Contents lists available at YXpublications

International Journal of Applied Mathematics in Control Engineering

Journal homepage: http://www.ijamce.com

Spark-based Backpropagation Neural Network Algorithm

Suqi Zhang[a,*], Shiyao She[b], Junhua Gu[b]

[a]*School of Information Engineering,Tianjin University of Commerce,Tianjin 300134,China*

[b]*School of Artificial Intelligence ,Hebei University of Technology,Tianjin 300401,China*

ARTICLE INFO

Article history:
Received 23 January 2020
Accepted 1 March 2020
Available online 1 March 2020

Keywords:
Large-scale data;
Backpropagation neural network;
Spark;
SBPNN;

ABASRACT

Abstract: Backpropagation neural network has been widely used in the fields of data mining and pattern recognition. The research shows that with the increase of training data, the accuracy of the model produced by a neural network can be improved. However, the iterative computation of large-scale data results in the low-speed process, which restricts its application in big data. In order to solve the problem of time-consuming of backpropagation neural network in training large scale data, this paper proposes a spark-based parallel backpropagation neural network algorithm called SBPNN. The experiment verifies the algorithm from the aspect of convergence, parallelism and high efficiency. It proves that the algorithm has good convergence and parallelism. Compared with the MBNN, the algorithm is faster and more suitable for training large scale data.

1. Introduction

As the theory and technology have developed rapidly in computational intelligence in recent years, ANN (Artificial Neural Network) has been widely used in data mining and pattern recognition applications. ANN can approximate the non-linear function with arbitrary precision based on the relationship between neurons in each layer, and it has high fault-tolerance and self-adaptability. Its performance can be determined by structural features and neuron characteristics inside the neurons. ANN has solved the problem of data analysis in a variety of fields. In many algorithms of artificial neural network, backpropagation neural network (BPNN) can adjust the weights and thresholds of the network according to the error backpropagation, with good function approximation function and generalization abilities. The network structure is simple and easy to deploy to computers, so it has been widely used in classification, image processing, and function prediction, etc. [1], it is the most widely used artificial neural network at present. In 2019, S W Zhang and P Wang [2] proposed a method for precise tracking of aircrafts which using BP neural network takes the TDOA measurements as input and the data trained could quickly fix target locations. In 2019, J Yao [3] proposed a method that the deep learning method built on CNN can be applied to the arteriovenous image reconstruction of a multi-electrode electromagnetic flowmeter. In 2019, W X Du [4] proposed that fault information of multiple sensors is decomposed into the sum of multiple intrinsic modal functions by the method of empirical mode decomposition and the neural network fault diagnosis model is established through training.

In the artificial neural network, the prediction results produced by training over large scale data are more accurate. Therefore, in application of BPNN, it is necessary to training over large scale data to ensure the accuracy of prediction results. However, the large amount of computation of network parameters in iterative training leads to low-speed training process, which restricts the use of neural networks in big data.

In early times, researchers tended to use special hardware to reduce training time for neural networks, such as neuro-hardware and neurocomputer, but they offer little flexibility and scalability [5].

In recent years, many researchers apply neural networks to distributed platforms for parallel training. In 2006, Y. Bo and W. Xun[6] proposed that in the grid computing model, distributed neural networks with data parallelism have better computational performance. In 2007, C. Chu, S. Kim [7] proposed the parallelization of BPNN on multi-core processors. However, both papers focus on the parallelization of neural networks without verifying the validity of parallelism on large scale data. In 2010, Z. Liu and G. Miao [8] proposed a MapReduce-based Backpropagation Neural Network (MBNN) based on the MapReduce framework. However, the iterative training process in BPNN will cause frequent data transfer between Map and Reduce, and the MapReduce framework is only based on the HDFS (Hadoop distributed file management system) for data transfer, so frequent reading and writing spend a lot of time. In 2013, R Gu and Y Huang

* Corresponding author.
E-mail addresses: zhangsuqie@163.com (S. Zhang)

[9] proposed a parallel computing technique for distributed training neural networks—CNeural. CNeural stores large-scale training data on HBase to achieve memory-based distributed computing. However, it provides little support in fault tolerance. If a fault occurs during iterative training, the whole training process would be restarted in 2011 Zaharia and M. Chowdhury [10] proposed spark which is a memory-based distributed platform, and they pointed out that spark is more suitable for iterative algorithms in machine learning with high fault tolerance. In 2016, Liu Y and Xu L proposed [11] the PBPNN (Parallelization of Backpropagation Neural Network), which is parallelization of BPNN in spark based on the spark platform. In classification problem, they implemented the parallel training of BPNN on the spark platform, but the training and prediction were carried out randomly. Therefore, a complete neural network model was not established after the training.

