Optimization Project - IASD

Christophe Ly

January 2022

Contents

1 Introduction

This project has been made for the Optimization for Machine Learning course from the M2 IASD cursus at Université Paris-Dauphine. It aims to study several approaches seen during the lectures and the lab sessions to solve an optimization problem. For this project, I chose to work on a linear regression problem. In a first part, I will introduce the dataset I used, and what the minimization problem of this project is. Then, I will present the studied methods to solve this problem such as *Batch Gradient Descent*, *Stochastic Gradient Descent* and their variants. I will also explore various values for the hyper-parameters to check their impact on the convergence rates of the learning curves.

2 Project settings

2.1 Introduction of the dataset

For this project, I decided to work on a dataset called *100,000 UK Used Car Data set* from Kaggle. This dataset gathers 100 000 listings of used cars which have been separated in multiple files depending on their manufacturers. Since, we only need a medium size dataset (around 10k points), I have decided to use only the files *merc.csv* which contains 13119 listing of used cars from the manufacturer Mercedes-Benz. After a preprocessing of the data to keep the relevant features we will use, we got the following layout.

	year	price	mileage	tax	mpg	engineSize
0	2005	5200	63000	325	32.1	1.8
1	2017	34948	27000	20	61.4	2.1
2	2016	49948	6200	555	28.0	5.5
3	2016	61948	16000	325	30.4	4.0
4	2016	73948	4000	325	30.1	4.0

Figure 1: Head of the dataframe of merc.csv

So, for each car we have data about their year of construction, their price, their mileage, the associated road tax, the consumption of the car with the miles per gallon (mpg), and the engine size. Then, we can have a look at the correlation matrix :

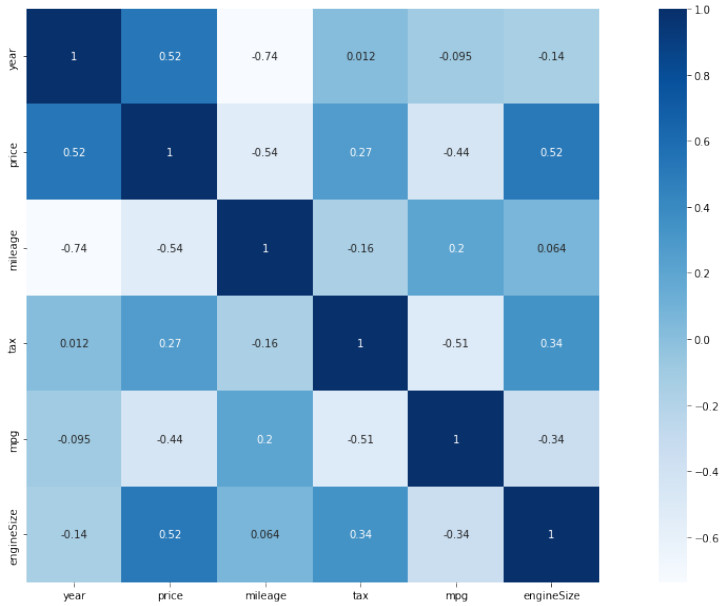

Figure 2: Correlation Matrix of the dataset

We can see that the price feature shows a decent correlation with every other features of the data, with $r^2 \approx 0.5$ everywhere. We can also plot the distribution of the used cars depending on the price and the other features to have an other view of the potential correlation.

