Gradient descent optimization in Spark

Chen Dang & Dinh Cong Minh
Machine learning for Big data, Master IASD, Paris Dauphine - PSL University

June 7, 2020

1 Introduction

Big data solutions focus mainly on the Extraction and Transformation aspect of processing. The MapReduce model allows us to easily implement information extractions, but many constraints and limitations appear when designing data algorithms.

For example, the iterative algorithms commonly used in machine learning are difficult to integrate into MapReduce models: the high level of data interaction requires complex management and synchronization at different stages of the analysis.

In this article we will try to face this difficulty and apply in a typical use case in machine learning: the design of an SGD model. Our goal is to demonstrate the adaptability and elegance of implementation using the distributed computing framework Spark.

2 Spark Learning

The learning phase of the model consists of iterating over the entire dataset, correcting the parameters at each stage in order to converge towards an optimum classification rate.

The important points of this architecture are:

- The dataset is cut and distributed on the nodes of the cluster.
- The parameters are sent to all the nodes at each new iteration.
- The aggregation of gradients corresponds to a reduce phase, therefore also operated in parallel.

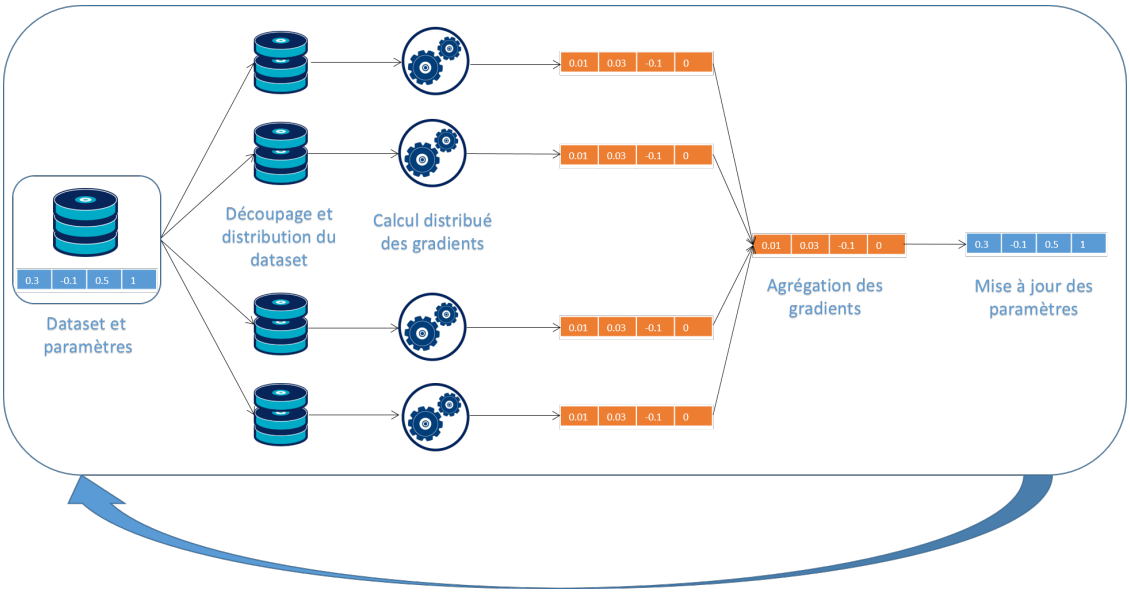


Figure 1: Architecture of Spark

2.1 Dataset

In this project we are going to two datasets:

1. Iris data set: the small set (50 samples)
2. California dataset: the big dataset (20640 samples)

We will study SGD algorithms for two types of data, one larger and one smaller to evaluate their effects for Spark.

2.2 Implementation

We use the Python API of the Spark framework. We wanted to use this language because Python offers many libraries facilitating scientific calculation (in our case NumPy).

First of all, we import the Python and Spark libraries necessary for the algorithm:

1. pyspark: import of the Spark API. The SparkContext object will manage all communications with the cluster.
2. numpy: used for matrix operations.
3. sklearn: used for downloading datasets.
4. matplotlib: plotting the result.

Next, we implement the code of SGD and its gradient descent optimization algorithms, which will be theoretically expressed in the next section.

