EVALUATION OF ADAPTIVE LEARNING RATE AND MOMENTUM PERFORMANCE

Authors: Girardi P., Rodio A.

ABSTRACT

Optimization is one of the most important part of machine learning, especially the optimization related to the model training phase. Numerous algorithms and different techniques have been developed over the years in order to reduce the model train time; among these, algorithms have been studied that consider an adaptive learning rate and the use of momentum. In this project we are going to study and analyze some of these algorithms in order to evaluate their absolute and relative performances in presence of strongly convex objective functions.

1 INTRODUCTION

The simplest algorithm used to perform iterative minimization is the gradient descent. The mini batch algorithm is a good trade-off between the number of iterations to converge and the speed of an iteration. Using a constant learning rate, this algorithm will converge to the minimum value of the function plus a certain error, which will depend on the size of the mini batch and the chosen learning rate (1). Other problems of this algorithm are the tendency to diverge and the difficulty in overcoming the saddle points, the maximum points and the local minimum points. These problems can be mitigated by using adaptive learning rate and momentum techniques (1).

We implemented some of the most famous gradient descent algorithms from scratch in python, and compared performances on a common dataset.

2 PROBLEM FORMULATION

As a reference test bench, we simulated a linear regression problem. We generated a straight line of equation $y = ax + b$, where the real parameters are $a = 4$ and $b = 18$. In order to create our dataset, we added Gaussian noise with standard deviation $\sigma = 0.5$, generating 800 data samples (Figure 1). On this dataset we applied all the algorithms that we want to analyze, trying to estimate the real parameters of the line. The goal of our ML problem is to minimize the loss function (mean squared error) over the parameter vector $\mathbf{w} = (a, b)$:

$$F(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - (ax_i + b))^2 \tag{1}$$

Plotting this function, it is possible to see that it is strongly convex. (Figure 1)

3 COMPARISON BETWEEN FULL-BATCH, MINI-BATCH AND STOCHASTIC GRADIENT DESCENT

As the figure 2 shows, in all three algorithms, the parameter a converges in very few iterations (order 10), while parameter b converges in about 1000 iterations. Considering the loss functions, the full batch converges in a few iterations but needs 589ms, the mini batch, considering a batch size of 200, converges in 347ms and the stochastic converges in 115ms, but enters the confusion region after a few iterations. Considering these results, we will use the mini batch with batch size of 200 data samples, as a starting point to analyze the performance of subsequent algorithms.

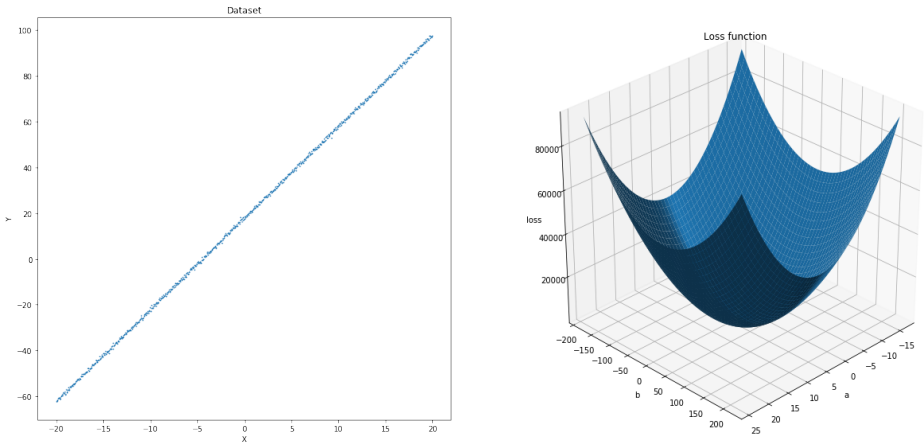

Figure 1: Plot of the dataset and loss function.

