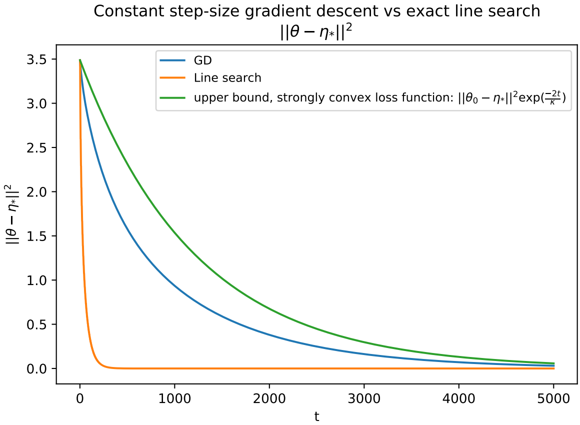
Machine learning I, supervised learning: gradient algorithms



Context

In machine learning, we often encounter problems in high dimension, where closed-form solutions are not avaible (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 avaible, 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 $\mathcal{O}(d^3)$.

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

Context

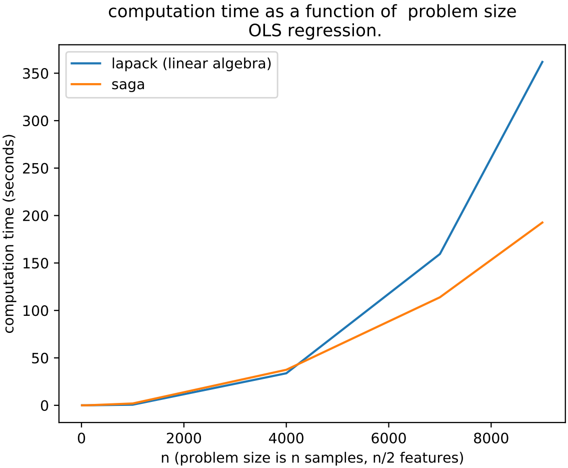
In machine learning, we often encounter problems in high dimension, where closed-form solutions are not avaible, or where even if they are available, the necessary computation time is too large.

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



Gradient descent

We want to minimize a function f defined over $\mathbb{R}^d$.

$$\theta \leftarrow \theta - \gamma \nabla_f(\theta) \tag{2}$$

Gradient descent

- In the case a function $f : \mathbb{R} \to \mathbb{R}$, we can study its variations by computing its derivative f', **if it exists**

Gradient descent

- In the case a function $f : \mathbb{R} \to \mathbb{R}$, 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, $f'(x) = 0$

Gradient descent

- In the case a function $f : \mathbb{R} \to \mathbb{R}$, 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 ?

Derivation

- We can use the derivative to look for a minimum value for the function
- Example with analytic solution

Analytic minimum

What is the minimum of the function

$$f : x \rightarrow (x-1)^2 + 3.5 \tag{3}$$

And for what value x is it obtained ?

Gradient update

- In one dimension :

$$x \leftarrow x - \gamma f'(x) \tag{4}$$

Gradient

- The **gradient** is similar to a derivative but in the case of a function with several inputs, such as our loss $l(w1, w2)$.
- Then we store the **partial derivative** with respect to each input in a **vector** called the gradient.

Gradient descent

Consider a function f that has 2 parameters as inputs.

$$\nabla_f(x, y) = (\frac{\delta f}{\delta x}, \frac{\delta f}{\delta y}) \tag{5}$$

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

$$x \leftarrow x - \gamma \nabla_f(x) \tag{6}$$

γ is a small parameter called the learning rate.

Gradient update

- In one dimension :

$$x \leftarrow x - \gamma f'(x) \tag{7}$$

- In more dimensions :

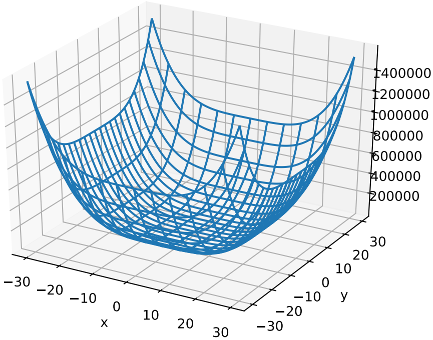
$$w \leftarrow w - \gamma \nabla_w (f) \tag{8}$$

- γ is the **learning rate.**

Gradient

Exercice 1 : Implementing the gradient algorithm

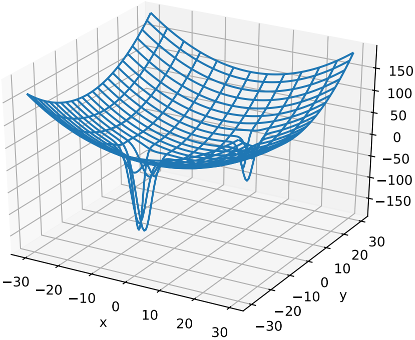
We will use the algorithm on two functions.