3 Theories

3.1 Stochastic gradient descent

Stochastic gradient descent (SGD) in contrast performs a parameter update for training example $x^{(i)}$ and label $y^{(i)}$

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

3.2 Mini-batch gradient descent

Mini-batch gradient descent finally takes the best of both worlds and performs an update for every mini-batch of n training examples:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

3.3 Gradient descent optimization algorithms

3.3.1 Momentum

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another, which are common around local optima.

It does this by adding a fraction γ of the update vector of the past time step to the current update vector:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

3.3.2 Nesterov accelerated gradient

Nesterov accelerated gradient (NAG) is a way to give our momentum term this kind of prescience:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

3.3.3 Adagrad

Adagrad is an algorithm for gradient-based optimization that does just this: It adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rates) and larger updates (i.e. high learning rates).

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \nabla_\theta J(\theta_{t,i})$$

As G_t contains the sum of the squares of the past gradients.

3.3.4 Adadelta

Adadelta is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate.

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$
$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

where the root mean squared error of parameter updates is:

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

and the exponentially decaying average:

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

3.3.5 RMSprop

RMSprop in fact is identical to the first update vector of Adadelta that we derived above but the decaying average is:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

3.3.6 Adaptive Moment Estimation (Adam)

In addition to storing an exponentially decaying average of past squared gradients v_t like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum.

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

where first and second moment estimates are:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

The parameters will be then updated:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

3.3.7 AdaMax

The v_t factor in the Adam update rule scales the gradient inversely proportionally to the ℓ_2 norm of the past gradients (via the v_{t-1} term) and current gradient $|g_t|^2$:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)|g_t|^2$$

$$u_t = \max(\beta_2 \cdot v_{t-1}, |g_t|)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t}\hat{m}_t$$

3.3.8 Nadam

Nadam (Nesterov-accelerated Adaptive Moment Estimation) thus combines Adam and NAG.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}(\beta_1 \hat{m}_t + \frac{(1 - \beta_1)g_t}{1 - \beta_1^t})$$

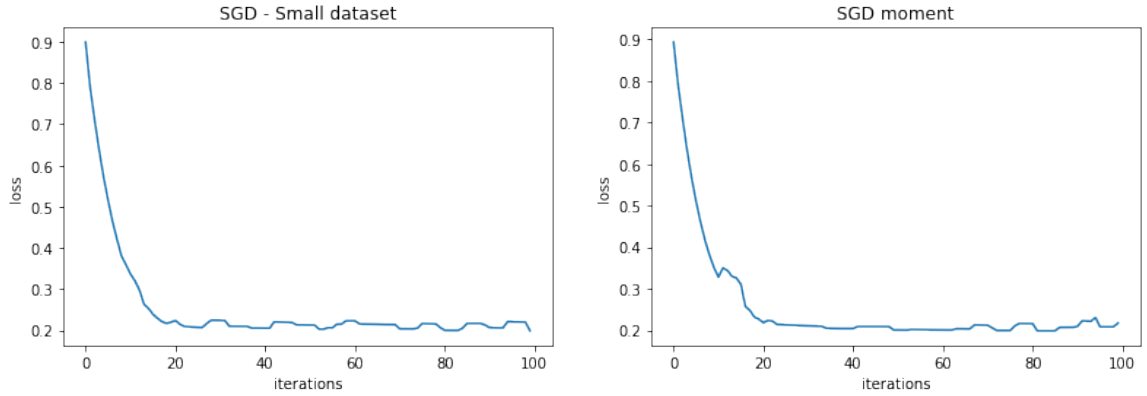
3.3.9 AMSGrad

In settings where Adam converges to a suboptimal solution, it has been observed that some minibatches provide large and informative gradients, but as these minibatches only occur rarely, exponential averaging diminishes their influence, which leads to poor convergence. The authors provide an example for a simple convex optimization problem where the same behaviour can be observed for Adam: AMSGrad.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2$$
$$\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}m_t$$

4 Result & Dicussion

Frist, we observe the result obtained by SGD:



