Training Backpropagation Neural Network in MapReduce

Zhou Binhan, Wang Wenjun, Zhang Xiangfeng
School of Computer Science and Technology, Tianjin University, Tianjin, China
{binhan198 & xfzhang}@tju.edu.cn, wwj@pku.org.cn

Abstract - BP neural network is generally serially trained by one machine. But massive training data makes the process slow, costing too much system resources. For these problems, one effective solution is to use the MapReduce framework to do the distributed training. Some methods have been proposed, but it is still very slow when facing the neural network with complex structure. This paper presents a new method for BP neural network training based on MapReduce, MR-TMNN (MapReduce based Training in Mapper Neural Network). This method puts most of the training process into Mappers, and then emits the variations of weights and thresholds to Reducer process to do the batch update. It can effectively reduce the volume of intermediate data created by Mappers, reducing the cost of I/O, thereby accelerating training speed. Experimental results show that MR-TMNN has a better convergence without losing too much accuracy, comparing with conventional training method, and it still performs well with the complexity of neural network structure increasing.

Index Terms - MapReduce, backprogation, neural network, intermediate data.

1. Introduction

Neural network is always a hotspot in the field of AI since its appearance last century. It can solve a series of problems of machine learning including a variety of non-linear data analysis. There exists many kinds of neural network training methods and BP (Backward propagation) algorithm has a significant effect and is the most commonly used training algorithm. Its basic idea is gradient descent and BP neural network is generally serially trained. With the rapid development of information times, the amount of training data skyrocketing, the shortcomings of conventional training method have been exposed. More and more people begin to explore a distributed approach to train neural networks in order to reduce the training cost. This paper introduces a new distributed training method, which combines the widely used framework for bigdata, MapReduce with the basic structure of BP neural network, called: MR-TMNN (MapReduce based Training in Mapper Neural Network). It is effective to big training data and has robustness against complex structure of neural networks. This paper uses hadoop as the implement of MapReduce framework.

The outline of the paper is as follows: Section II discusses the related research. Section III presents detailed description of the algorithm. Section IV introduces the experiments and section V draws a conclusion.

2. Related Research

In fact, two decades before MapReduce's appearance, many people have begun to explore distributed training and made some concepts. There are two kinds of weight update modes: incremental mode and batch mode [1]. In incremental mode, neural network is updated after each training item; in batch mode, it is updated once after all the training items are presented. The former one always can't begin the next training until the previous training is completed, which makes it not suitable for parallel implementation. So for distributed training, the latter is always used [1][2]. There are also two kinds of parallelization approaches: data parallelization and node parallelization [1]. In node parallelization, the nodes of neural network are mapped onto the nodes of clusters [3]. Its disadvantage is the data transmission will become a bottleneck. In data parallelization, the whole training data is divided into many small blocks of dataset and each node process some blocks. After all the nodes finish its training, results will be collected and used to do batch update [2].

For MapReduce, batch mode and data parallelization is used, because on the one hand, we always have to deal with a very big training data, on the other hand, we can't control the MapReduce nodes in this framework exactly. Existing MapReduce based training method reads one item into Mapper and completes training for once, and then collects and calculates average weights and thresholds influence in Reducers. After that, it then does batch update [4][5]. However, when facing neural network with complex structure, it will create very big intermediate data even bigger than training data, which makes the reduction of training efficiency. We will see details in the following experiments. In reference [6], local iteration is added and the original method has been improved someway, but it does not address the problem of large intermediate data. The method proposed by this paper will effectively solve this problem, speed up training process, with robustness against different neural network structures.

3. MR-TMNN: MapReduce based Training in Mapper Neural Network

For MapReduce, There exists three parts: Mapper process, Reducer process and User process [7]. The big training data will be divided into many small blocks of dataset when uploaded to HDFS. Each block will be delivered to a Mapper to finish a map task. When all the map tasks have been finished, then the Reducer process begins. The intermediate key-value pairs after Mapper process will be sent to Reducers to finish reduce tasks. After Reducer process, final result will be written on HDFS and the program will back to User process. MR-TMNN is a data parallelization method. Every map task has the same integral neural network. The block delivered to this Mapper is only used to train the local neural network, which is the neural network in this map task. Then

the training result, that is the weights and thresholds, will be emitted. Reducers receive all the weights and thresholds from map tasks as input values, and then calculate the total influence and save the result on HDFS. In User process, user function downloads the result and processes the batch update. The details are as follows.