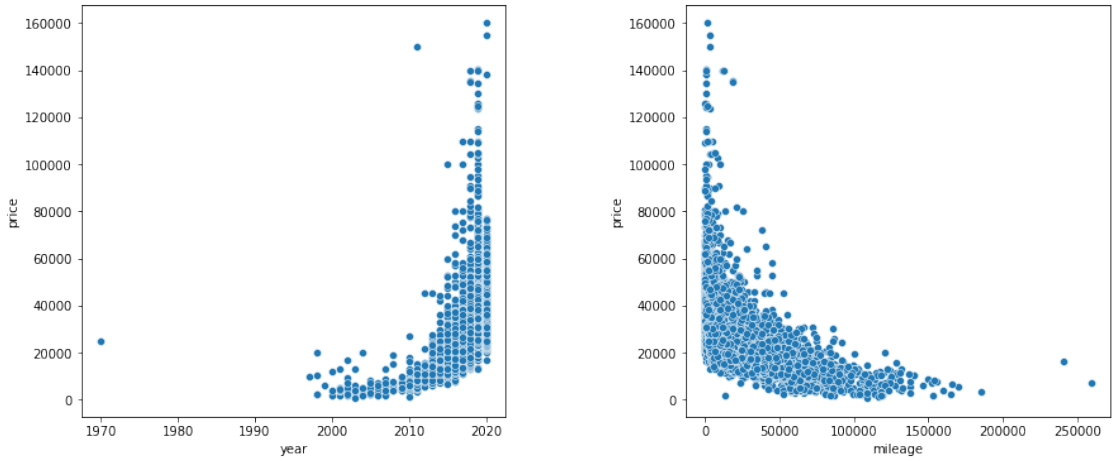

Figure 3: Repartition of the cars depending on their price and the year of construction / the mileage

As we can see on figure 3, we can see that there are more expensive used car among the recent ones, and among the ones with a low mileage. After preprocessing our data, we have splitted the dataset in train and test sets with a split of 0.33 for test set.

2.2 Presentation of the problem

Through this project and this report, we will work on a linear regression problem. The goal is to predict the price of an used car by using the 5 remaining features of the dataset (year, mileage, tax, mpg, and

enginesize). So, our minimization problem is equivalent to :

$$\underset{w\in\mathbb{R}^d}{\text{minimize}} f(w) = \frac{1}{2n}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2 = \frac{1}{2n}\left(\sum_{i=1}^{n}(x_i^\top w - y_i)^2 + \frac{\lambda}{2}\|w\|^2\right) \tag{1}$$

Note that X is a matrix of shape ($n = 13119$, $d = 5$) which contains each listing of used car and their 5 features, and Y is of shape (n, 1), and it contains the associated price. We have also introduced the Ridge regularization term in the problem statement, since we will study regularization later.

3 Studied methods to solve the problem

3.1 Batch Gradient Descent and Stochastic Gradient Descent

3.1.1 Introduction

We began the project with the implementation of classical Batch Gradient Descent algorithm. This method consists in looking for the minimizer of f in the direction of the steepest descent (so, at the opposite of the gradient of the function). Hence, given a $W_0 \in \mathbb{R}^5$, a stepsize / learning rate $\alpha_k \in \mathbb{R}$, we need to compute :

$$w_{k+1} = w_k - \alpha_k \nabla f(w_k) \tag{2}$$

According to the lectures, this methods works especially in the context of a convex function with a convergence rate of $\mathcal{O}(\frac{1}{K})$, or of a strong convex function with a convergence rate of $\mathcal{O}((1 - \frac{\mu}{L})^K)$ for a constant stepsize $\alpha_k = \frac{1}{L}$ with L the lipschitz constant.

A variant of the Gradient Descent is the Stochastic Gradient Descent, which consists in computing the same iteration as the Batch Gradient Descent one, but using partial gradient on a randomly drawn sample of X. Hence, an iteration of a Stochastic Gradient Descent consists in computing :

$$w_{k+1} = w_k - \alpha_k \nabla f_{i_k}(w_k) \tag{3}$$

SGD appears to be way cheaper than the classical full BGD, and to converge to a way better solution. Moreover, multiple samples can be randomly drawn to compute. We call this a batch and its size depends on a hyperparameter called the batch size. So, this is the more general version of Batch Gradient Descent :

$$w_{k+1} = w_k - \frac{\alpha_k}{|S_k|}\sum_{i\in S_k} \nabla f_i(w_k) \tag{4}$$

So, the classical BGD we introduced before picks a batch size equal to the whole X, while the SGD runs the BGD with a batch size equal to 1. Usually, the batch size equals to 1 is the most popular. The full gradient descent can be to heavy to compute, but oscillates way less than SGD.

