MapReduce for Neural Networks

Report

Olivier DUTFOY : A20364492
Nicolas LASSAUX: A20365397

April 18, 2016

Contents

1 Main Paper and publication details **2**

2 Problem Statement **2**

2.1 MRBPNN_1 2

2.2 MRBPNN_2 3

2.3 MRBPNN_3 3

3 Implementation **4**

3.1 Parallelized Neural Networks 4

3.2 Apache Spark 4

3.2.1 Map 5

3.2.2 Broadcast 5

4 Results **5**

5 Conclusion **7**

6 References **8**

1 Main Paper and publication details

We've decided to do our project based on the work of Yang Liu, Jie Yang, Yuan Huang, Lixiong Xu, Siguang Li and Man Qi in their paper **MapReduce Based Parallel Neural Networks in Enabling Large Scale Machine Learning** [1]. This paper was received on May 28th 2015 and accepted on the 11th of August 2015 before being published by *Hindawi Publishing Corporation.*

2 Problem Statement

We've chosen this subject because we feel like it is an important problem to address. The amount of data used has reached insane numbers in today's society. It is therefore important to be able to process as efficiently as possible.

Map reduce is a fairly common solution to improve computation time given the adequate resources. However it isn't always easy to implement depending on the problem that needs solving. Neural Networks is one of the most difficult topics we've addressed this semester while being currently relatively popular and therefore seemed like an interesting enough challenge.

The paper presents 3 different MapReduce algorithms for Neural Networks, which all fit a particular situation.

2.1 MRBPNN_1

The first algorithm focuses on the case in which the testing dataset is very big. Let n be the number of nodes (or servers) one is able to use. Each node will create it's own neural network (with the same initial parameters). The training data is then sent to each node in order to obtain n identical models (since both the initial parameters and the training data are identical. The testing data is then divided into n chunks, each of which will be processed by a node and output a result. Finally the output of each mapper is reduced together to obtain to full final results.

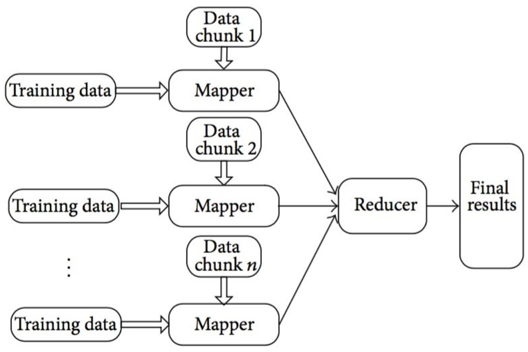

Figure 1: MRBPNN_1

While this algorithm is interesting in some situation, we felt like it wasn't very fulfilling to implement. In the end there is a unique model which is what we are usually faced with, and each node is only used to process different data once the model is created. This is why we decided to focus on the second algorithm.

2.2 MRBPNN_2

This time, the algorithm is interesting when the training dataset is large. Each node will once again create it's own neural network. However this time the training set is dividing into n chunks, each of which is given to a node. This provides n different models.

Each testing instance in then classified by each model. The final output is the class selected by the most models (called *Majority Voting*).

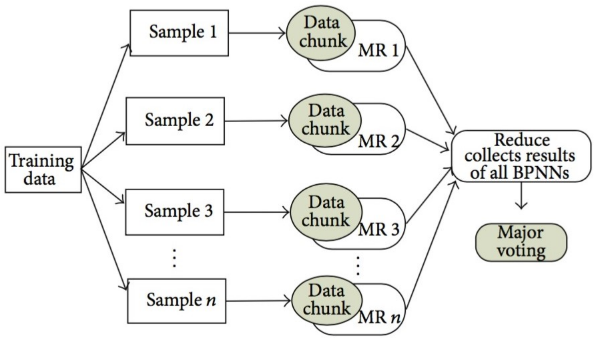

Figure 2: MRBPNN_2

2.3 MRBPNN_3

We also showed some interest in the third algorithm but unfortunately did not have enough time to implement anything functional. This algorithm differs from the previous ones and is used when the number neurons is very high. The neural network is maintained throughout the different nodes. Without going into details it uses a series of MapReduces for each instance during the feed-forward process before doing the back-propagation.

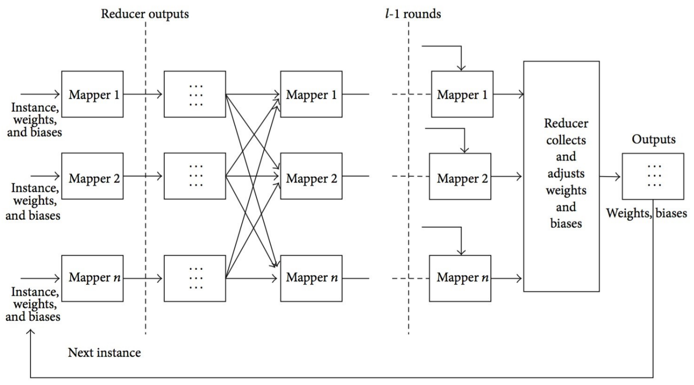

Figure 3: MRBPNN_3

3 Implementation

We implemented classic back-propagation neural networks in order to have a baseline and then used **Apache Spark** to implement the second MapReduce algorithm. We used 1 hidden layer for each of our computations.

