f(\theta) = \frac{1}{n} \sum_{i=1}^n l(y_i, f_{\theta}(x_i)) + \Omega(\theta) \quad (15)$$

Instead of computing the **batch gradient** $\nabla_{\theta} f$, we will compute :

$$u(i, \theta) = \nabla_{\theta} [l(y_i, f_{\theta}(x_i)) + \Omega(\theta)] \quad (16)$$

for a randomly sampled $i \in [1, n]$.

$u(i, \theta)$ is an **estimation** of the full (batch) gradient $\nabla_{\theta} f$.

Tradeoff

By replacing the computation of the full gradient (GD) by this estimation (SGD) :

- ▶ we reduce the computation time (divide it by n)
- ▶ but reduce precision of the optimization

To summarize : if n is really large, SGD is often better.

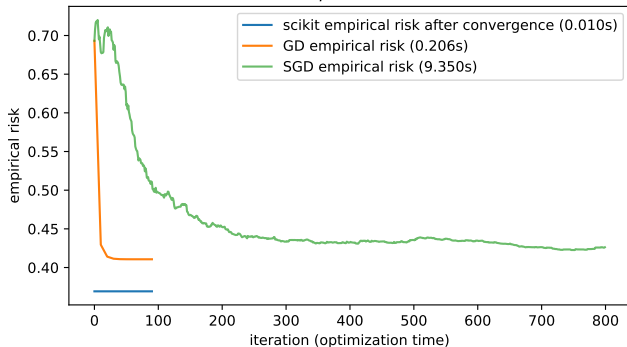
Logistic regression: loss with GD, SGD, scikit

$n=10000$, $d=80$

$\gamma_{SGD0} = 0.05$, schedule: decreasing 1

$\gamma_{GD} = 0.5$

$\mu = 0.1$



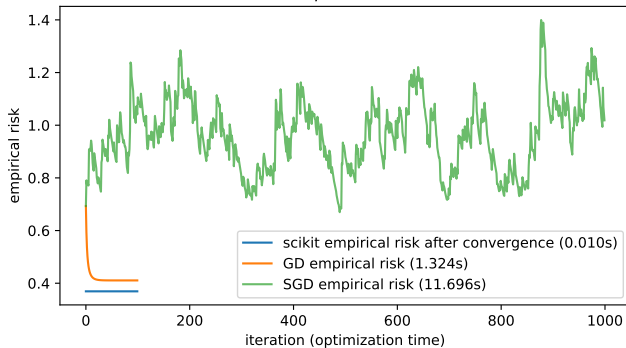
Logistic regression: loss with GD, SGD, scikit

$n=10000$, $d=80$

$\gamma_{SGD0} = 0.2$, schedule: constant

$\gamma_{GD} = 0.5$

$\mu = 0.1$



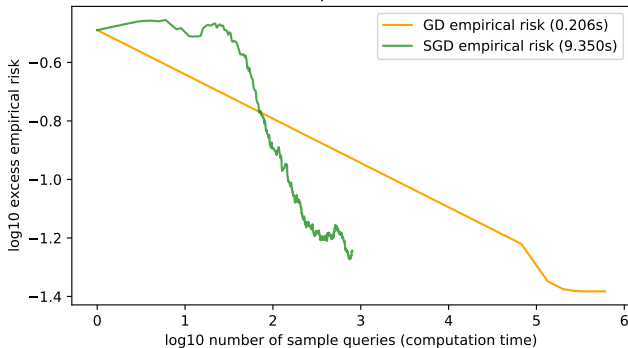
Logistic regression: log excess loss with GD, SGD, scikit

$n=10000$, $d=80$

$\gamma_{SGD0} = 0.05$, schedule: decreasing 1

$\gamma_{GD} = 0.5$

$\mu = 0.1$



Comparison (advanced)

The ridge regression problem is smooth and strongly convex.

- ▶ GD has a convergence rate of $\mathcal{O}(\exp(-\frac{t}{\kappa}))$. To get an error of ϵ , we must have $t = \mathcal{O}(\kappa \log \frac{1}{\epsilon})$. Since each iteration requires $\mathcal{O}(nd)$ computations, the computation time will be $\mathcal{O}(\kappa nd \log \frac{1}{\epsilon})$.
- ▶ SGD has a convergence rate of $\mathcal{O}(\frac{\kappa}{t})$. To get an error of ϵ , we must have $t = \mathcal{O}(\frac{\kappa}{\epsilon})$. Since each iteration is $\mathcal{O}(d)$, we have a computation time of $\mathcal{O}(\frac{\kappa d}{\epsilon})$.

Comparison (advanced)

As a consequence :

- ▶ When n is large and ϵ not too small, GD will need more computation time to reach error ϵ . An order of magnitude can be obtained by studying the value ϵ^* such that

$$\kappa n d \log \frac{1}{\epsilon^*} = \frac{\kappa d}{\epsilon^*}$$

Which translates to

$$\epsilon^* \log \epsilon^* = -\frac{1}{n}$$

- ▶ When $\epsilon \rightarrow 0$, GD becomes faster than SGD to reach this precision.

Conclusion

For lower precision and large n , SGD is a preferable.

In machine learning, due to the estimation error that is $\mathcal{O}(\frac{1}{\sqrt{n}})$, a very high precision is often not needed

Extensions of SGD

See also :

- ▶ Variance reduction methods (SAG, SAGA)