Gradient

Exercice 1 : Implementing the gradient algorithm

We will use the algorithm on two functions.

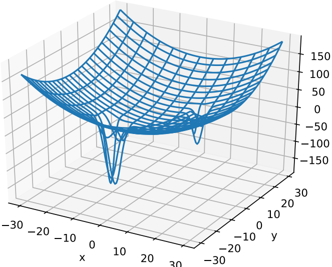


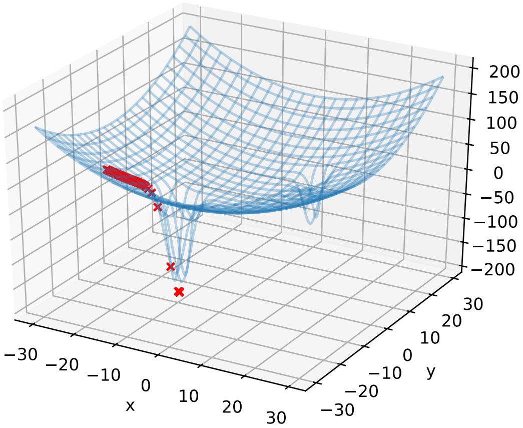
Gradient

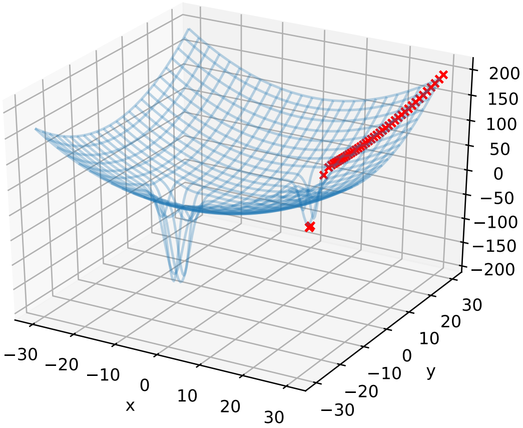
Exercice 1 : **Implementing the gradient algorithm**

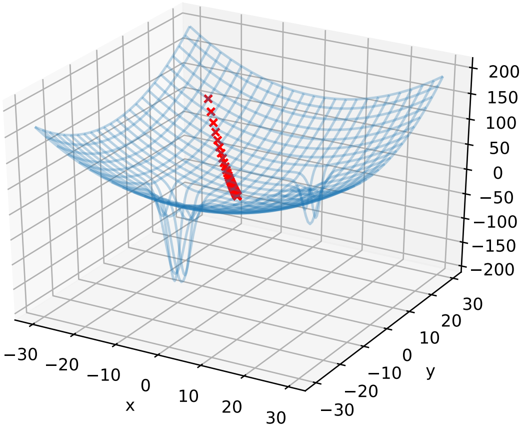
cd ./gradient and use the files **gradient_algo_1.py** and **gradient_algo_2.py** in order to implement the algorithm to find **minima**.

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




200
150
100
50
0
−50
−100
−150
−200
−30
−20
−10
0
10
20
30
x
30
20
10
0
−10
−20
−30
y


200
150
100
50
0
−50
−100
−150
−200
−30
−20
−10
0
10
20
30
x
30
20
10
0
−10
−20
−30
y


200
150
100
50
0
−50
−100
−150
−200
−30
−20
−10
0
10
20
30
x
30
20
10
0
−10
−20
−30
y

Convergence speed

For some problems, it is possible to have garantees 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
- confition number

Smoothness

Définition

Smoothness

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

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

Smoothness

Lemme

f is L-smooth if and only if it has L-Lipshitz continuous gradients.