3.1.2 Learning Rates exploration

In our problem, we tried to explore the impact of the stepsize on the convergence rates of our learning curves for both constant and decreasing stepsize. First, we ran our Batch Gradient Descent for 5000 epochs on several constant learning rates at Fig.4.

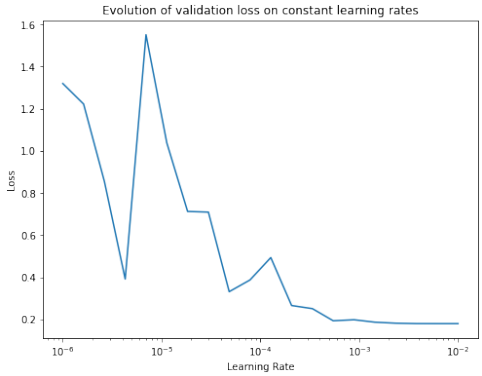

Figure 4: Evolution of the validation loss for constant learning rates

We can see that the optimal learning rate seems to be achieved for a value between 10^{-3} and 10^{-2}. We can also have a look at the learning convergence rate and the test convergence rate for constant values of the Stochastic Gradient Descent. We can observe in Fig.4 that we get better results for stepsize close to the one we have highlighted in Fig.4. We can also see that for a high value of stepsize, SGD oscillates way too much. Comparing the test and train curves, we manage to get pretty much the same behavior, but reaching a better value for the train curve.

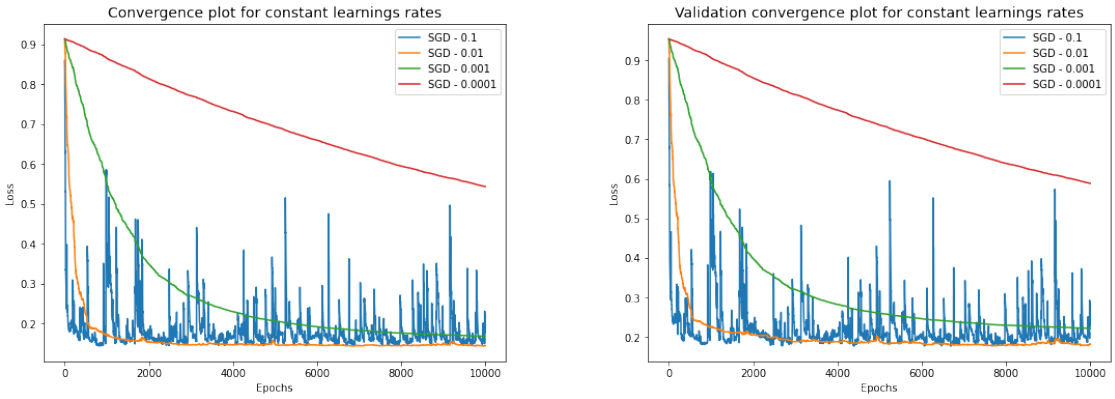

Figure 5: Evolution of learning curves of SGD for various constant stepsizes

Then, we tried to see how the learning curves behaves with decreasing stepsizes. It consists in computing a new stepsize α_k at each iteration of the algorithm following this relation :

$$\alpha_k = \frac{\alpha_0}{(k+1)^m} \tag{5}$$

Note that m is a hyperparameter we called *StepChoice* in our Python implementation. The usual value of m is 0.5 which computes the square root. We have tried to study the impact of a decreasing stepsize on the convergence plots compared to an optimal constant one. We ran these for both SGD and BGD with $\alpha_k = 0.001$ for the constant stepsizes, and with $\alpha_0 = 0.2$ and $StepChoice = 0.50$ for decreasing stepsizes.