A. Mapper Process

Mapper process can be divided into three parts: setup function, initial the weights and thresholds for this task; map function, train the local neural network with the delivered block of dataset; cleanup function, emit the training result, including weights, thresholds and the number of training items. The intermediate key-value pair is as <weight, weight value_number> or <threshold, threshold value_number>.

Setup function is called when map task begins and cleanup function is called after the training is finished. We record the number of training items because we can't ensure all the block of datasets delivered to map tasks are the same. So if the block is smaller, the local neural network in this map task has been trained for fewer times, and the weights and thresholds are considered less important.

B. Reducer Process

Reducers receive the result emitted by Mappers and do weighted average, then save them on HDFS.

Here is the pseudo code, Mapper and Reducer process:

```
Input: initial Weight W0, Threshold Q0; Data subsets
Output: key; Weight W, Threshold Q, item number C
Mapper{
   State Weight W, Threshold Q, item number C
   setup (W0, Q0){
         For(All the Weight W and Threshold Q){
              W = W0, Q = Q0, C = 0
         }
   }
   map (Data subsets-key/value){
         Training neural network
         Fix W, Q & Let C plus 1
   }
   cleanup (){
         Emit (Weight, W_C), Emit (Threshold, Q_C)
   }
}
```

```
Input: Weight, W_C or Threshold, Q_C
Output: W or Q after weighted average
Reducer{
      for (all values){
          Collect all the influence of weights or thresholds
          Collect all the number of training items
      }
      Weighted average the collected results
      Write the weight or threshold after average on HDFS
}
```

C. User Process

Download the result about the weights and thresholds, and then batch update the neural network in User process. If necessary, continue the next training.

Up to now, we know that the number of map tasks is the same as the number of blocks of dataset that divided. In hadoop, the default block volume is 64M and every map task creates only one group of intermediate key-value pairs, which is emitted by cleanup function before the task is completed. That is every 64M data creates one group of intermediate key-value pairs. But some previous methods, one training item creates one group of intermediate key-value pairs. When the number of weights and thresholds of the neural network is much more than the dimension of the input training item, the intermediate data created by Mapper process is even bigger than the input training data. Through a comparative analysis, we can see MR-TMNN effectively reduce the volume of intermediate data, reducing the pressure of I/O and accelerating training speed.

4. Experiments

A. Experimental environment

All Experiments in this paper are carried out on ordinary computers, the configuration is as follows: Dell OptiPlex 790 with 64-bit ubuntu, 4G memory, 1T hard disk. One of the seven computers acts as the namenode and jobtracker, and the rest of them act as datanodes and tasktrackers. The hadoop we adopt is 1.0.4 with default HDFS configuration.

B. Experiment on convergence

A small experiment can expose the problem in MBNN [4]. The dataset used in these experiments is randomly generated, 300000000 items, 9.1G. The structure of neural network is 4-4-1, that is 4 input nodes, 4 hidden nodes and 1 output node, and all the initial weights and thresholds are also randomly generated. After the first training with the data, we have found that serial training using stand-alone machine spends 319 seconds totally for the neural network; MR-TMNN spends 134 seconds, and MBNN spends almost 4 hours. The big intermediate data after map perhaps is the reason for the slowness, because for one input training item which contains 5 numbers, we should record the change of all the weights and thresholds after one map, totally 25 numbers, the intermediate item is almost 5 times bigger than the training item. Oppositely, MR-TMNN can effectively reduce the big intermediate data, accelerating training speed. We will ignore the experiments on MBNN later.

Fig.1 shows the detailed information. We can see for the data, training for the same M times, MR-TMNN is faster than serial method, but it doesn't mean the convergence rate is better, so we analysis the convergence rate under different learning rates. It is calculated after the whole data is trained for M times. Equation (1) describes the convergence error. N is the number of training items of the dataset, X_{ti} is the excepted result and X_{pi} is the predicted result. Fig.2 and Tab.1 show the details.