In conclusion, BPNN in distributed platform for training over large scale data needs further study. Therefore, based on the previous research results, this paper proposes Spark-based Back Propagation Neural Network algorithm (SBPNN), which can solve the problem of time-consuming when training over large scale data. After the training, it can establish a complete neural network model for prediction. The algorithm was experimentally implemented on the spark platform using large scale data. First of all, we train the algorithm in Mini-batch update mode with different numbers of training samples l and analyze the convergence of the SBPNN. Then, we calculate the speedup ratio by using the different cluster sizes to verify the parallelism of the SBPNN. Finally, on the time efficiency, comparisons with MBNN (the parallel backpropagation neural network algorithm based on MapReduce framework under Hadoop platform) verify the high efficiency of the SBPNN.

2. BACK PROPAGATION NEURAL NETWORK

Backpropagation neural network is one of the most widely used machine learning algorithm [12], it was proposed by a group of scientists headed by Rumelhart and Mc Celland in 1986. The research shows that the three-layer BPNN can approximate any non-linear continuous function with arbitrary accuracy [13]. This paper illustrates the concrete training process by three-layer BPNN as an example. As shown in Figure 1, BPNN is a three-layer neural network model, including input layer, then hidden layer, and hidden layer connected with output layer. In BPNN, the number of neurons in input layer and output layer is determined by the input vector of training data and the number of elements in output vector. The neurons in the same layer are not connected, and the neurons in two adjacent layers are full connection.

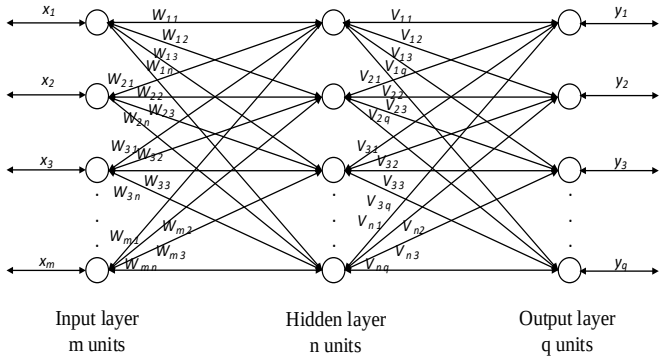


Fig.1. The structure of BPNN

BPNN adjusts the weights and thresholds by the gradient descent method. The whole training process includes forward and back propagation. Forward propagation refers to that input data from the input layer, the middle of the hidden to spread the data to the output layer. If there is error in actual output result and the expected output results, the back propagation is carried out. The error is input from the output layer, the error is propagated backward layer by layer through the hidden layer to the input layer, and the weight and threshold of the network are adjusted. The training work continues until the mean square error meets the specified threshold or network iteration number reaches the limited round.

Suppose the total number of training samples is N, the number of neurons in the input layer is denoted by m, and $x_i(i=1, 2....m)$represents the ith input vector; the number of neurons in the hidden layer is denoted by n, u_j and $h_j(j=1, 2....n)$respectively represent the input data and output data of the jth neuron in the hidden layer; the number of neurons in the output layer is denoted by q, l_k and $c_k(k=1, 2....q)$denote the input data and the output data of the kth neuron in the output layer, t_k denotes the expected output of the kth neuron in the output layer, d_k denotes the output error of the kth neuron in the output layer, o_j denotes the error of the jth neuron in the hidden layer error; w_{ij} denotes the weight between the ith neuron in the input layer and jth neuron in the hidden layer, v_{jk} denotes the weight between the jth neuron in the hidden layer and kth neuron in output layer, α_j and β_k respectively denote the thresholds of the jth neuron in the hidden layer and the kth neuron in the output layer, and η is the learning rate factor.

2.1 ONLINE UPDATE MODE

The online update mode is to update network weights and thresholds after every training sample is calculated. As shown in Figure 2, N times of weights and thresholds are updated in one iteration process, and the specific implementation process is as follows.

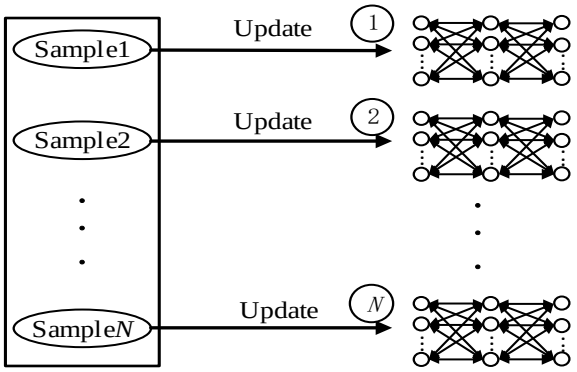


Fig. 2. Online update mode training process

1. Input a training sample $\boldsymbol{x}=(x_1,x_2,\ldots\ldots,x_m)$into the input layer and calculate the input u_j and output h_j of each neuron in the hidden layer according to formula (1) (2), where h_j is calculated by the sigmoid activation function, it can limit the range of function values is [0,1].

