Parallelization of Machine Learning Algorithms Respectively on Single Machine and Spark

JIAJUN SHEN
Northwestern Polytechnical University
lonelyinnovator@mail.nwpu.edu.cn

June 16, 2022

Abstract

With the rapid development of big data technologies, how to dig out useful information from massive data becomes an essential problem. However, using machine learning algorithms to analyze large data may be time-consuming and inefficient on the traditional single machine. To solve these problems, this paper has made some research on the parallelization of several classic machine learning algorithms respectively on the single machine and the big data platform Spark. We compare the runtime and efficiency of traditional machine learning algorithms with parallelized machine learning algorithms respectively on the single machine and Spark platform. The research results have shown significant improvement in runtime and efficiency of parallelized machine learning algorithms.

I. Introduction

i. Big Data Platform Spark

We are now in the age of data explosion, every day there will be tons of data generated, which contains a great deal of valuable information. So how to dig out useful information from massive data becomes a significant problem. However, the traditional algorithms on single machine processing are insufficient to process too much data either in computing capacity or in efficiency.

Hadoop [1], a distributed big data framework emerged to solve these problems. The core design of Hadoop framework is HDFS and MapReduce [2]. HDFS provides storage for massive data, while MapReduce provides computing for massive data. Although Hadoop is efficient to speed up processing large data through parallelized computing, the MapReduce computing model is only applicable to the offline batch processing scenarios because it can not calculate data fast and in real-time.

Spark [3], the new big data framework, not only retains the scalability and fault tolerance of Hadoop MapReduce but also solves the issue of MapReduce. Spark is a fast data analysis framework based on the memory storage computing, it uses resilient distributed datasets (RDDs), which can provide interactive searching in real-time, store datasets in memory to improve reads and writes, reuse the datasets during computation and optimize iterative workloads. Therefore, Spark is more suitable for data mining and machine learning algorithms which have lots of iterations. This paper uses Spark platform to do some research on the parallelization of machine learning algorithms.

ii. Machine Learning Algorithms

Machine learning addresses the question of how to build computers that improve automatically through experience. It is one of today’s
most rapidly growing technical fields, being applied to computer vision, speech recognition, natural language processing, robot control, and other applications [4].

Machine learning offers a wide range of statistical algorithms for analysis, mining, and prediction. It includes various techniques such as association rule mining, decision trees, regression, support vector machines, and other data mining techniques. All these algorithms are computationally expensive which makes them the ideal cases for implementation using parallel architecture/parallel programming methods [5]. Therefore, it is significant to design the parallel methods for these machine learning algorithms, and this paper has done some research on the parallelization of these classic machine learning algorithms.

### iii. Algorithm Parallelization

In a single machine, different threads can run in parallel in different cores when using multithreading on multi-core CPUs. Besides, parallel computing can also be performed on multiple GPUs.

In the distributed environment, methods for parallelization can be classified into model parallelization, data parallelization, and hybrid parallelization. Model parallelization means that different machines (GPU/CPU) in the distributed system are responsible for different parts of the network model; Data parallelization is when different machines have multiple copies of the same model, each machine is assigned different data, and then the results of all the machines are combined in some way; Hybrid parallelization is a cluster with both model parallelization and data parallelization.

## II. Literature Review

### i. SVM Algorithm Parallelization

When the input data is too large, training Support Vector Machine requires a large amount of memory and will be time-consuming. Plat proposed Sequential Minimal Optimization Algorithm [6] based on the idea of divide and conquer to accelerate SVM training and reduce memory usage. However, acceleration algorithm cannot deal with the big data analysis, thus SVM parallelization has become the main research orientation. Grafs proposed the Cascade SVM [7], which splits the training dataset, and each subset trains a SVM model. Meanwhile, the adjacent SVM models are merged as the input of the next layer, and the final SVM model is obtained through continuous iteration. Hsieh proposed DC-SVM [8] based on K-Means which is used to segment data to improve the accuracy of prediction in each layer.

### ii. K-Means Algorithm Parallelization

In order to improve the data processing speed of k-means algorithm. At present, KD tree is widely used, which divides data according to the distribution of data sets and initializes central points according to the data density. Besides, many research used OpenMP, MPI and other methods to parallelize k-means algorithm. PKMeans [9] algorithm based on Hadoop MapReduce is proposed, and map, combine and reduce operations are used to implement parallelization. Farivar proposed an efficient parallel K-Means algorithm based on GPU [10].