We can see that we obtained faster convergences by using decreasing stepsizes. This is more observable for the Stochastic Gradient Descent with constant stepsize that has not converged to the same results as the other methods. We can also notice that SGD shows better results than BGD, since it only uses one sample per iteration. So it is cheaper, and the lectures also illustrates this by the fact that 1 iteration of a full Batch Gradient Descent is equivalent to n computation of iterations of SGD.

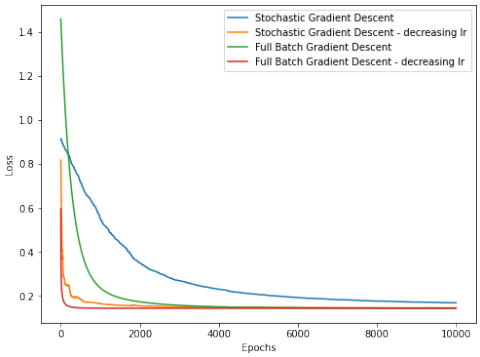

Figure 6: Comparison between SGD and BGD for constant and decreasing stepsizes

3.1.3 Exploration of Mini-Batches

One of the main hyperparameter of Stochastic Gradient Descent is the *BatchSize*. We have tried to run several Stochastic Gradient Descent for various sizes of mini-batch.

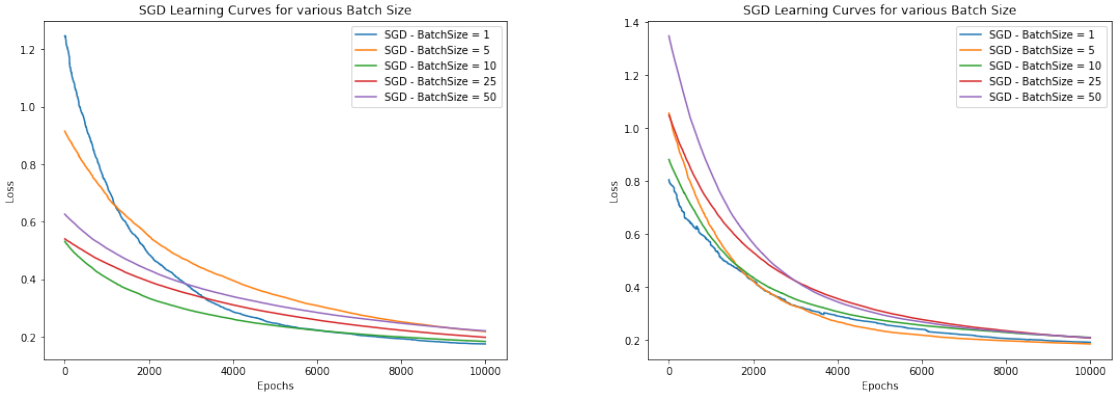

Figure 7: Evolution of learning curves of SGD for various batch size

We can see that we got better results for small size of batch size. However, note that SGD relies on some randomness when samples are drawn at each iteration, which can explain why we can obtained different results if we run again the SGD. To solve this issue, we can try to fix the seed of the random functions called in our implementation. However, since the batches share some samples in the case of a fixed seed, we did not manage to get efficient visualization to compare the impact of batch size, and we got almost the same curves for each batch size.

3.1.4 Diagonal Scaling

Since Stochastic Gradient Descent is sensitive to the conditioning of the problem, it has been shown that linear transformations can have a non negligeable impact. Diagonal Scaling is one of them, and the two methods AdaGrad and RMSProp may give good results applied on our problem. This corresponds to rescaling the stochastic gradient step componentwise as follows

$$[\mathbf{w}_{k+1}]_i = [\mathbf{w}_k]_i - \frac{\alpha}{\sqrt{[\mathbf{v}_k]_i + \mu}}[\nabla f_{i_k}(\mathbf{w}_k)]_i, \tag{6}$$