4 Comparison between mini batch, momentum and Nesterov momentum

Momentum Gradient methods with momentum are procedures in which each step is chosen as a combination of the steepest descent direction and the most recent iterate displacement. Specifically, with an initial point w_1, learning rate α_k and constant β_k, these methods are characterized by the iteration:

$$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \alpha_k \nabla F_{w_k} + \beta_k(\mathbf{w}_k - \mathbf{w}_{k-1}) \tag{2}$$

Using the gradient method with momentum, it is possible to avoid the ill conditioning problem, to go through the saddle points and to archive noise reduction (1).

Nesterov momentum The accelerated gradient method, also called Nesterov momentum, is a procedure that follows the momentum term first, and then applies a steepest descent step.

$$\mathbf{g}_k = \nabla F(\mathbf{w}_k + \beta(\mathbf{w}_k - \mathbf{w}_{k-1})) \tag{3}$$

It is proved that the Nesterov momentum archive the fastest possible convergence speed (1).

Comparison As we can see from the figure 3, using momentum and Nesterov momentum, the parameter b converges in about 200 iterations against the 1000 iterations necessary for the mini batch. The three algorithms have very similar execution times: to execute 1000 iterations, the two algorithms that use momentum need 358ms. Convergence instead is reached in 75ms, against the 348ms of the mini batch. Performance improves by 80% compared to the mini batch.

5 Comparison between mini batch, AdaGrad, RMSProp and Adam

AdaGrad The AdaGrad algorithm individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all the historical squared values of the gradient. This means that the parameters with the largest partial derivative of the loss have a correspondingly rapid decrease in their learning rate, while parameters with small partial derivatives have a relatively small decrease in their learning rate (2).

RMSProp The RMSProp algorithm uses an exponentially weighted moving average to perform better in the non-convex setting. This algorithm uses an adaptive learning rate and the Nesterov momentum (2).

Adam The Adam algorithm uses momentum incorporated directly as an estimate of the first-order moment (with exponential weighting) of the gradient and includes bias corrections to the estimates

of both the first-order moments and the second-order moments to account for their initialization at the origin (2).

Comparison The advantages of these algorithms are known in the literature when applied to non-strongly convex functions. In this project we asked ourselves how these algorithms perform in the presence of strongly convex functions.

Performing 1000 iterations with each of these algorithms, we have shown that the times are very similar to each other and to those of the mini batch and momentum algorithms. In particular, AdaGrad took 401ms, RMSProp 361ms and Adam 374ms. As it is possible to see in the figure 4, with the use of AdaGrad, the convergence of the parameter b is very similar to the convergence of the Nesterov.

It is possible to see that parameter b converges in about 200 iterations using AdaGrad, in about 10 iterations using RMSProp and Adam converges with the order of 10 iterations but it is possible to notice periodic up and down phenomena.

Considering the convergence times, AdaGrad converges in 80ms, roughly like the convergence time of Nesterov, RMSProp converges in less than 10ms and Adam converges in about 20ms.

6 CONCLUSION

We can conclude that these algorithms can also be used in optimization problems with non-strongly convex functions and provide excellent performance. In particular, we noticed that RMSProp outperforms the others. Adam shows approximately the same convergence time of RMSProp but shows more noise. We found out that the momentum algorithms strongly depend on the user choice of the learning rate. The three adaptive learning rate algorithms, on the contrary, don't show this dependency and so are easier to setup.

In (2), Goodfellow et al. claim that Adam is generally robust to the choice of hyper-parameters. Instead, in our experience, we found some difficulties in the tuning of hyper-parameters. The same difficulties are described in (3), in case of non-strongly convex objective. We didn't find the same difficulties in RMSProp, that revealed easy to tune and led to the following results: improvement of the performance of 98% compared to the mini-batch gradient descent and 95% with respect to the Nesterov momentum.

REFERENCES

[1] Bottou, L and Curtis, FE and Nocedal, J. Optimization methods for large-scale machine learning. ArXiv, 2016

[2] Goodfellow, Ian and Bengio, Yoshua and Courville, Aaron. Deep learning. MIT press, 2016.