Définition

L-Lipschitz continuous gradients

f has L-Lipschitz continuous gradients if $\forall x, y \in \mathbb{R}^d$,

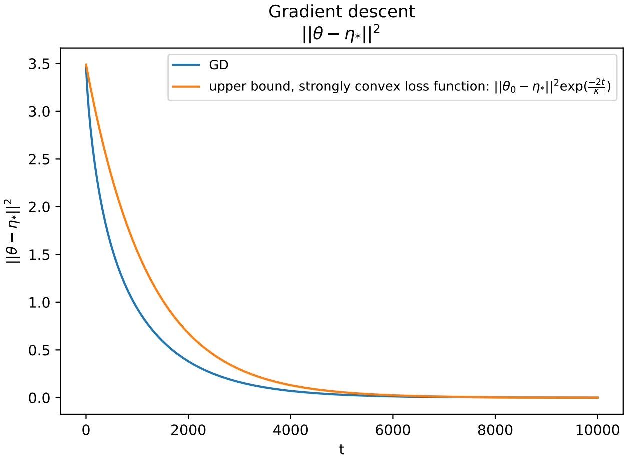
$$||\nabla_x f - \nabla_y f|| \leq L||x - y||$$

Smooth, strongly convex functions

Théorème

Convergence of GD for a strongly convex function
Let $f : \mathbb{R}^d \Rightarrow \mathbb{R}$ *be a* μ*-strongly convex function with L-Lipshitz continuous gradients. Let* x^* *be the global minimum of* f *(which we know exists since* f *is strongly convex),* $x_0 \in \mathbb{R}$, $T \in \mathbb{N}$.
With constant step size $\gamma_t = \frac{1}{L}$, *we have*

$$\begin{aligned} f(x_t) - f(x^*) &\leq (1 - \frac{1}{\kappa})^t (f(x_0) - f(x^*)) \\ &\leq \exp(-\frac{t}{\kappa})(f(x_0) - f(x^*)) \end{aligned} \tag{9}$$

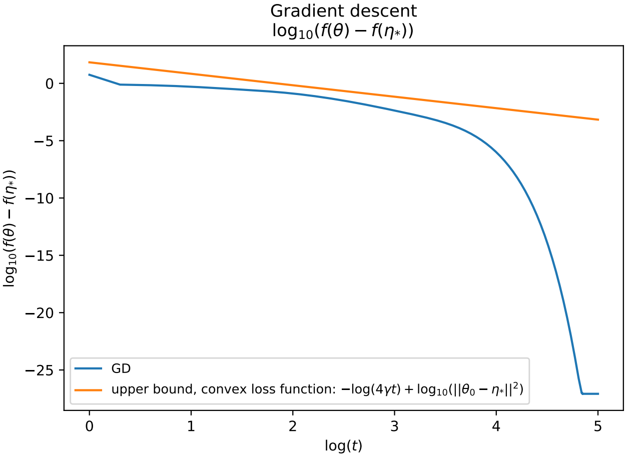

Gradient descent
$||\theta - \eta_*||^2$
GD
upper bound, strongly convex loss function: $||\theta_0 - \eta_*||^2 \exp(\frac{-2t}{\kappa})$
$||\theta - \eta_*||^2$
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2000
4000
6000
8000
10000
t

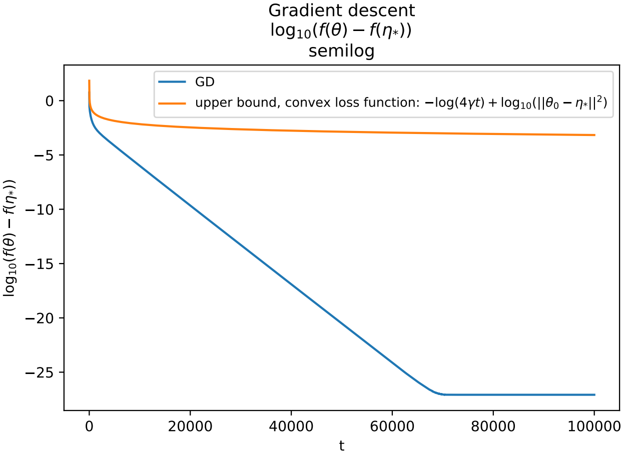
Smooth, convex function (not necessary strongly convex)

Théorème

Convergence of GD for a smooth convex function
Let $f : \mathbb{R}^d \to \mathbb{R}$, with global minimiser x^*. With constant step-size $\gamma_t = \frac{1}{L}$, the iterates x_t of GD satisfy :