$$u_j = \sum_{i=1}^{m} w_{ij} x_i + \alpha_j \qquad j=1, 2....n \qquad (1)$$

$$h_j = \frac{1}{1+e^{-u_j}} \qquad j=1, 2....n \qquad (2)$$

According to the output data of each neuron in the hidden layer, input l_k and output c_k of each neuron in output layer are calculated according to formula (3) (4).

$$l_k = \sum_{i=1}^{n} v_{jk} h_j + \beta_k \qquad k=1，2....q \qquad (3)$$

$$c_k = \frac{1}{1+e^{-l_k}} \qquad k=1，2....q \qquad (4)$$

2. Calculate the error of neurons in the output layer and the hidden layer according to the formula (5) (6) according to the output data of each neuron in the output layer.

$$d_k = c_k(1-c_k)(t_k - c_k) \qquad k=1，2....q \qquad (5)$$

$$o_j = h_j(1-h_j)\sum_{k=1}^{q} d_k v_{jk} \qquad j=1，2....n \qquad (6)$$

3. According to the errors of neurons in the output layer and the hidden layer, the change value of weights and thresholds are calculated according to formula (7) (8).

$$\Delta v_{jk} = d_k c_k \qquad \Delta\beta_k = d_k \qquad k=1，2....q \qquad (7)$$

$$\Delta w_{ij} = o_j h_j \qquad \Delta\alpha_j = o_j \qquad j=1，2....n \qquad (8)$$

4. Update the network according to formula (9) (10) based on the change value weights and thresholds.

$$v_{jk} = v_{jk} + \eta\Delta v_{jk} \quad \beta_k = \beta_k + \eta\Delta\beta_k \qquad k=1，2....q \qquad (9)$$

$$w_{ij} = w_{ij} + \eta\Delta w_{ij} \quad \alpha_j = \alpha_j + \eta\Delta\alpha_j \qquad j=1，2....n \qquad (10)$$

5. Select the next training sample, and repeat the calculation of 1 to 4 steps, until all the training samples are completed. When the network completes an iterative training, perform the next step.

6. Calculate the mean square error of the whole training data set, and the training work continues until the mean square error meets the specified threshold or network iteration number reaches the limited round, then output neural network model Array[*w*]. Otherwise return to the first step to continue.

2.2 MINI-BATCH UPDATE MODE

Mini-batch update mode is to divide the whole *N* training samples evenly into several batches. Each batch has *l* training samples, and when *l* = *N*, it is the full batch updating mode. The training samples of each batch are calculated according to the steps 1 ~ 3 of 2.1, and obtain the change value of weights and thresholds form *l* samples. Afterward, the weights and thresholds are updated according to formula (11) (12) to achieve the 4th step, and the next batch of training samples will be calculated. As shown in Figure 3, once iteration of the mini-batch update mode updates *N* / *l* times weights and thresholds [14].

$$v_{jk} = v_{jk} + \eta\frac{\sum\Delta v_{jk}}{l}$$

$$\beta_k = \beta_k + \eta\frac{\sum\Delta\beta_k}{l} \qquad k=1，2....q \qquad (11)$$

$$w_{ij} = w_{ij} + \eta\frac{\sum\Delta w_{ij}}{l}$$

$$\alpha_j = \alpha_j + \eta\frac{\sum\Delta\alpha_j}{l} \qquad j=1，2....n \qquad (12)$$

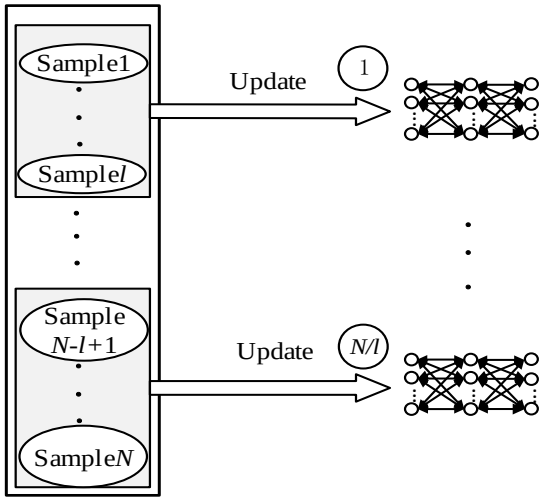


Fig.3. Mini-batch update mode training process

The online update mode is that every iteration of network training needs to update the weight and threshold *N* times, which consumes a long time. Each update of weights and thresholds is calculated based on the previous training sample, and it will have a certain influence on the training results. The later calculation of training samples on the network will cover the results of the previous calculation. When the training data set is large, there is low accuracy; thereby, it is easy to fall into local optimal solution. The mini-batch update mode avoids the influence of the input order of training samples on the training results, and ultimately solves the optimal global solution and reduces the time spent on each iteration [15].