TABLE I. convergence rate of MR-TMNN and stand alone in different training times and learning rates

Training methods M training times	Learning rate		
MR-TMNN	0.01	0.02	0.05
1	28892.237	28892.84	28893.59
2	28892.125	28892.56	28893.42
3	28892.056	28892.42	28893.4
4	28892.01	28892.34	28893.39
5	28891.978	28892.28	28893.38
6	28891.955	28892.25	28893.38
7	28891.938	28892.22	28893.38
8	28891.925	28892.19	28893.37
stand-alone	0.01	0.02	0.05
1	28894.45	28892.46	28957.7
2	28894.377	28892.47	28957.63
3	28894.352	28892.47	28957.6
4	28894.34	28892.47	28957.59
5	28894.332	28892.47	28957.58
6	28894.326	28892.47	28957.58
7	28894.322	28892.47	28957.57
8	28894.318	28892.47	28957.57

$$\sigma = \sqrt{\sum_{i=1}^{N} (X_{ti} - X_{pi})^2 / N} \qquad (1)$$

We can find that comparing with serial training using stand-alone computer, MR-TMNN is better on convergence rate under particular learning rate when training for the same M times.

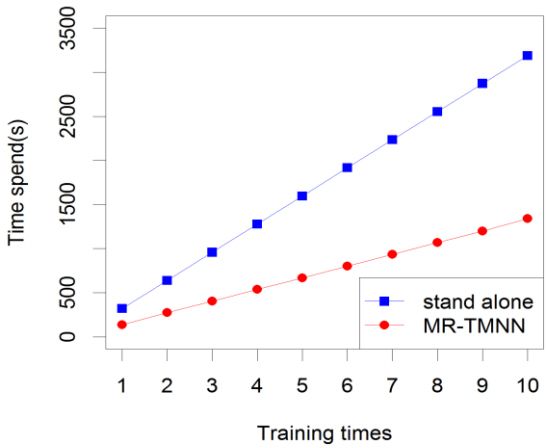

Figure 1. Time cost by MR-TMNN and stand alone with M times' training

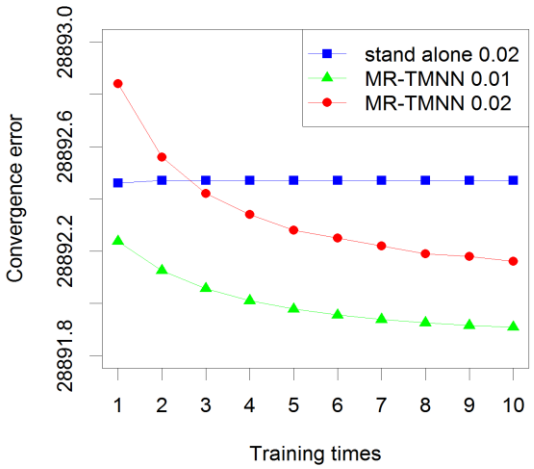

Figure 2. Parts of convergence curve of MR-TMNN and stand alone in different learning rates

C. *Experiment on scalability*

The purpose of this experiment is to find out the affection to MR-TMNN with different neural network structures and MapReduce working nodes. We compare it with serial method separately under different hidden nodes and MapReduce working nodes. First, we reduce hidden nodes, and Fig.3 shows the details. Then we reduce working nodes, Fig.4 shows the result on speedup.

Based on this experiment, we discover that MR-TMNN loses less training rate and is more robust when facing different neural network structures. The speedup of MR-TMNN is also nearly linear under different working nodes.

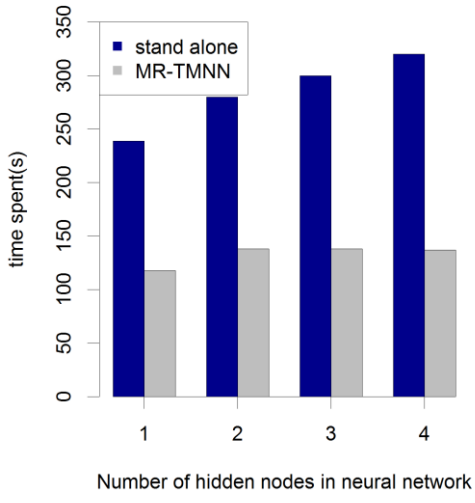

Figure 3. Time cost by MR-TMNN and stand alone to train the neural network with different hidden nodes, using 6 MapReduce working nodes to train once