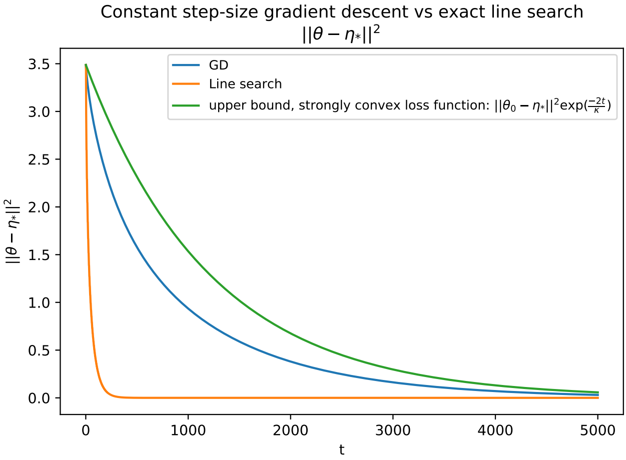
$$f(x_t) - f(\eta^*) \leq \frac{L}{2t} ||x_0 - \eta^*||^2$$


Gradient descent
$\log_{10}(f(\theta) - f(\eta_*))$
$\log_{10}(f(\theta) - f(\eta_*))$
0
−5
−10
−15
−20
−25
0
1
2
3
4
5
$\log(t)$
GD
upper bound, convex loss function: $-\log(4\gamma t) + \log_{10}(||\theta_0 - \eta_*||^2)$


Gradient descent
$\log_{10}(f(\theta) - f(\eta_*))$
semilog
GD
upper bound, convex loss function: $-\log(4\gamma t) + \log_{10}(||\theta_0 - \eta_*||^2)$
$\log_{10}(f(\theta) - f(\eta_*))$
0
−5
−10
−15
−20
−25
0
20000
40000
60000
80000
100000
t

Extensions

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


Constant step-size gradient descent vs exact line search
$||\theta - \eta_*||^2$
GD
Line search
upper bound, strongly convex loss function: $||\theta_0 - \eta_*||^2 \exp(\frac{-2t}{\kappa})$
$||\theta - \eta_*||^2$
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1000
2000
3000
4000
5000
t

Comparison with Newton method

Newton's method minimizes the second-order Taylor expansion around θ_{t-1} in order to compute θ_t.
The convergence of Newton method is faster in the number of iterations, but each iteration is expensive : $\mathcal{O}(d^3)$ since it requires to solve a linear system.

$$C||\theta_t - \theta_*|| \leq (C||\theta_t - \theta_*||)^2 \tag{10}$$

As in machine learning we often have an estimation error $\mathcal{O}(\frac{1}{\sqrt{n}})$, the tradeoff is not in favor of Newton's method.

Stochastic gradient descent

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

$$f(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(y_i, f_\theta(x_i)) + \Omega(\theta) \tag{11}$$

Batch gradient

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

$$f(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(y_i, f_\theta(x_i)) + \Omega(\theta) \tag{12}$$

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

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

Instead of computing the **batch gradient** $\nabla_\theta F$, we will compute **unbiased stochastic estimations** of the gradient, $g_t(\theta_{t-1})$. For all t,

$$E[g_t(\theta_{t-1})] = \nabla_{\theta_{t-1}} F \tag{14}$$

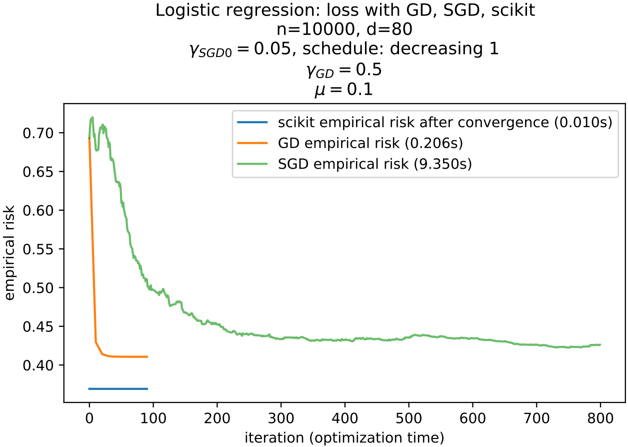
SGD update

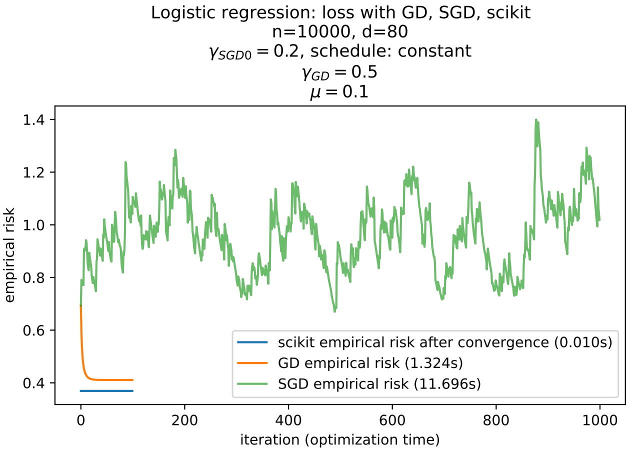
The SGD update reads

$$\theta_t = \theta_{t-1} - \gamma_t g_t(\theta_{t-1})$$

Empirical risk minimization : at each time step we choose uniformly $i(t) \in \{1, \ldots, n\}$ and

$$g_t = \nabla_\theta \Big(l(y_{i(t)}, f_\theta(x_{i(t)})) + \Omega(\theta) \Big) \tag{15}$$