In the online update mode, every update is the result of the previous training sample calculation, and the calculation of the next training sample is also performed on the network after the last update. Therefore, the online update mode is not easy to implement the parallelization of training samples calculation. The mini-batch update mode is to update the network after a batch of training samples are calculated, and parallel computation of a batch can be implemented on multiple processors. Therefore, this paper adopts mini-batch update mode to realize SBPNN.

3. SBPNN

3.1. Spark

Spark is a memory-based, distributed parallel computing platform that takes full advantage of the Hadoop platform and the MapReduce framework. Besides, intermediate results from Spark operations can be stored in memory without the need to read and write HDFS and improve parallel computing speed, so the spark is more suitable for data mining and machine learning, especially for iterative algorithm [16]. The Spark cluster starts with a master node and several worker nodes. The master node is mainly responsible for the management of the cluster resources. The worker node is for data calculation [17]. Spark workflow is shown in Figure 4, when the master node uses the spark-submit command to submit jobs, it first starts a driver process in the local client. The driver process will be set according to the parameters of the master node to apply for the corresponding cluster resources, such as the number of worker nodes, the size of executor's memory, and the number of CPU cores on each worker node. The master node communicates with the worker node, informing the worker node to start the

executor and registering with the Driver process; the driver process being connected to the worker node and the tasks to be executed are assigned to the various worker nodes in the cluster, the worker node reads the data from HDFS according to the task allocation and caches it into memory, and the driver process collects and summarizes the results processed by each worker node.

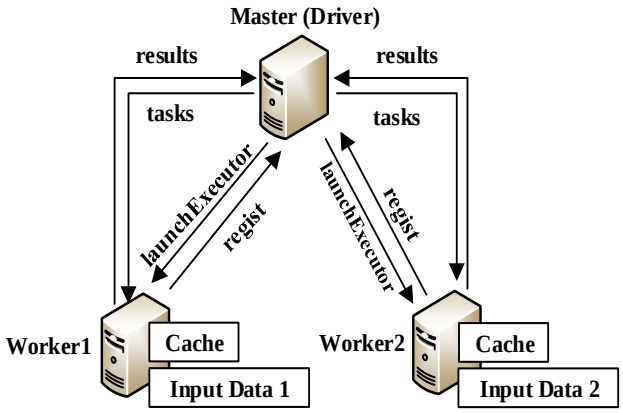


Fig.4. Spark workflow diagram

3.2. SBPNN algorithm flow

The neural network is a typical iterative algorithm, and it adjusts the neural network model if calculation data stored in memory can greatly improve the algorithm iteration speed [18]. Therefore, the neural network algorithm is more suitable for parallel model building on the Spark platform.

There are two parallelization approaches of the neural network, node parallelism, and training data parallelism. The node parallelism divides the neurons of neural networks into different worker nodes for processing [19]. Each worker node carries out data communication in the process of calculation to realize parallel training. Training Data parallelism divides the training samples into different worker nodes [20]. Each worker node has a complete neural network structure and calculates the training samples on the local machine so as to realize the parallel computing of the training samples. The SBPNN algorithm in this paper adopts the mini-batch update mode. Therefore, it is more appropriate to implement parallel training by training data parallelism.

Before carrying out the training of SBPNN, we need to define the structural information of the network, including the number of neural networks, the number of neurons per layer, the initial weights and thresholds of the network, the learning rate factor, and the size of the value of l. First, the master node initializes the neural network structure information and broadcasts to the worker nodes in the cluster so that each worker's memory stores a complete neural network, and the initial state of the network is the same. In the SBPNN training, each worker node gets part of the training data set, and multiple workers calculate Δw(Δw represents the Δw、Δv、$\Delta \alpha$、$\Delta \beta$)of each training sample in parallel according to formula (1) to formula (8), and then update the weights and thresholds according to formula (11) (12) on the master node. The Master node and the Worker nodes respectively perform different tasks. Figure5 illustrates the parallel implementation process of the SBPNN algorithm by taking two worker nodes as an example. Specific steps are as follows:

a. Master node defines BP neural network structure information, and worker nodes read training data set from HDFS;

b. The master node broadcasts the defined BP neural network structure information to each worker node so that a complete BP neural network can be instantiated on each worker node;

c. Conduct neural network training in parallel on each worker node to calculate the change value of each training sample's weight and threshold Δw;

d. Worker nodes return the weight and threshold change value Δw to the master node;

e. The master node integrates Δw of each worker node, updates the weights and thresholds of the neural network, and judges whether all batches of training samples have been learned, and proceeds to the next step; otherwise, returns to step *b*;

f. judging whether the mean square error of the whole training data meets the requirement or the number of iterations of the network reaches the upper limit, if yes, ending the training; otherwise, returning to step *b* to continue the training;

g. The training is finished, and the trained neural network model Array[w] is output.