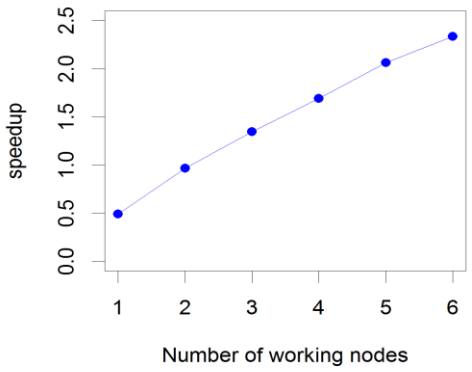

Figure 4. Speedup with different MapReduce working nodes, training the neural network with 4 hidden nodes for once

D. *Experiment on effectiveness*

Finally, we use the dataset collected and processed from GPS to demonstrate the effectiveness of MR-TMNN. It is the average vehicle's speed of a road in Tianjin province recorded every 5 minutes. Total recording time is 2 months and we use the first month's (8060 items) as training data and the second month's (8636 items) as validation data to do the time series prediction, that means we use the former 20 minutes' speed to predict the next 5 minutes' speed. Fig.5 shows the convergence curve under different training times.

We can figure out that after the first training, the convergence error has been almost stable and very close to the condition generated by serial method. But for the time spent, as mentioned, MR-TMNN is more efficient.

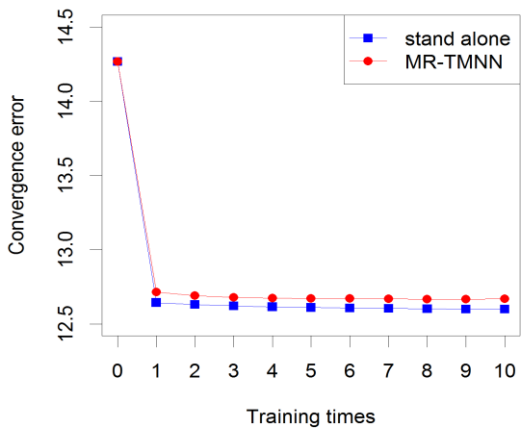

Figure 5. Convergence rate of MR-TMNN and stand alone

5. Conclusion

In this paper, we propose a new method called MR-TMNN for neural network training based on MapReduce. Comparing with the conventional training method as well as existing MapReduce based training method, MR-TMNN effectively solves the problem on big intermediate data, accelerating training speed without losing too much accuracy. When facing neural network with complex structure, it also exhibits great robustness.

Acknowledgment

This research is supported by Tianjin Key Laboratory of Cognitive Computing and Application. And also we would like to express our gratitude to the National Science and Technology Pillar Program (key technology and application research of the safe construction of urban emergency response system of prevention and control model based on the Internet of Things 2013BAK02B06, research and application of intelligent judgment model based on dynamic video and public information technology 2013BAK02B00) and the National Natural Science Funds Major Research Plan Support project of emergency management research on unconventional events (top design and model reconstruction of emergency management system with the new era Chinese characteristics 91224009).

References

[1] Novokhodko Alexander, and S. Valentine. "A parallel implementation of the batch backpropagation training of neural networks." Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on. Vol. 3. IEEE, 2001.

[2] D. Hwang, C. Xu, F. Fotouhi, "A Parallel Backpropagation Learning Algorithm for Urban Traffic Congestion Measurement," in Intelligent Engineering Systems Through Artificial Neural Networks, 1999, vol. 9, pp. 75-80.

[3] Zickenheiner S, Wendt M, Klauer B, Waldschmidt K. "Pipelining and parallel training of neural networks on distributed-memory multiprocessors." Neural Networks, 1994. IEEE World Congress on Computational Intelligence. 1994 IEEE International Conference on. Vol. 4. IEEE, 1994.

[4] Liu Zhiqiang, Hongyan Li, and Gaoshan Miao. "MapReduce-based backpropagation neural network over large scale mobile data." Natural Computation (ICNC), 2010 Sixth International Conference on. Vol. 4. IEEE, 2010.

[5] Chu C., Kim S. K., Lin Y. A., Yu Y., Bradski G., Ng A. Y., et al. "Map-reduce for machine learning on multicore."Advances in neural information processing systems 19 (2007): 281.

[6] Chenje Zhu, and Rao Ruonan. "The Improved BP Algorithm Based on MapReduce and Genetic Algorithm." Computer Science & Service System (CSSS), 2012 International Conference on. IEEE, 2012.

[7] Dean Jeffrey, Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.