[3] Sylvain Gugger and Jeremy Howard. AdamW and Super-convergence is now the fastest way to train neural nets. `https://www.fast.ai/2018/07/02/adam-weight-decay/`, 2018.

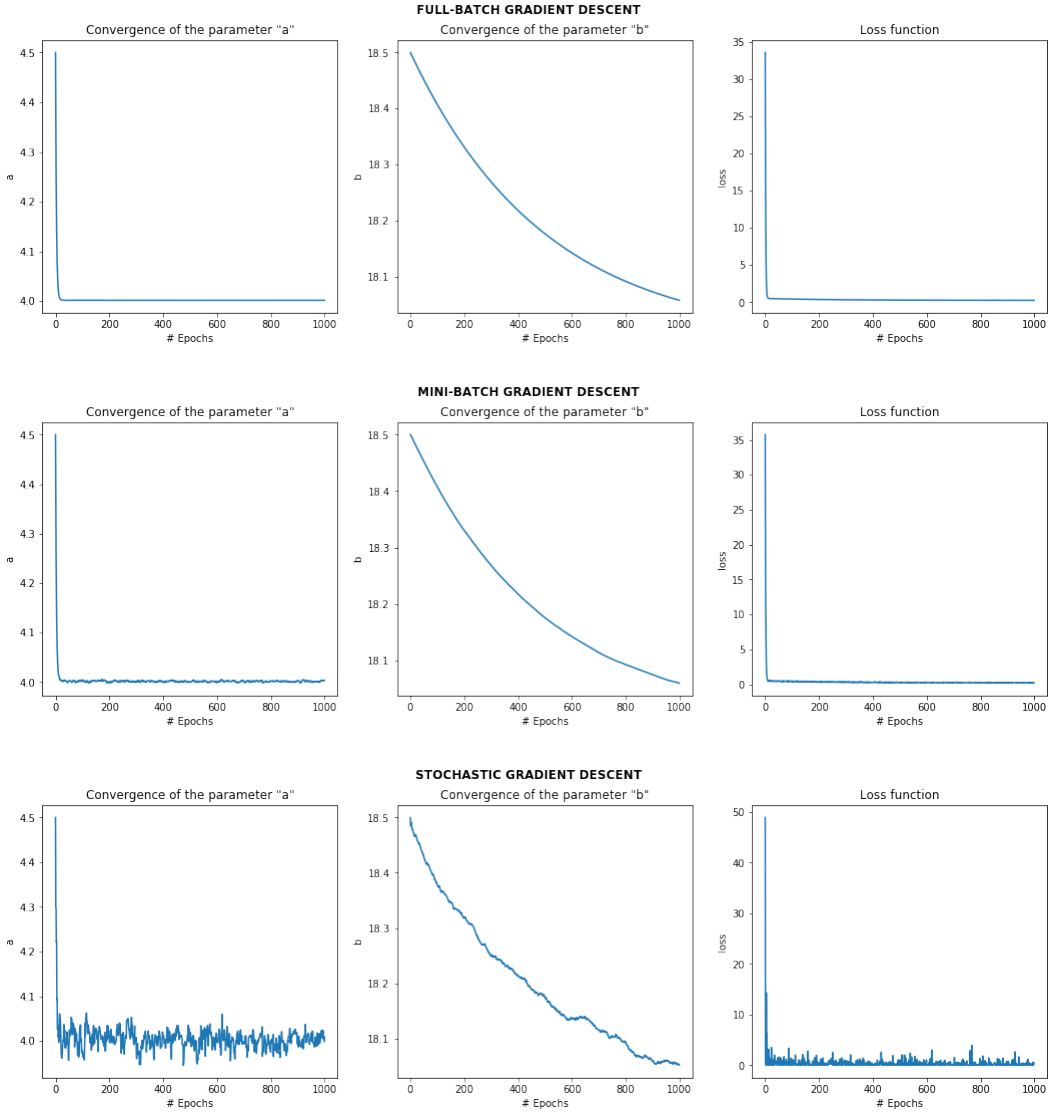

Figure 2: Comparison between full-batch, mini-batch and stochastic gradient descent.