Where $\mu > 0$ is a regularization parameter, and $\mathbf{v}_k \in \mathbb{R}^d$ is defined recursively by $\mathbf{v}_{-1} = 0_{\mathbb{R}^d}$ and

$$\forall k \geq 0,\ \forall i = 1, \ldots, d, \qquad [\mathbf{v}_k]_i = \begin{cases} \beta[\mathbf{v}_{k-1}]_i + (1-\beta)[\nabla f_{i_k}(\mathbf{w}_k)]_i^2 & \text{for RMSProp,} \\ [\mathbf{v}_{k-1}]_i + [\nabla f_{i_k}(\mathbf{w}_k)]_i^2 & \text{for Adagrad.} \end{cases} \tag{7}$$

We will use the values suggested in the lab session : $\mu = \frac{1}{2\sqrt{n}}$, $\beta = 0.8$.

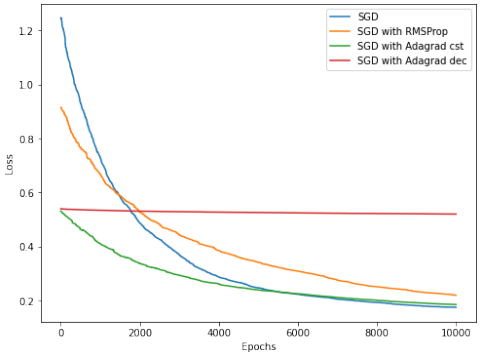

Figure 8: Comparison of diagonal scaling methods

In our problem SGD appears to converge faster with AdaGrad constant, than the other diagonal scaling methods.

3.1.5 Nesterov Acceleration

Now, we will use the Nesterov acceleration to improve our Gradient Descent methods. This consists in adding a momentum term $w_k - w_{k-1}$ at each iteration to the computation of the gradient. So, the new iteration of the Gradient Descent is :

$$w_{k+1} = w_k - \alpha_k \nabla f(w_k + \beta_k(w_k - w_{k-1})) + \beta_k(w_k - w_{k-1}) \tag{8}$$

With β_k a parameter to tune. The optimal one in the context of our problem is 0.9.

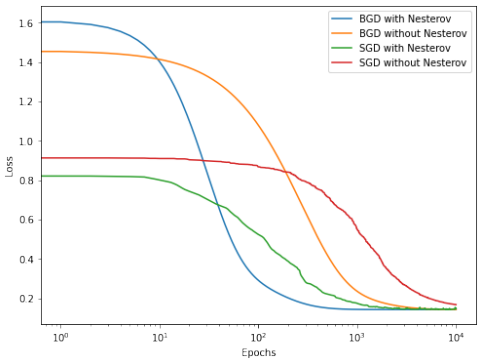

Figure 9: Impact of Nesterov acceleration on SGD and BGD

So, we can clearly see that for every methods, the add of momentum appears to make the convergence faster.

3.1.6 Ridge Regularization

As introduced on the introduction of the problem, we have also tried to use Ridge regression as it helps with strong convex problems. However, it does not appears to work very well for our regression problem as we can see on the plot Fig.10.

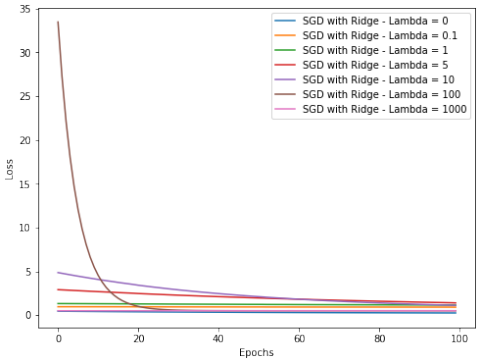

Figure 10: Impact of Ridge regularization for various λ

3.2 Other Methods

3.2.1 Variance Reduction Method - SVRG

We have decided to look at one of the two variance reduction method we saw during the lectures : the Stochastic Variance Reduced Gradient. This methods consists in computing a full gradient at the beginning of each iteration. Then, we perform partial gradient in an inner loop for m sub-iterations. We can summarize the process as follow :