3.3 Parallel Conversion of Data States in SBPNN Algorithm

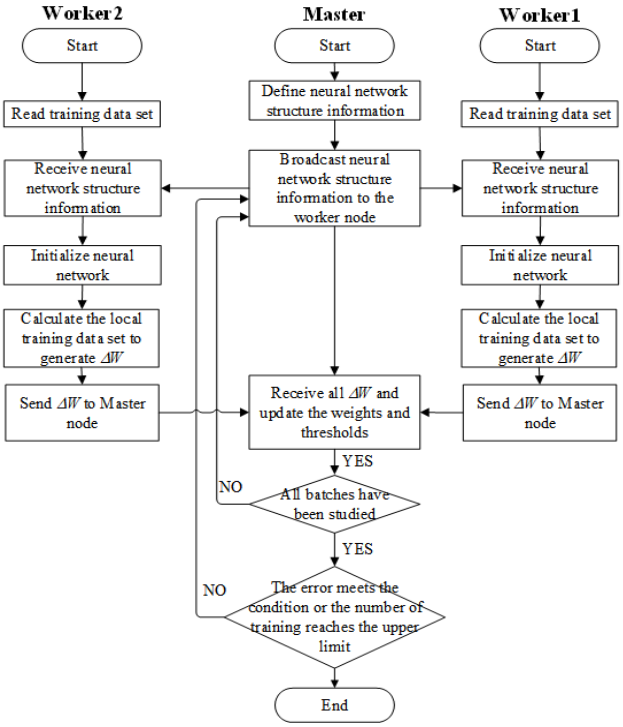


Fig.5. The parallel training algorithm of SBPNN

The advantage of the Spark platform is based on memory computing, RDD (Resilient Distributed Data) is an abstraction of distributed memory in the Spark platform, mainly from the creation of distributed files on HDFS or from the other RDDs [21]. RDD is divided into multiple partitions. Each partition is located in different nodes in the cluster so that the data in the RDD can be parallel operations. In order to achieve SBPNN used by various RDD operators, operations are carried out in memory. Figure 6 shows two worker nodes as an example of the parallel conversion process in the parallel training phase of SBPNN. The details are as follows: First, the worker nodes read the training data from HDFS and

caches it in memory, and then uses the parallelize method provided by the Spark environment Spark Context to convert the training data into RDD marked as RDD1. Before calling RDD1.map, the output value of each neuron in the output layer is calculated, and the return value is recorded as RDD2. Then the RDD2.map operator is used to achieving back propagation, and the error of each neuron in the output layer and the hidden layer is calculated. The return value is RDD3, then RDD3.map operator calculated the Δw, the return value recorded as RDD4, and finally the RDD4.tree aggregate operator to each partition aggregates, and returns an updated neural network model Array[w]. The initialization neural network structure information involved in the implementation process is broadcast by the master node to each worker node by using the broadcast operator. The neural network model Array[w] is also rebroadcast after each update.

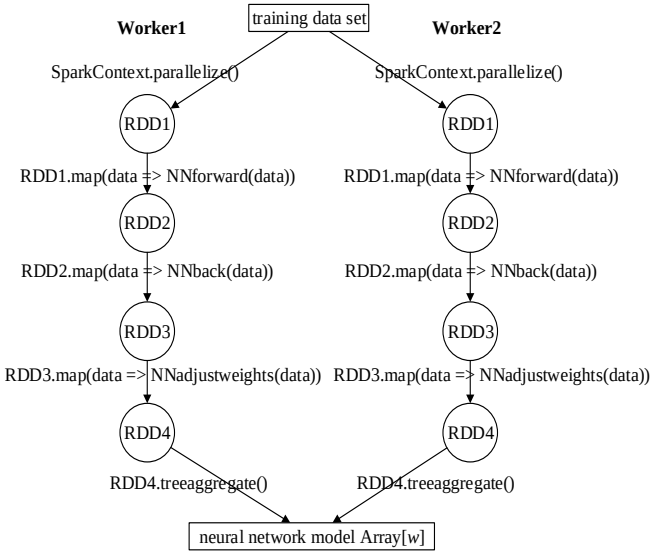


Fig. 6. SBPNN data state parallel conversion diagram

4. Analysis of results

In order to verify the performance of the SBPNN algorithm, the experiment is mainly divided into three parts. First, the error accuracy is used as the training end condition in mini-batch update mode. The convergence of SBPNN was analyzed by the number of iterations and training duration at the end of training under different l; Then, take $l = N$, train with different training data, calculate the speedup ratio to analyze the parallelism of SBPNN; Finally, take $l = N$ as the same, compare with MBNN(a parallelized BP neural network algorithm based on Hadoop platform)to verify the effectiveness of SBPNN.