3.1 Parallelized Neural Networks

The Neural Networks of the paper are presented slightly differently then the ones we implemented. Every neuron has a weight w_{ij} with i representing the source and j the destination. Each neuron also has a bias θ_j for the j^{th} neuron in the layer (we did not use this parameter in our implementation). With o'_j being the output of the previous layer, we can compute the following inputs for neurons :

$$\boxed{I_j = \sum_i w_{ij} o'_j + \theta_j}$$

we can then compute the output of a neuron using the sigmoid function :

$$\boxed{o_j = \frac{1}{1+e^{-I_j}}}$$

In practice we used the *tanh* function for the hidden layer and the sigmoid for the output.

We can then start the back-propagation process using the following errors function. For a neuron in the output layer :

$$\boxed{E_j = o_j(1-o_j)(t_j - o_j)}$$

with t_j the expected output. The error for previous layers is then :

$$\boxed{E_j = o_j(1-o_j)\sum_k E_k w_k j}$$

The parameters are then updated as follows :

$$\boxed{w_{ij} = w_{ij} + E_j o_j} \boxed{\theta_j = \theta_j + E_j}$$

and the operation is repeated for every training example until the maximum number of iterations is reached.

The weights and bias' are initialized randomly between -1 and 1.

3.2 Apache Spark

Apache Spark allows cluster computing and possesses a **Python** API. It enables the creation of nodes on different servers. We then have to modify our code in order to do the required parallelization. In case one does not have access to external server, **Spark** locally optimized the computation by using the different processors of the machine.

Spark is built on **Apache Hadoop** (library used by the paper) and uses a cache to improve computation time. We'll now discuss 2 of the main functions we used.

3.2.1 Map

Once the data is transformed into an RDD (resilient distributed dataset), a collection of elements, it can be distributed amongst all the nodes. The *map* function allows to apply any chosen operation for each partition on it's attributed node.

3.2.2 Broadcast

The other useful function is the *broadcast* one. We used this to give common initial variables to each of our nodes (such as the instantiation of the neural networks)

4 Results

Our evaluation focused on two primary aspects. We compared both the changes in accuracy and the computation time between using a MapReduce or the classical approach.

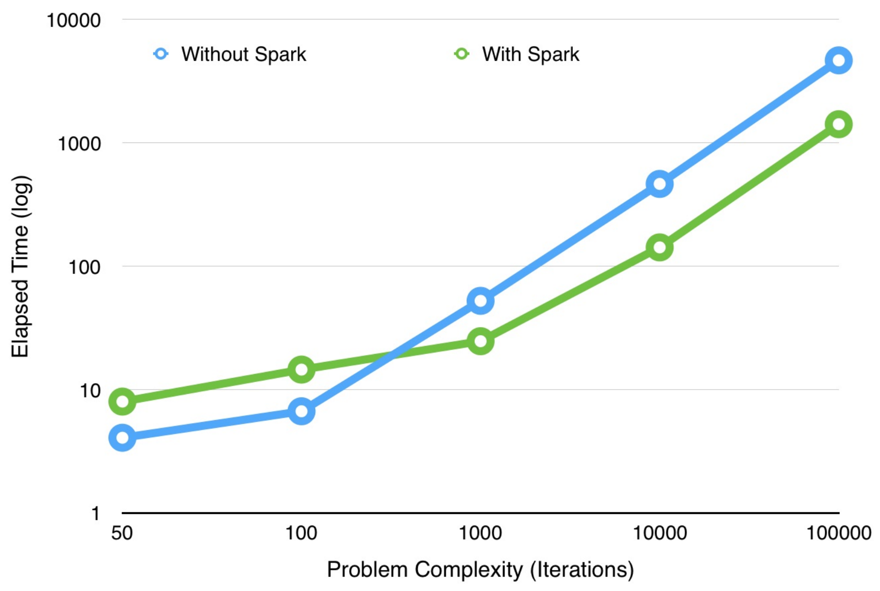

Figure 4: Spark Configuration: 8 workers (locally), on Iris dataset

Figure 4 displays computation time with and without the use of map reduce depending on the number of iterations used in the propagation of the neural network. As is expected, the use of MRBPNN_2 is faster on the long run then the more classic approach. However it does perform a little slower on a small number of iterations. This is no surprise and due to the fact that the program needs to launch **Spark** which takes a little time. After a given number of iterations, the time multiplier gain reaches a constant value. This is due to the fact that we ran the program locally on one of our computers and are limited by our hardware.

