FTML practical session 4: 2023/04/07

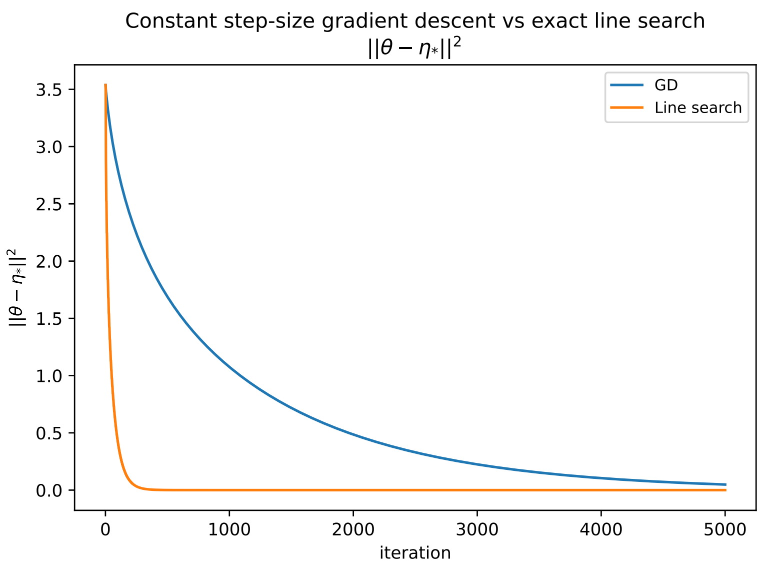

TABLE DES MATIÈRES

INTRODUCTION

The three parts of the session are independent and in a dedicated folder in the repo.

1 NUMBER OF CLUSTERS FOR K-MEANS

— folder : **k_means/**

— template file : **k_means_heuristics.py**

A company has gathered data about its customers and would like to identify similar clients, in order to propose relevant products to new clients, based on their features. This can be represented as a clustering problem. The data are stored in **data.npy**. They are 4 dimensional.

We would like to study methods to automatically find a relevant number of clusters in this dataset for the K-means algorithm.

1.1 Visual method

Is there a direct, visual way to have an idea of a relevant number of clusters for these 4-dimensional data ?

1.2 Algorithmic heuristics

We would like to see if algorithmic heuristics are consistent with the previous results.

Experiment with the following heuristics, metrics and tools in order to obtain a suggested number of clusters :

— possible metrics to monitor :

  — inertia :
    `https://scikit-learn.org/stable/modules/clustering.html#k-means`

  — silhouette score
    `https://scikit-learn.org/stable/modules/clustering.html#silhouette-coefficient`

  — Calinksi-Harabasz score :
    `https://scikit-learn.org/stable/modules/clustering.html#calinski-harabasz-index`

— Knee detection tools :

  — `https://github.com/arvkevi/kneed`

  — `https://www.scikit-yb.org/en/latest/api/cluster/elbow.html`

1.3 Chaning the data

You can generate the data again in order to study the behavior of the previous heuristics on a dataset with different statistical properties.

2 CONVERGENCE OF SGD FOR A LOGISTIC REGRESSION

— folder : **lr/**

— template files to fix or edit : **lr.py, learning_rate_schedules.py**

We have seen in the previous session that with a constant step size, SGD does not always converge when this step is too large. In this exercise we will study some **learning rate schedules**. Namely, the learning rate will not be constant throughout the optimization, and will decrease. The question is then naturally : how fast should we decrease the learning rate ?

A classical strategy is to use a decreasing learning rate of the form

$$\gamma_t = \frac{\gamma_0}{1 + \frac{t}{t_0}} \tag{1}$$

where γ_0 is the initial value of γ and t_0 is a parameter controling the speed of the evolution of γ_t as a function of t. We will call this schedule **"decreasing 1"** in the practical session.

Another possibility is the following :

$$\gamma_t = \frac{\gamma_0}{1 + \sqrt{\frac{t}{t_0}}} \tag{2}$$

With this schedule, γ decreases more slowly than with 1, and γ_t is thus larger. We will call it **"decreasing 2"** in the practical session.

Experiment with these schedules, and with the constants, in order to find values of γ_0 sich that with a decreasing learning rate of the previous form, SGD converges to a good solution, but does not with a constant learning rate.

2.1 Libraries

Many other learning rate schedules exist. For instance, in Pytorch, they are called schedulers :

https://pytorch.org/docs/stable/optim.html

The choice of the scheduler itsself may be seen as a hyperparameter of the optimization problem.

Also note that the convergence speed of SGD is often optimized in these lower level implemented libraries, as compared to a pure python implementation where we would use a for loop.

3 LINE SEARCH FOR LEAST SQUARES

— folder : **line_search/**

— template files to fix or edit : **gd_line_search.py, optimal_gamma.py**

We come back to a least squares problem, for which we will illustrate another type of learning rate optimization for gradient descent. We recall the setting here :

— $\mathcal{X} = \mathbb{R}^d$

— $\mathcal{Y} = \mathbb{R}$

— design matrix : X

— vector of outputs : $y \in \mathbb{R}^n$.