The experiment was implemented on Spark2.0.0 and Hadoop 2.4.1 clusters. The cluster environment consisted of 6 nodes, including 1 master node and 5 worker nodes. Each node was configured identically and located in the same local area network. The operating system was CentOs6. 5, CPU is E5-2620 v4, the core frequency is 2.10GHZ, the node memory is 32GB.

In this paper, the supersymmetry particle data set SUSY was selected as the experimental data from the UCI machine learning database of the University of California. The data has a total of 5 million samples, accounting for 2.2G of storage space. Each of these data contains 18 feature items and 1 category identifier item, which is a typical data to solve the classification problem using neural network algorithm. This paper uses three layers of BP neural network. The number of neurons in input layer m = 18, the number of neurons in hidden layer n = 10, and the number of neurons in output layer q = 1.

4.1. Analysis of SBPNN Convergence

BP neural network is one of the iterative algorithms. The convergence of the iterative algorithm shows whether or not to converge and the convergence rate [22]. The good iterative algorithm has a faster convergence rate under stable convergence. In this experiment, the mean square error of the whole training data is less than or equal to 0.075 in the end condition of SBPNN, and the convergence rate of SBPNN is analyzed by the number of iterations and training duration of SBPNN under different l. The experimental results are shown in Figure 7 and Figure 8.

In figure 7, the abscissa indicates the number of iterations, and the ordinate indicates the mean square error. The four lines in the figure represent the trend of mean square error under four different l. It can be seen from the figure that when different l for training is used, the mean square error shows a gradual downward trend, which shows that SBPNN has a stable convergence in the training large scale data.

In figure 8, the abscissa indicates the mean square error, and the ordinate indicates the training time. The five lines in the figure represent the training time when achieving different mean square errors at five values of l. When the mean square error is 0.075, the training time of l = 80000 is the shortest, and the training time of l = 5000000 is the longest. Thus, for large scale training data, mini-batch update mode to take the appropriate l can make a faster convergence rate.

The smaller training means square error leads to a better training result. When l is too small, the training data will be divided into many batches. Each batch has fewer training samples, and it cannot reflect the characteristics of the whole data, although the number of iterations less. Each iteration takes a longer time, the total training time will be longer. When l is too large, each batch contains more training samples, the number of update times on each iteration will be reduced, it makes each iteration takes less time, the result of learning is more similar to the optimal global solution, but the mean square error produced by each iteration decreases slowly, thereby, more iterations are needed to get a better model. This makes the overall training time longer. Thus, under the large-scale training data, taking the appropriate l training for SBPNN can achieve the optimal training time, which makes SBPNN have better convergence speed.

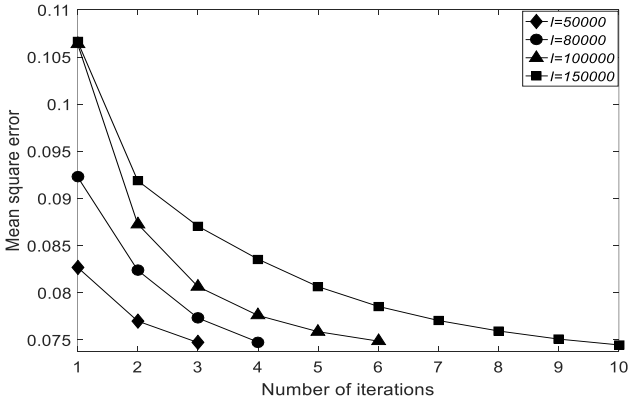


Fig. 7. Different l value of the number of iterative comparison chart

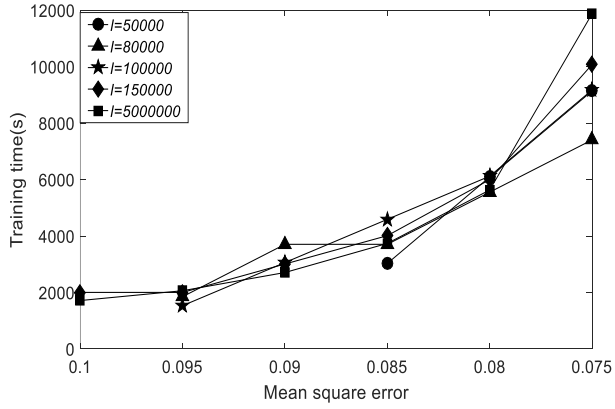


Fig.8. Different l value training time comparison chart

4.2. SBPNN parallelism analysis

The experiment verifies the parallelism of the algorithm by changing the size of the training data, counting the training duration under different nodes to calculate the speedup. The speedup is calculated, as shown in Equation (13).

$$S_p = \frac{T_1}{T_p} \tag{13}$$

In the above formula, T_1 represents the training duration when using 1 worker node, T_p represents the training duration when using p worker nodes, and S_p represents the acceleration ratio, which indicates the efficiency increase after parallelization. Among them, when $S_p = p$, it is the linear acceleration ratio, and the parallelism of the algorithm is best [23].