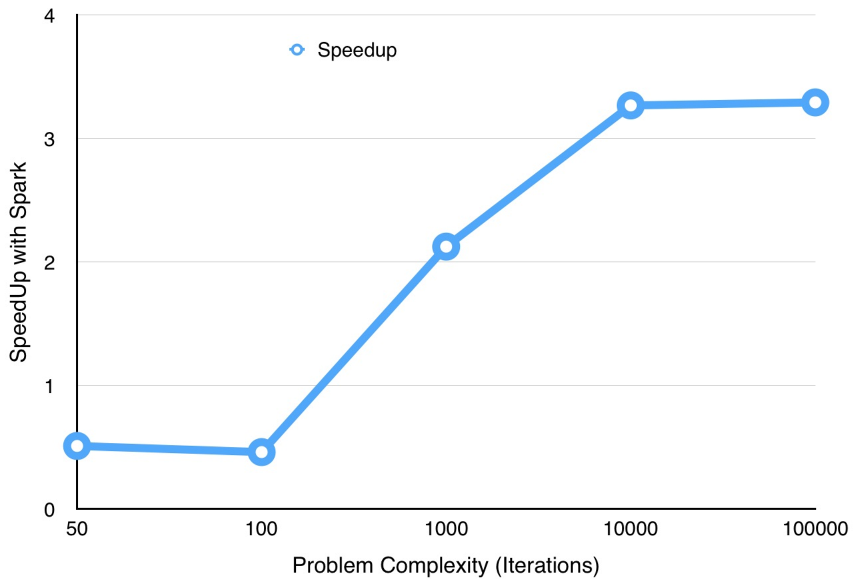

Figure 5: Spark Configuration: 8 workers (locally), on Iris dataset

Table 1: Spark Configuration: 4 workers (locally)

	Dataset Iris	Dataset Breast
Accuracy Spark	76,7%	93,6%
Accuracy Classical Neural Network	91,3%	99,0%

Even though the computation time is improved, we notice a certain loss of accuracy. While it still pretty good for the breast dataset, the data concerned requires it to be more precise. The loss of accuracy for the iris dataset is probably not worth the loss of accuracy.

We also compared the difference of computation time and accuracy depending on the number of workers used.

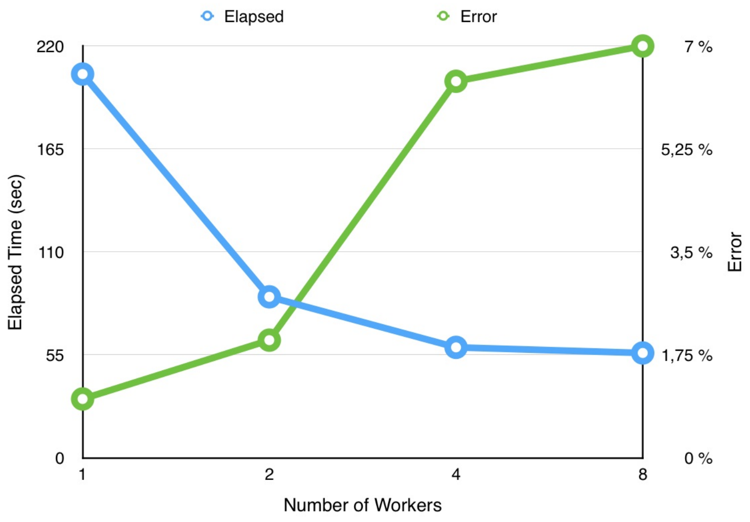

Figure 6: On Cancer Detection Dataset

5 Conclusion

We've presented the implementation of a MapReduce algorithm for Neural Networks. We've clearly shown that the computation time is improved with this technique (even locally). However we've seen that our program contains some flaws since there is a non negligible decrease in accuracy compared to the classic approach. The results they show in the paper are better (in terms of precision). However they use duplication of their data for training (they've used the Iris Dataset as well) which may over fit the model. We believe we could improve our implementation to better use the parallelization providing more desirable results. It would have been interesting to have more time in order to implement the third algorithm which is substantially different from the others. We also wanted to see how far we could push the algorithm using external servers, but we did not have the funds to do so.

6 References

Papers

[1]: Yang Liu, Jie Yang, Yuan Huang, Lixiong Xu, Siguang Li, and Man Qi, **MapReduce Based Parallel Neural Networks in Enabling Large Scale Machine Learning**, in *Computational Intelligence and Neuroscience* , August 2015, Volume 2015

Web Sources

http://spark.apache.org/docs/latest/index.html

Datasets

The iris dataset : https://archive.ics.uci.edu/ml/datasets/Iris
https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29