Fast SGD with Adagrad and Momentum (RDD, Dataframe, Dataset)

Merhben Oumaima , Maryam Azzouz , Aouadni Amel

Paris Dauphine University PSL

1 Introduction

Spark MLlib is a scalable machine learning library, that provides RDD-based APIs to support scalable machine learning. It consists of common machine learning algorithms in the area of distributed machine learning, to handle very large datasets and/or complex models.

Gradient descent, is one of the most popular methods in the field of machine learning since it can minimize error rate by following the fastest direction.

In this project we designed different asynchronous parallel gradient descent algorithms with iterative local search to reuse the data partition multiple times in a single global iteration.

We implement these different algorithms with the 3 APIs of Spark using scala: RDD, Dataframe and Dataset.

2 Implemented Gradient Descent algorithms :

2.1 Gradient Descent :

We use all the samples in the data set to optimize the parameters. For each iteration, we make a single update.

2.2 Stochastic gradient descent

Gradients are calculated for each sample in the data set and therefore updated for each sample in the data set.

2.3 Mini Batch :

The Mini batch captures the good aspects of stochastic gradient descent and gradient descent. Instead of a single sample (stochastic GD) or the dataset (GD), we take mini batchs or pieces of the dataset and set up the parameters accordingly.

2.4 Momentum :

It helps SGD to navigate in the relevant directions and softens the oscillation in the irrelevant directions. We add a fraction of the direction of the previous step to that of the current step, which increases the speed of amplification in the right direction

2.5 Adagrad :

It allows learning to adapt according to different parameters. It performs small updates for frequent settings and large updates for infrequent settings. It also eliminates the need to adjust the learning rate. Here, each parameter has its own learning rate and it decreases monotonously.

After implementing the different functions, we will use the 3 Spark APIs: RDD, Dataframe and Dataset and we will compare their performance. To test the algorithms, we use syntactic data that we generated.

We considered the couples: ((x, x + 1), y) where:

$$5 * x + 2 = y$$

3 RDD implementation

To distribute the tasks between the nodes and at the same time update the parameters each time according to the different gradient descent algorithms, we use different actions and transformations on the RDDs provided by spark.

1/ We create RDDs by parallelizing our data.

2/ We make a repartition of the RDDs to define the number of partitions and at the same time to shuffle the data.

3/We use Glom and zipWithIndex to index partitions in order to filter them in the next step.

4/Finally we update the parameters consecutively. Each time there is only one worker who updates the parameters and then passes them to the driver which passes them to the next worker.

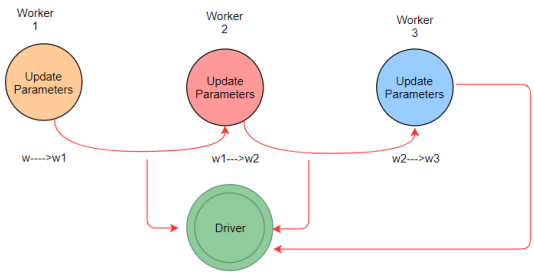

Figure 1: Update weights vector

4 Dataframe

Unlike an RDD, the data is organized in named columns. It is an immutable distributed collection of data. DataFrame in Spark allows us to impose a structure on a distributed collection of data, allowing higher level abstraction. We therefore transform the data into a dataframe which contains the x and the y. For this we use the toDF () function.

We make then a repartition of the DF on a certain number of partitions that we define and at the same time to shuffle the data.

After that we use the function provided by spark.sql: spark partition id, we add a column contains the partition ids which allow us to filter them later.

We continue then the same process as we did with the RDDs.

5 Dataset

Datasets offer a compromise or a mix between RDD and DataFrames. The dataset implementation is practically the same as dataframe (using toDS ()).

6 Comparison

Comparing the execution time of each Spark API we find that a DataFrame/Dataset tends to be more efficient than an RDD in term of time .

7 Conclusion

RDD are typed and offer a way to get analysis errors at compile time.

It is a low level API with no performance optimization. It is less expressive too and is more useful for unstructured data.

DataFrame is more expressive and more efficient.

However, it is untyped and can lead to runtime errors.

Dataset looks like DataFrame but it is typed. With them, we have compile time errors.

8 References

1/ https://www.dropbox.com/s/uaq4jzpcx1kz1kv/MOMENTUM-ADAGRAD-1609.04747.pdf

2/ https://spark.apache.org/docs/2.3.0/api/java/index.html?org/apache/spark/sql/Dataset.html

3/ http://spark.apache.org/docs/latest/sql-programming-guide.htmlstarting-point-sparksession