(a) SGD (b) SGD moment

Figure 2: Loss of SGD and SGD moment

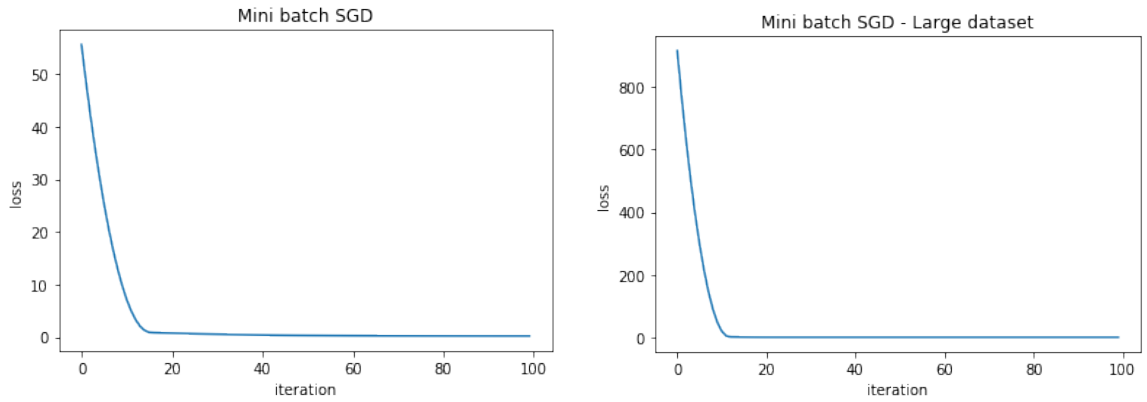
We have a convergence here but the result is not really good, indeed there is still noise. So, we did a momentum step and added it to the gradient step. Our result SGD-moment is indeed better than SGD but still can be improved.

We will comment on the elimination of these noises in the next section by different optimization methods in the following sections.

To be transparent, we also note that the same parameters used for all the optimization methods: $\eta = 0.002, \gamma = 0.01, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10e - 8$.

4.1 Comparison of small and large dataset

Mini batch reduces the variance of the parameter updates and lead to more stable convergence. Indeed, in the following figure, we have seen that convergence is very stable, even for a small or a large set of data.