Logistic regression: loss with GD, SGD, scikit
n=10000, d=80
$\gamma_{SGD0} = 0.05$, schedule: decreasing 1
$\gamma_{GD} = 0.5$
$\mu = 0.1$
scikit empirical risk after convergence (0.010s)
GD empirical risk (0.206s)
SGD empirical risk (9.350s)
empirical risk
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0
100
200
300
400
500
600
700
800
iteration (optimization time)


Logistic regression: loss with GD, SGD, scikit
n=10000, d=80
$\gamma_{SGD0} = 0.2$, schedule: constant
$\gamma_{GD} = 0.5$
$\mu = 0.1$
empirical risk
1.4
1.2
1.0
0.8
0.6
0.4
0
200
400
600
800
1000
iteration (optimization time)
scikit empirical risk after convergence (0.010s)
GD empirical risk (1.324s)
SGD empirical risk (11.696s)

Learning rate for SGD

The learning rate is more tricky to set for SGD.

SGD as an estimation of GD

Given θ_{t-1}, we have that

$$\begin{aligned} E\left[\theta_t\right] &= E\left[\theta_{t-1} - \gamma_t g_t(\theta_{t-1})\right] \\ &= \theta_{t-1} - \gamma_t E\left[g_t(\theta_{t-1})\right] \\ &= \theta_{t-1} - \gamma_t \nabla_{\theta_{t-1}} F \end{aligned}$$

SGD as an estimation of GD

Given θ_{t-1}, we have that

$$\begin{aligned} E\left[\theta_t\right] &= E\left[\theta_{t-1} - \gamma_t g_t(\theta_{t-1})\right] \\ &= \theta_{t-1} - \gamma_t E\left[g_t(\theta_{t-1})\right] \\ &= \theta_{t-1} - \gamma_t \nabla_{\theta_{t-1}} F \end{aligned}$$

In expectation, SGD behaves as GD.

Convergence result

For SGD, the convergence results and proofs are more abstract. In the general case :

- Either we have results on expected values, such as that of $||\theta_t - \theta^*||$.
- Or convergence garantees on **averages of the iterates.**

Algorithmic complexities

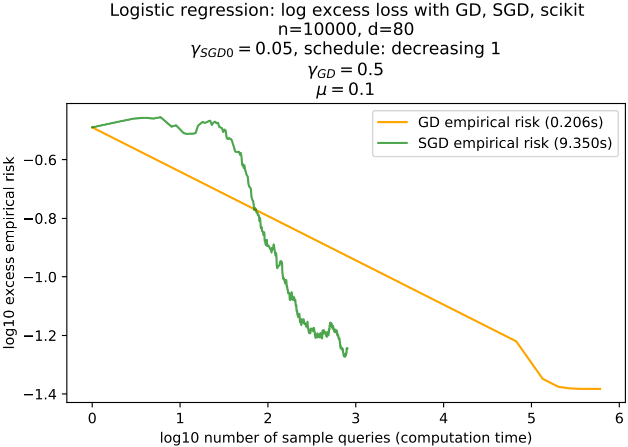
Exercice 2 : We consider a least squares problem. Compute the computational complexities of

- an iteration of GD.
- an iteration of SGD

Algorithmic complexities

Exercice 3 : We consider a least squares problem. Compute the computational complexities of

- an iteration of GD : $\mathcal{O}(nd)$.
- an iteration of SGD : $\mathcal{O}(d)$.


Logistic regression: log excess loss with GD, SGD, scikit
n=10000, d=80
$\gamma_{SGD0} = 0.05$, schedule: decreasing 1
$\gamma_{GD} = 0.5$
$\mu = 0.1$
GD empirical risk (0.206s)
SGD empirical risk (9.350s)
log10 excess empirical risk
−0.6
−0.8
−1.0
−1.2
−1.4
0
1
2
3
4
5
6
log10 number of sample queries (computation time)

Comparison

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

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 $\epsilon \to 0$, 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)