The experimental results are shown in Figure 9, the abscissa is the number of worker nodes, and the ordinate is the acceleration ratio. The three lines in the figure are the speedup of 1 million training data and 5 million training data and the linear. It can be seen from the figure that the speedup of two different training data is approximately linear with the number of nodes, and the speedup ratio of 5 million data sets is closer to the linear speedup. This shows that SBPNN has good parallelism in dealing with large scale training data.

4.3. SBPNN High-Efficiency Verification

SBPNN is implemented on the spark platform, and it has a faster iteration rate theoretically. In this paper, comparing with the parallel BP neural network algorithm MBNN based on the Hadoop platform and counting the training duration of each iteration under the same training data set to analyze the efficiency of SBPNN.

The experimental results are shown in Figure 10. The abscissa is the size of the experimental data, and the training data of 1 million, 2 million, 3 million, 4 million, and 5 million are used respectively, and the ordinate is the training duration of once iteration training of the current training data. The two histograms in the figure show the operation of SBPNN and MBNN respectively.

Seen from Figure 10, the SBPNN iterative training takes much less time than MBNN under different training data, and the training speed of SBPNN is faster than MBNN about 10 to 30 times. This shows that SBPNN has a higher iteration speed than MBNN.

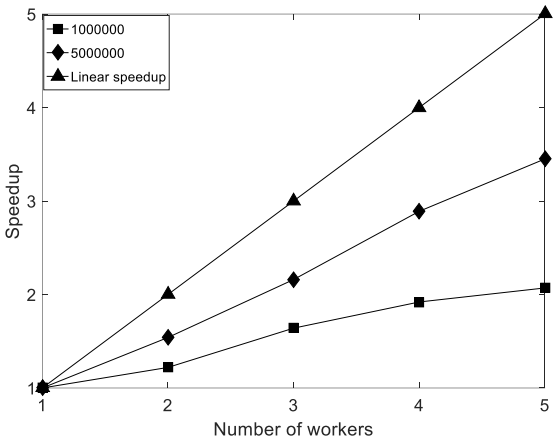


Fig. 9. Speed ratio diagram

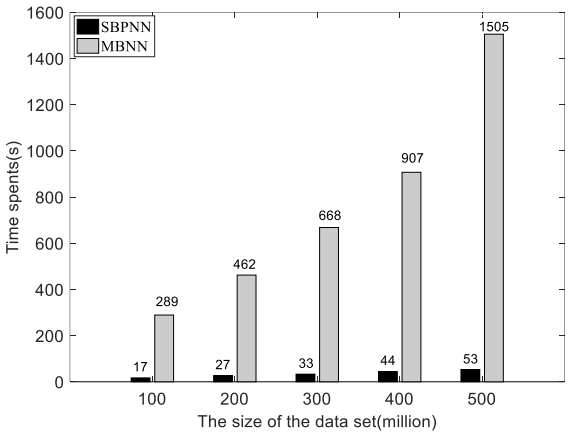


Fig.10. SBPNN Efficient Comparison Chart

5. Conclusion

In this paper, Spark parallel backpropagation neural network algorithm SPNNN is proposed, which solves the problem of time-consuming in training large scale data. Experimental results show that the proposed algorithm has good stability and convergence and it can effectively reduce the training duration by using the appropriate *l*. The proposed algorithm is proved in parallelism, and comparing with MBNN, SBPNN achieves faster iteration speed when dealing with large scale training data.

ACKNOWLEDGMENT

The paper is funded by the National Natural Science Foundation of China under Grant 61802282.

REFERENCES

[1] Cui-Chi R, Shu-Ying Y, Jun H. Handwritten character recognition based on BP neural network[J]. Journal of Tianjin University of Technology, 2006, 11(3):65-79.

[2] Shaowei Zhang, Pu Wang, Jin Yin. Neural Networks Approach Multipoint Orientation System Design[J]. International Journal of Applied Mathematics in Control Engineering,2019,2 (2):151-157.

[3] Jian Yao, Xueli Wu. Reconstruction of Arteriovenous Images based on CNN and Multi-electrode Electromagnetic Measurement[J]. International Journal of Applied Mathematics in Control Engineering,2019,2(2): 202-209.