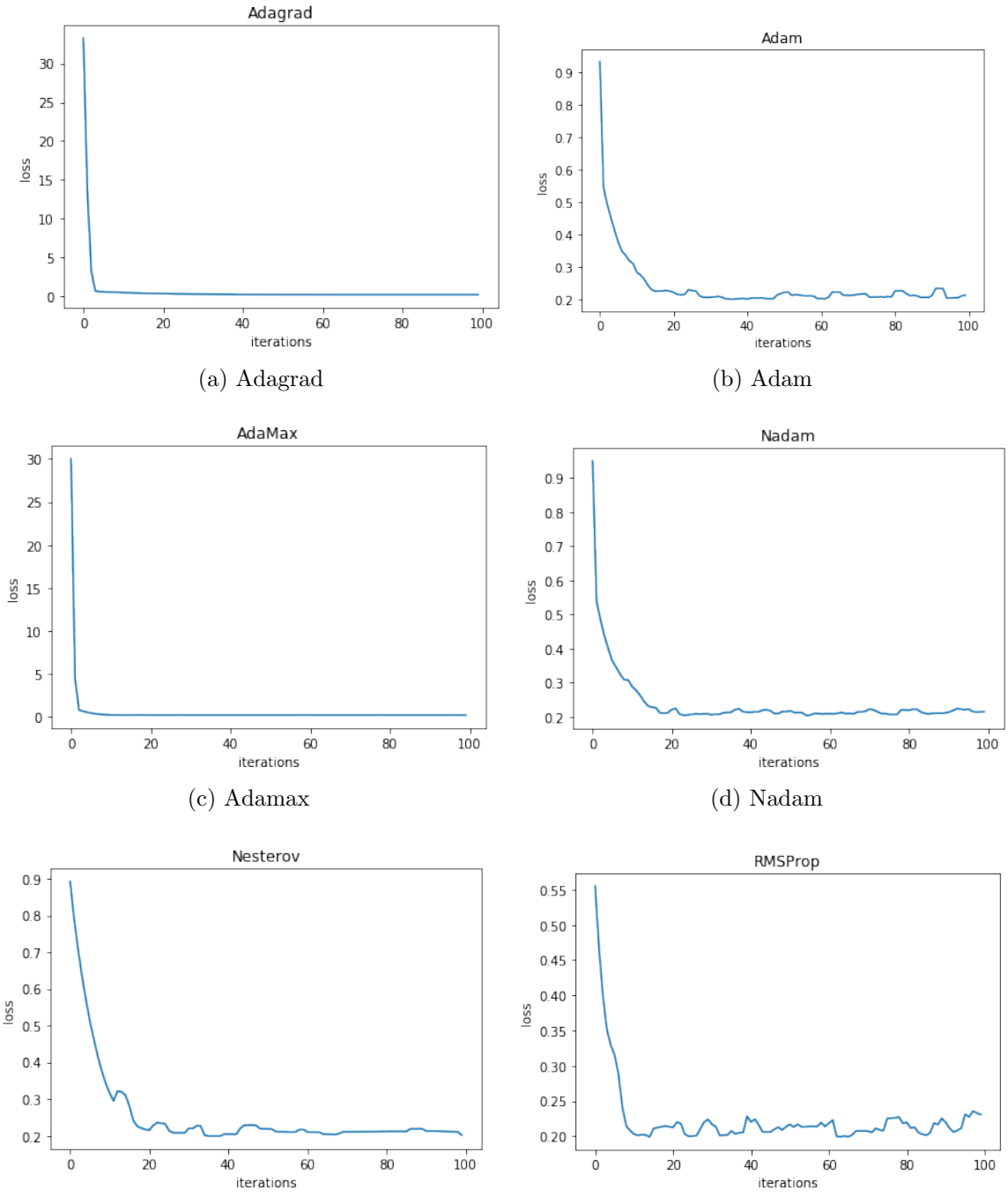
(a) Small dataset (b) Large dataset

Figure 3: Loss of Mini batch SGD

We get the same convergence result for the large and the small data set. For the large

data set, the initial loss of large dataset is much higher than the one of the small data set, but after 15 iterations, they are also both convergent. We have noted that the algorithm works well even for a small or large set of data.

4.2 Comparison of different methods



(a) Adagrad

(b) Adam

(c) Adamax

(d) Nadam

(e) Nesterov

(f) RMSProp

Figure 4: Loss function for different optimization methods

NAG (Figure e) is not much different but it wants to take the momentum step first and calculate the gradient on the top of it. The result with NAG is indeed better than SGD moment.

Adagrad (Figure a) understands where the learning rates must be slow or fast (for dense and sparse features). The convergence obtained in the figure is very fast and very stable.

However, one disadvantage to the Adagrad is that there might be a problem of vanishing gradient due to the learning rate being divided by the sum of the gradients. To solve this problem of vanishing gradients, instead of summing the gradients, RMSProp can sum the exponentially weighted averages. The result of RMSProp is obtained in the Figure f.

The result obtained in RMSProp is not really good. We have a better result in Adam (Figure b). Indeed, it is because Adam combines the advantages of RMSprop with momentum.

We can also improve the result in Adam. Indeed, an improvement of Adam - Nadam - modifies Adam to use Nesterov's momentum instead of the classical momentum and returns a better result (Figure d).

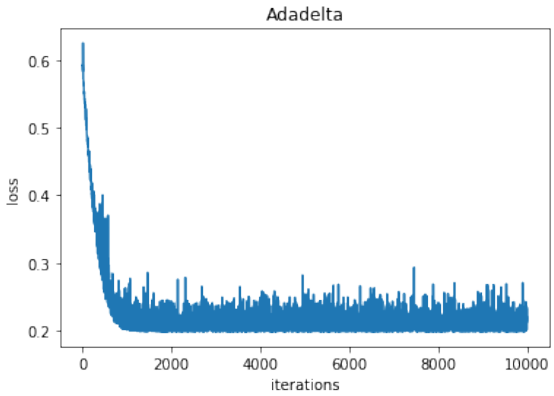


Figure 5: Adadelta

Adadelta is also used to solve the problem of the vanishing gradients. Adadelta convergence is obtained more slowly and there is a lot of noise compared to the other methods as shown in the figure above.

4.3 Time processing

The biggest advantage of using Spark is the largely improved time processing. Indeed, the processing time with Spark is much faster than the case without using Spark.

In particular, we also note the treatment of time for two cases of dataset:

- For the small dataset: $2s$ for one iteration
- For large dataset: $150s$ for one iteration

We also noted that the small dataset had only 50 samples and that the large dataset had 20,000 samples (more than 400 times). But the processing time between large and small dataset is only 75 times more. We conclude that Spark really helps a lot with a large dataset and the use of Spark is essential for a large-scale dataset.

General conclusion

In this project, we have implemented many methods for gradient descent optimizations using spark and compared their performances. The results we obtained demonstrate the good performance of distributed computing on large data sets. We also showed the importance of using Spark for a large dataset.

Apache spark appears today as an environment to explore. Spark is becoming a benchmark big data framework, and is gaining more and more attention.