IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received October 7, 2019, accepted October 29, 2019, date of publication November 4, 2019, date of current version November 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2951189

Training Back Propagation Neural Networks in MapReduce on High-Dimensional Big Datasets With Global Evolution

WANGHU CHEN [1], JING LI [1], XINTIAN LI [1], LIZHI ZHANG [1], AND JIANWU WANG [2]
[1]College of Computer Science and Engineering, Northwest Normal University, Lanzhou 730070, China
[2]Department of Information Systems, University of Maryland, Baltimore County, Baltimore, MD 21250, USA

Corresponding author: Wanghu Chen (chenwh@nwnu.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61967013 and Grant 61462076.

ABSTRACT Owing to its scalability and high fault-tolerance even on a distributed environment built up with personal computers, MapReduce has been introduced to parallelise the training of Back Propagation Neural Networks (BPNNs) on high-dimensional big datasets. Based on the evolution of local BPNNs produced by distributed Map tasks with different data splits, the paper proposes a novel approach to the distributed data-parallel training of BPNNs in MapReduce. The approach provides a reasonable measure to get global convergent BPNN candidates from local BPNNs only convergent on the specific data splits. Further, it not only can reduce the iterations to get the global convergent BPNN, but also shows great advantages in avoiding the training to get trapped into a local optimum on high-dimensional big datasets. To improve the training efficiency further, local BPNNs from the same computing node are merged based on the average of their weight matrices before they act as individuals of the population for the global evolution. Our approach also leverages Random Project based sampling techniques to evaluate the fitness of each individual in order to lower the computation cost in the evolution stage. Experiments show that our proposed approach improves the training efficiency highly compared to the stand-alone or traditional MapReduce BPNN training, and improves model accuracy for larger datasets. The comparison with 23 other popular classification approaches also shows that our proposed approach has big advantages in accuracy.

INDEX TERMS Convergency, distributed data-parallelism, evolution, MapReduce, neural network.

I. INTRODUCTION

An Artificial Neural Network (ANN) is a computer model to essentially mimic the knowledge acquisition and organisational skills of the human brain, which consists of a number of interconnected processing elements called neurons [1]. The neurons of an ANN are usually arranged into two or more layers logically, and interact with each other via weighted connections. These scalar weights determine the nature and strength of the influence between the interconnected neurons. Each neuron can be connected to all the neurons in the next layer. There is an input layer where data are presented to the neural network, and an output layer that holds the response of the network to the input. It is the intermediate layers, also known as hidden layers, that enable these networks to represent and compute complicated associations between patterns.

The associate editor coordinating the review of this manuscript and approving it for publication was Huanqing Wang.

Neural networks essentially learn through the adaptation of their connection weights according to input data [2]. Back Propagation Neural Network (BPNN), one of the most popular ANNs, employs the back-propagation algorithm for its connection weight adaptation and can approximate any continuous nonlinear functions by arbitrary precision with enough number of neurons [3]. We call this process the *training* of a neural network and the input data containing potential patterns is called training samples.

In the past decades, ANNs have been widely used to model uncertain nonlinear functions [4], [5], and have shown great advantages in pattern recognition, classification and modelling of nonlinear relationships involving a multitude of variables [6].

A. A REAL WORLD APPLICATION AND CHALLENGES

The following is a real-world scenario. *To take targeted measures in poverty alleviation in a province of China,*

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

information about millions of poor families is gathered to analyse the possible reasons for their poverties and recommend the most proper measures to them. One of such measures is that some families can apply an interest-free loan every year. So, it is essential for the government to determine the families that enjoying such loans will be the most proper measure to help them to alleviate their poverties. At the same time, if a family is suitable to get the interest-free loan, the loan amount needs to be evaluated and the profit in next few years also needs to be predicted.