### iii. Neural Network Algorithm Parallelization

The general parallelization method of neural network algorithm is data parallelization [11]. Multiple machines process the partitioned dataset in parallel and then synchronize network parameters between the machines. In addition, there is a method of model parallelization [12], which broadcasts unrelated neural network model subsets to each machine.
III. Single Machine Parallelization

i. Experiment Method

In this paper, the neural network algorithm is parallelized on a single machine. This paper uses PyTorch’s distributed training framework. Pytorch’s distributed training framework can be used for training either on a single machine using a multi-core CPU or multi-card GPU, or in a distributed method. In this paper, we used data parallelization to split training data. Single CPU is used as the unparallelized algorithm, and multiple CPUs are used as the parallelized algorithm to train the partitioned dataset respectively, so as to compare the training time of the unparallelized algorithm and the parallelized algorithm of neural network algorithm. The training process is as Table 1:

| Training process on multiple CPUs |
|-----------------------------------|
| **Input**: Training dataset.      |
| **Output**: Training time and loss.|
| 1. Initialize the distributed environment and set the IP address and port number of the master node. |
| 2. Initialize each process in the environment, set the number of processes, their priority, and how the processes communicate with each other. |
| 3. Split the dataset, randomly shuffle the sub-dataset, and pass it to each process. |
| 4. Each process is trained on the splitted sub-dataset respectively. When calculating the gradient, sum the gradients of all processes and calculate the average value, and then return the average gradient to all processes. |
| 5. Get the training time and loss on single CPU and multiple CPUs, and visualize the results. |

ii. Experiment Result

The experimental environment of single machine is Ubuntu 20.04. CPU is AMD Ryzen 7 4800H, 8 cores with 16 processors. GPU is NVIDIA RTX 2060. Python3.7, Pytorch1.2.

On a single machine, the MNIST dataset is trained on a single CPU first. The learning rate is 0.001, epoch is 100. The final training time is about 20.0 minutes, and the training loss was 0.09021.

After that, MNIST dataset is trained on two CPUs, and other parameters are completely the same with single CPU. The final training time is about 17.0 minutes, and the training loss is 0.090729. The visualization results are shown as Figure 1, the horizontal axis is training time, and the vertical axis is training loss:

And we also train the neural network on the single-card NVIDIA 2060. The GPU training time is about 11.7 minutes, and the loss is 0.08892. The result is shown as Figure 2.

iii. Experiment Conclusion

In the above experimental results, the training time of the neural network algorithm running in the multiple CPUs parallelization algorithm is less than that of the single CPU non-parallelization algorithm, and the final training loss is the same. Meanwhile, the training time of algorithm running on GPU is much lower than that of multiple CPUs, and the training loss is also the same.

The above experimental results show that the parallelization of the neural network algorithm can improve the efficiency of the algorithm and has no influence on the prediction accuracy of the algorithm in the single machine environment.

However, we also find that when we use too much CPU cores, the running time of the algorithm will increase. In this paper, we use 8 core CPUs with 16 processors. When the CPU cores we use are greater than 3, the CPU usage will reach 100% and the algorithm’s running time increases. So it is necessary to balance the used CPU cores and the efficiency of algorithm.
Parallelization of Machine Learning Algorithms Respectively on Single Machine and Spark

Figure 1: Time and loss of training neural network on CPUs.

Figure 2: Time and loss of training neural network on CPUs and GPU.
IV. DISTRIBUTED PARALLELIZATION

i. Experiment Method

For the distributed environment, we do some research on Spark platform to design the parallelization of machine learning algorithm. In this paper, Cascade SVM algorithm is used for parallel processing on Spark. The algorithm is as Table 2:

| Table 2: Cascade SVM on Spark |
|-------------------------------|
| **Input:** Training dataset. |
| **Output:** Training time and accuracy. |
| 1. Read the training dataset on the HDFS, and split it into m subsets randomly, and the number of training samples in each subset is the same. |
| 2. Read each subset as an RDD and make each subset to a Partition. |
| 3. Run Spark foreachPartition to train a support vector machine. Each partitioned RDD trains a support vector machine, and persists each model and related support vector machine to HDFS. |
| 4. After the training of all subsets is completed, the support vectors trained by each subset are combined in pairs using the RDD merge operation. |
| 5. Repeat steps 2 and 3 in the next layer, using the subsets to train the SVM continuously, and the support vectors are combined in pairs continuously, until the final model is obtained. |
| 6. Get the training time and the training accuracy of the final model, and visualize the result. |