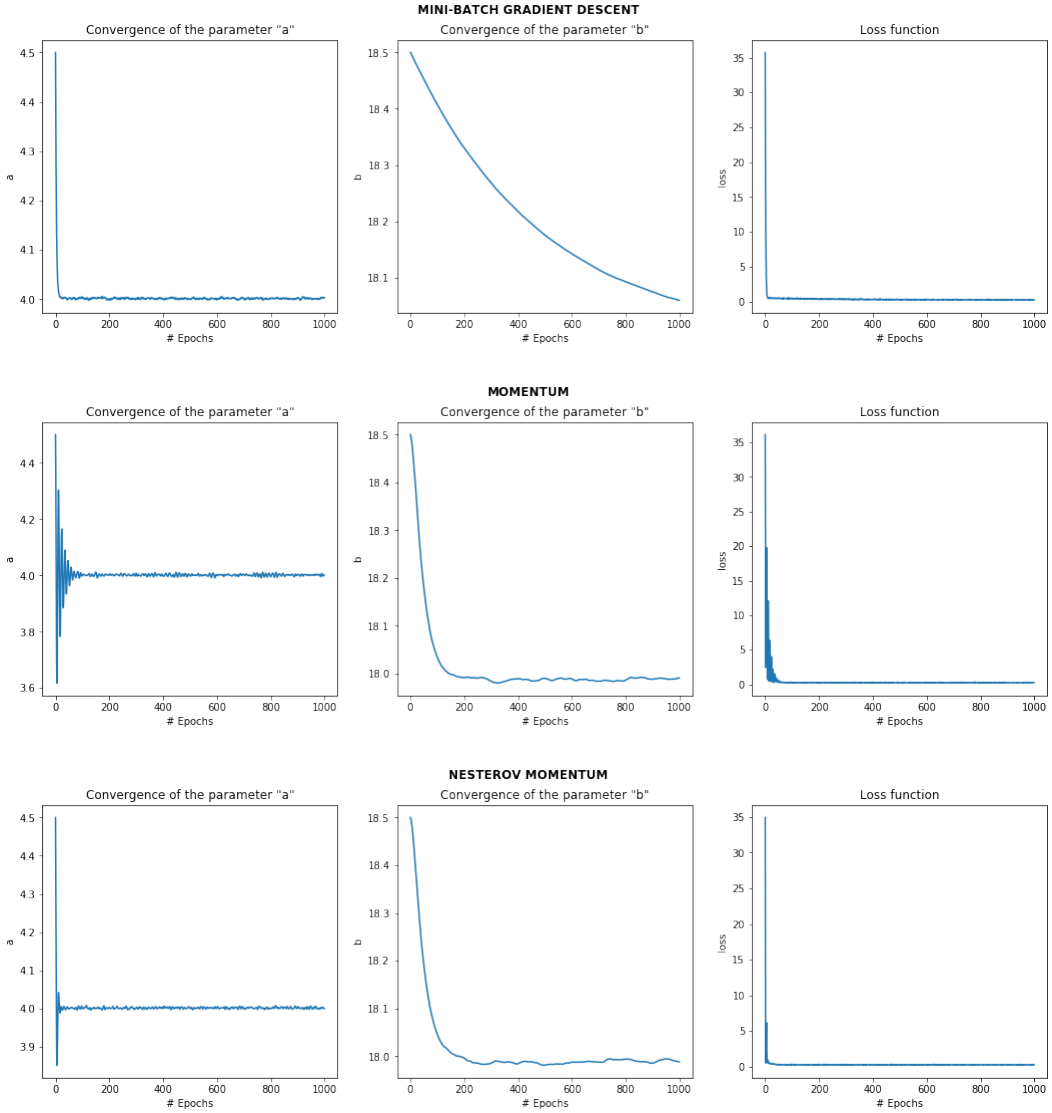

Figure 3: Comparison between mini batch, momentum and Nesterov momentum.