- Every outer iteration k begins with a full gradient computation, yielding $\nabla f(\mathbf{w}_k)$;
- The method then performs an inner loop of m iterations starting from $\tilde{\mathbf{w}}_0 = \mathbf{w}_k$; for each iteration of index $j \in \{0, \ldots, m-1\}$ within this loop, an index i_j is drawn (uniformly) at random within $\{1, \ldots, n\}$ and the method performs the update $\tilde{\mathbf{w}}_{j+1} = \tilde{\mathbf{w}}_j - \alpha \tilde{\mathbf{g}}_j$, where

$$\tilde{\mathbf{g}}_j = \nabla f_{i_j}(\tilde{\mathbf{w}}_j) - \nabla f_{i_j}(\mathbf{w}_k) + \nabla f(\mathbf{w}_k).$$

- At the end of the inner loop, we obtain the next outer iterate by $\mathbf{w}_{k+1} = \tilde{\mathbf{w}}_m$.

One iteration of SVRG is equivalent in cost to a full gradient iteration, because 2m + n gradients are required per iteration. However, it can still be faster than gradient descent, because of the intrinsic randomness.

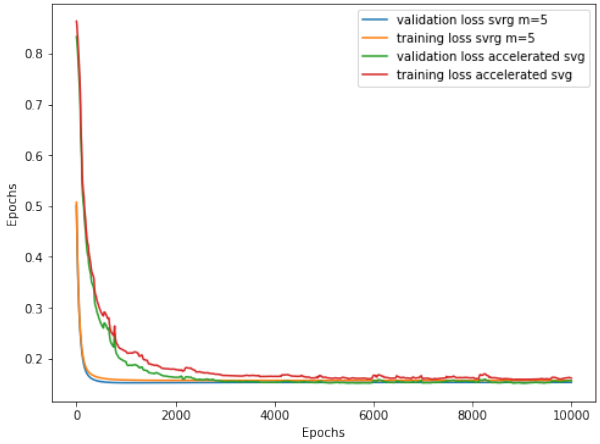

Figure 11: Comparison between SVG and SVRG

We managed to get faster convergence for SVRG in both training and validation phases as shown in Fig.11 for m=5.

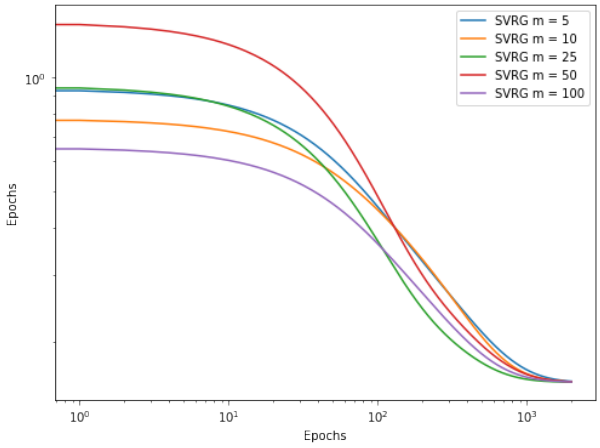

Figure 12: Impact of m for SVRG

As we increase the size of m, we can see that we got slightly better results in term of convergence to the optimal solution.

4 Results comparison

To check the efficiency of our methods, we will use the *Scikit-Learn* library to check our result. We used the *LinearRegression()* and the *SGDRegressor()* functions to compare our results. We decided to check with the SVRG and the accelerated SGD methods which provided the best results throughout the project. We got the following results :

Method	Validation Loss	Training Loss
SkLearn LinearRegression	0.1679	0.1512
SkLearn SGDRegressor	0.1682	0.1512
Accelerated SGD	0.1738	0.1559
SVRG	0.1678	0.1512

Table 1: Loss comparison between Scikit Learn models and our methods

So, we got convincing results. We managed to match Scikit Learn values with our SVRG method and to get pretty close with the Stochastic Gradient Descent with Nesterov acceleration ($\beta = 0.9$, $batchsize = 1$).