Table 3: Training datasets

| name  | class | training size | feature |
|-------|-------|---------------|---------|
| a9a   | 2     | 32,561        | 123     |
| ijcnn1| 2     | 49,990        | 22      |
| covtype| 7    | 581,012       | 54      |

First, we train the SVM algorithm without parallelization using different training datasets on Spark. The training iterations is 100, and we use the default hyperparameters of SVM. Then we train the Cascade parallelized SVM algorithm. The training datasets, iterations and hyperparameters we use are the same with the non-parallelized algorithm. We can get the training time and accuracy after training and visualize the result to make a comparison shown as Figure 3 and Figure 4:

ii. Experiment Result

The experimental environment of this paper is: 5 Ubuntu20.04 virtual machines, 1 as master, 4 as slaves. Hadoop2.7.1, Spark2.4.0, Scala2.11.0, Python3.7, Java-jdk1.8. And We use a9a, ijcnn1 and covtype as the training datasets. The detail of these datasets is as Table 3:

iii. Experiment Conclusion

By comparing the running time of the non-parallel SVM algorithm and the parallel SVM algorithm on different datasets, we find that the running time of the parallel SVM algorithm is less than that of the non-parallel SVM algorithm on all datasets, which indicates that the parallelization of the SVM can indeed improve the algorithm efficiency. Moreover, when the training time of the dataset is longer, the improvement of the parallelization algorithm is greater. We find that the training time of the parallelization algorithm on the Covtype dataset is reduced a lot, while it is slightly reduced on the other two datasets.

However, we also find that the accuracy of the parallel SVM algorithm is lower than that of the non-parallel SVM algorithm. It may be caused by the unbalanced number of positive and negative categories when splitting the whole dataset into sub datasets. Although the accuracy of parallel algorithm may decrease, but the loss is slight and we can increase the number of training iterations or use other methods to cover it.
Figure 3: Runtime of training parallel SVM and non-parallel SVM on Spark.

Figure 4: Accuracy of training parallel SVM and non-parallel SVM on Spark.
V. Conclusion

In this paper, the parallelization of machine learning algorithm is implemented on a single machine and Spark platform respectively. On a single machine, this paper uses neural network algorithm and PyTorch distributed framework to compare the training time and training loss of the algorithm using single CPU without parallelization and the neural network algorithm using multiple CPUs parallelization on the same dataset. The experiment results show that the running efficiency of the neural network algorithm parallelized on multiple CPUs is higher than that of the non-parallel algorithm on single CPU, and the final training loss of both algorithms is approximately the same. At the same time, this paper also parallelizes the SVM algorithm on the Spark platform, and compares the training time and accuracy of the non-parallel SVM algorithm and the parallel SVM algorithm on different training datasets. The experiment results show that the parallel SVM algorithm has higher running efficiency than the non-parallel SVM algorithm on all datasets, and the performance of the former is only slightly lower than the latter. Above all, this paper implements parallelization of machine learning algorithms on single machine and Spark platform respectively, and proves that parallelization can indeed improve the running efficiency of machine learning algorithms on single machine and Spark platform.

References

[1] Dhruba Borthakur. The hadoop distributed file system: Architecture and design. Hadoop Project Website, 11(2007):21, 2007.

[2] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. Communications of the ACM, 51(1):107–113, 2008.

[3] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10), 2010.

[4] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. Science, 349(6245):255–260, 2015.

[5] Sujatha R Upadhyaya. Parallel approaches to machine learning—a comprehensive survey. Journal of Parallel and Distributed Computing, 73(3):284–292, 2013.

[6] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft, April 1998.

[7] Hans Graf, Eric Cosatto, Leon Bottou, Igor Dourdanovic, and Vladimir Vapnik. Parallel support vector machines: The cascade svm. Advances in neural information processing systems, 17, 2004.

[8] Cho-Jui Hsieh, Si Si, and Inderjit Dhillon. A divide-and-conquer solver for kernel support vector machines. In International conference on machine learning, pages 566–574. PMLR, 2014.

[9] Weizhong Zhao, Huifang Ma, and Qing He. Parallel k-means clustering based on mapreduce. In IEEE international conference on cloud computing, pages 674–679. Springer, 2009.

[10] Reza Farivar, Daniel Rebolledo, Ellick Chan, and Roy H Campbell. A parallel implementation of k-means clustering on gpus. In Pdpta, volume 13, pages 212–312, 2008.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25, 2012.
[12] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc’auerlio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.