[4] Wenxia Du, Xun Bai, et al. Fault Diagnosis of Sensors based on Empirical Mode Decomposition and Neural Network[J]. International Journal of

Applied Mathematics in Control Engineering,2019,2 (1):66-73.

[5] Turchenko V, Grandinetti L. Parallel batch pattern BP training algorithm of recurrent neural network[C]. International Conference on Intelligent Engineering Systems, IEEE Press, 2010.

[6] Y. Bo and W. Xun. Research on the performance of grid computing for distributed neural networks[J]. International Journal of Computer Science and Network Security,2006,6(4):179–187.

[7] Lin Y A. Map-Reduce for Machine Learning on Multicore[C]. Advances in Neural Information Processing Systems, MIT Press, 2006.

[8] Liu Z, Li H, Miao G. MapReduce-based Backpropagation Neural Network over large scale mobile data[C]. Sixth International Conference on Natural Computation, IEEE, 2010.

[9] Gu R, Shen F, Huang Y. A parallel computing platform for training large scale neural networks[C]. IEEE International Conference on Big Data, IEEE, 2013.

[10] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[C]. Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, USENIX Association, 2012.

[11] Liu Y, Xu L, Li M. The Parallelization of Back Propagation Neural Network in MapReduce and Spark[J]. International Journal of Parallel Programming, 2016, 45(4):181-195.

[12] Cui-Chi R, Shu-Ying Y, Jun H. Handwritten character recognition based on BP neural network[J]. Journal of Tianjin University of Technology, 2006,25(3):135-150.

[13] Yang J, Huang L. An Improvement and Application of Genetic BP Neural Network[C]. International Conference on Computational Intelligence and Security (CIS), IEEE, 2015.

[14] Turchenko V, Grandinetti L. Efficiency Analysis of Parallel Batch Pattern NN Training Algorithm on General-Purpose Supercomputer[J]. Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living,2009,11(2):226-232.

[15] Liu Wei, Liu Shang, Zhou Xuan. Sub batch learning method for BP neural networks[J]. Transactions on Intelligent Systems,2016, 11(2):226-232.

[16] Liu T, Fang Z, Zhao C, et al. Parallelization of a series of extreme learning machine algorithms based on spark[C]. International Conference on Computer and Information Science (ICIS), IEEE Computer Society, 2016.

[17] Jiao Runhai, Zhang Qian, Chen Chao. Improved algorithm for mining maximum frequent itemsets based on Spark[J]. Computer Engineering and Design,2017,38(07):1839-1843.

[18] Hui W, Yong W, Wen-Long K E. An Intrusion Detection Method Based on Spark and BP Neural Network[J]. Computer Knowledge and Technology, 2017,14(11):246-260.

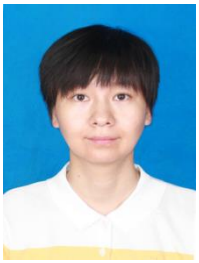
[19] Ganeshamoorthy K, Ranasinghe D N. On the Performance of Parallel Neural Network Implementations on Distributed Memory Architectures[J]. International Symposium on Cluster Computing and the Grid,2008,27(2):90-97.

[20] Haomin H U, Deyun M A. A Neural Network Prediction Model Based on Data Parallelism[J]. Computer Engineering, 2005,31(11):162-164.

[21] Jing W, Huo S, Miao Q, et al. A Model of Parallel Mosaicking for Massive Remote Sensing Images Based on Spark[J]. IEEE Access, 2017, 15(99):1-10.

[22] Shi J, Sullivan B, Mazzola M, et al. A Relaxation-based Network Decomposition Algorithm for Parallel Transient Stability Simulation with Improved Convergence[J]. IEEE Transactions on Parallel & Distributed Systems, 2018,22(4):56-70.

[23] Puljic K, Manger R. A distributed evolutionary algorithm with a superlinear speedup for solving the vehicle routing problem[J]. Computing & Informatics, 2012, 31(3):675-692.



Suqi Zhang received the Ph.D.degree from the School of Electronic Information Engineering, Tianjin University, Tianjin, China, in 2014. She is currently working at School of Information Engineering, Tianjin University of Commerce, Tianjin, China. Her research interests include intelligent recommendation, complex network analysis and data mining.

Shiyao She is currently studying at School of Artificial Intelligence, Hebei University of Technology. His research interests include intelligent recommendation, complex network analysis and data mining.

Junhua Gu born in 1966, Ph. D. Member of China Computer Federation. He is currently working at School of Artificial Intelligence, Hebei University of Technology, Tianjin, China. His main research interests include Data Mining, Intelligent Information Processing, Information Acquisition and Integration, Intelligent Computing and Optimization, Function and Information Display, and SoftWare Engineering and Project Management.