We want to minimize the function f representing the empirical risk :

$$f(\theta) = \frac{1}{2n}||X\theta - y||^2 \tag{3}$$

3.1 Setting

We recall that the gradient and the Hessian write :

$$\begin{aligned} \nabla_\theta f(\theta) &= \frac{1}{n} X^T (X\theta - y) \\ &= H\theta - \frac{1}{n} X^T y \end{aligned} \tag{4}$$

$$H = \frac{1}{n} X^T X \tag{5}$$

We note the gradient update

$$\theta_{t+1} = \theta_t - \gamma \nabla_{\theta_t} f(\theta) \tag{6}$$

We consider the minimizers of f, noted η^*. They all verify

$$\nabla_\theta f(\eta^*) = 0 \tag{7}$$

Which means that

$$H\eta* = \frac{1}{n} X^T y \tag{8}$$

If H is not invertible, there might be several minimizes. However, if f is strongly convex, then H is invertible and η^* is unique. We will consider such a case here. In general, H is a positive semi-definite matrix, and here we hence consider the case where it is definite positive.

3.2 A nested optimization problem

Considering a fixed iteration step θ_t, we note

$$\alpha(\gamma) = \theta_t - \gamma \nabla_\theta f(\theta_t) \tag{9}$$

The **exact line seach** method attempts to find the optimal step γ^*, at each iteration. This means, given the position θ_t, the parameter γ that minimizes the function defined by

$$\begin{aligned} g(\gamma) &= f(\theta_t - \gamma \nabla_\theta f(\theta_t)) \\ &= f(\alpha(\gamma)) \end{aligned} \tag{10}$$

Is $g : \mathbb{R} \to \mathbb{R}$ a convex function ?

Find the value γ^* that minimizes g for a given θ_t, and implement both a constant step GD and a GD where γ is set optimally thanks to this method (exact line search). Compare the convergence speeds by measuring the distance between the iterate and η^* at each iteration.

You should observe a result like figures 1 and 2.

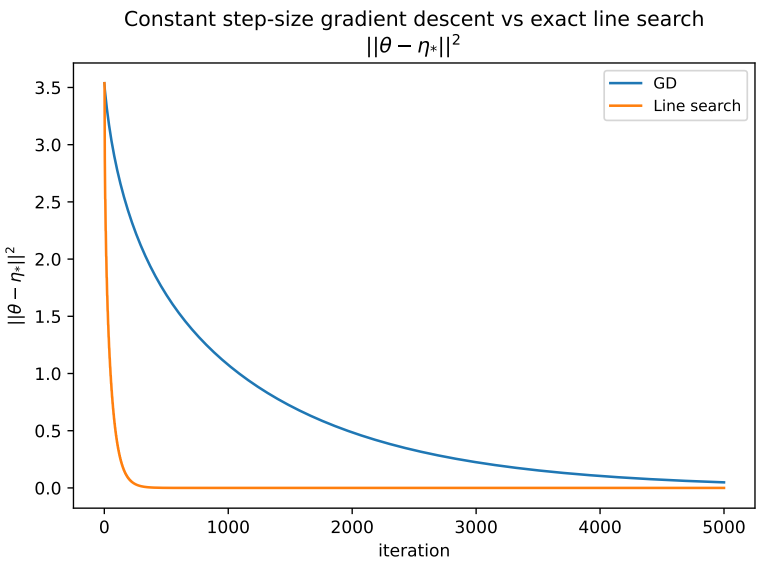

Figure 1 – Convergence to the global minimizer.

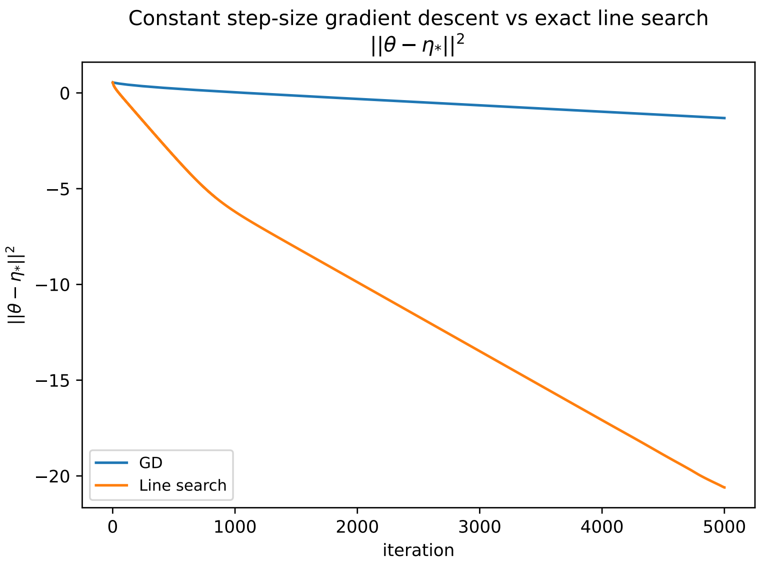

Figure 2 – Convergence to the global minimizer in log scale.