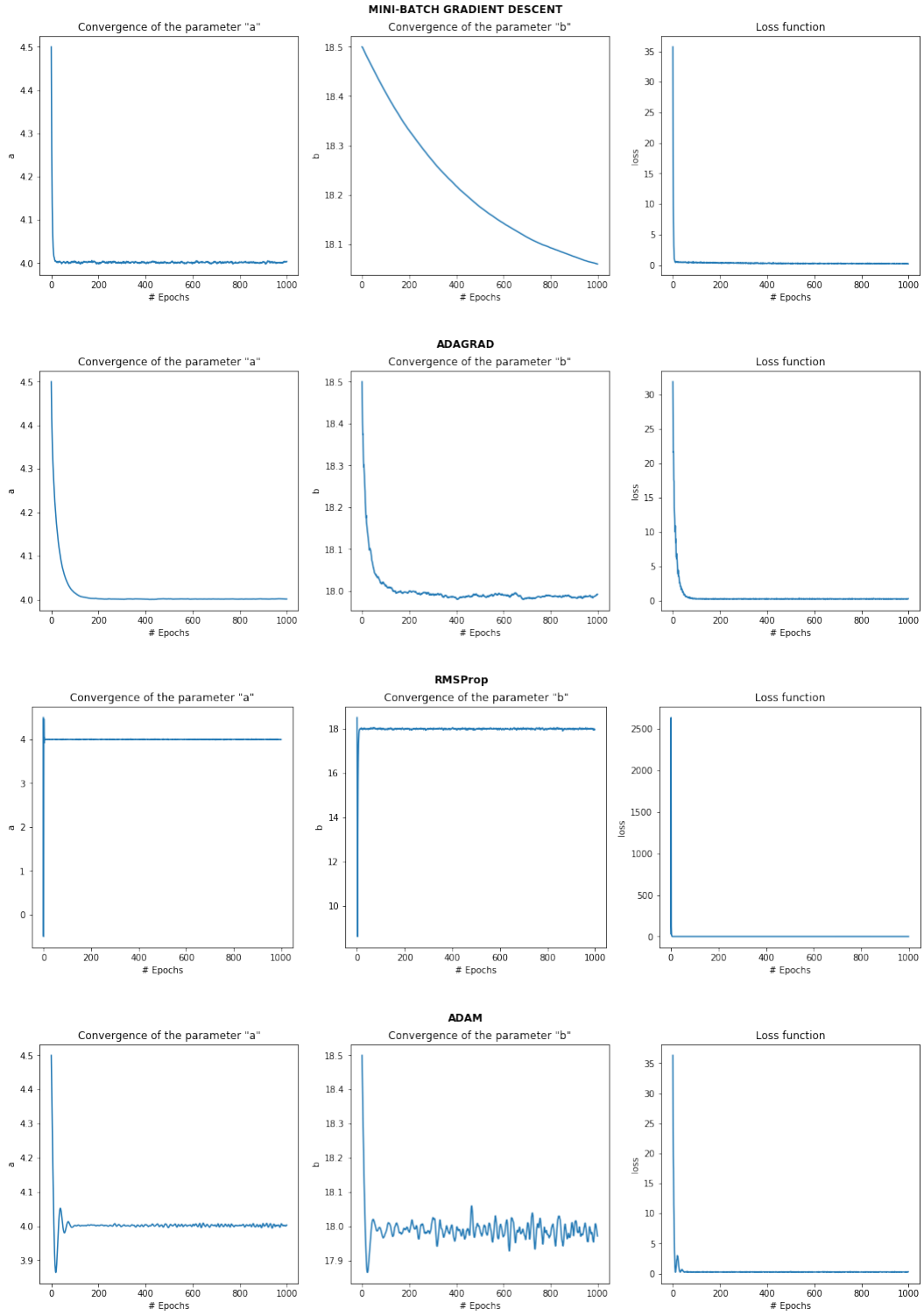

Figure 4: Comparison between mini batch, AdaGrad, RMSProp and Adam.