Considering the simplicity and wide applications of BPNNs, we tend to design a BPNN to learn patterns in the data. We are provided with 3-year's information about a million of poor families for the training of the BPNN. The data describing each poor family has 53 dimensions and covers information about family size, member composition, education levels, healthy status, labor powers, financial status, family assets, living environments and so on. We found there are two challenges to apply BPNN in the application. (1) The stand-alone training of such a BPNN on the big and high-dimensional dataset is very time-consuming. (2) It is full of challenges to avoid the training process to get trapped into a local optimum on a big and high-dimensional dataset. Though we always search for a globally optimal solution, while a gradient descent algorithm can only find a local optimum in a neighbourhood of the initial solution [3].

B. EXISTING SOLUTIONS AND REMAINING CHALLENGES

As the discussion above, on a high-dimensional big dataset, it is essential to parallelise the training of BPNNs. In the past decades, MPI has been applied to the parallel training of BPNNs usually on supercomputers with distributed memory sharing system [7]. On the other way, GPU is also utilised to implement ANN training algorithms usually on a cluster of GPGPUs [8]. Besides the special programming necessary, such devices are also not so accessible to some users. More importantly, the fault tolerance of an MPI based approach will be affected largely in an unreliable environment, because processes on different nodes need to communicate each other, and all intermediate data are stored in the memory.

As a kind of data-parallel programming paradigms for distributed applications, MapReduce has become one of the de facto programming standards in big data applications. It supports distributing a big dataset on a Distributed File System (DFS) cross multiple computing nodes of a cluster. Then, lots of Map tasks are configured and scheduled on each computing node. The $< key, value >$ pairs generated by each Map task will be serialised into files and some Reduce tasks will pull them to get the final result. Because all Map and Reduce tasks do not need to communicate each other and all intermediate results are serialised into files on DFS, when such a task fails unexpectedly, it will be re-scheduled to another node transparently, but the whole job does not need to restart. So, it can get high performance and fault tolerance even on a cluster composed of personal computers compared to other traditional program paradigms.

Therefore, in recent years, MapReduce has been introduced to parallelise the training of BPNNs. For example, the approaches proposed in [9] and [10] utilise parallel Map tasks for forward propagation, and parallel Reduce tasks for both error back propagation and connection weight adaptation. Compared to the stand-alone training, this can really improve the efficiency. However, if all Map tasks share a same global BPNN, the necessary synchronisation will still be the bottleneck of the training efficiency. If each Map task has its own local BPNN, it is still a challenge on how to generate convergent global BPNN on the whole training dataset. An interesting approach is proposed in [11], which implements the weight adaptation of the global convergent BPNN in a Reduce task based on the average of the weights of local BPNNs produced by all Map tasks. Nevertheless, there is little evidence to show it can speed up the convergence process as the training iteration increase. Moreover, these approaches did not discuss how to avoid the training process to get trapped into a local optimum. The approach proposed in [12] produces lots of local BPNNs on each node by Map tasks and has to do predictions based on the voting of these local models in an application. This will make the model more complex and affect the prediction efficiency. In [13], Genetic algorithm is used only to find appropriate initial weights of BPNN.

As the analyses above, in MapReduce training of a BPNN, because each local BPNN whose connection weighs are adjusted by a specific Map task is only convergent on a specific splits of the training dataset, the generation of a global BPNN candidate that may be convergent on the whole training dataset is still essential to be explored. At the same time, how to avoid the training process to get trapped into a local optimum is also an important problem to be addressed.

According to the discussion above, it is interesting and meaningful to explore the BPNN parallelisation based on MapReduce. The reasons include: (1) It is an interesting topic to verify whether MapReduce is feasible and efficient to support BPNN parallelism in big data applications. (2) MapReduce can get high fault tolerance even on an unreliable cluster. For big data applications, it is very important that the whole job does not need to restart or trap into waiting when a process on a single node halts abnormally. (3) MapReduce jobs can run on a cluster composed of cheap personal computers. So it is more convenient to some users. (4) Moreover, current BPNN training approaches based on MapReduce have not given an effective and efficient measure for global convergent BPNN generation, and focus little on avoiding the training process to get trapped into a local optimum. These problems are still essential to be addressed.

C. OUR APPROACH AND CONTRIBUTIONS

As a subset of evolutionary computation, Evolution Algorithm is a famous generic population-based meta-heuristic optimisation algorithm [14]. It uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination and selection. Candidate solutions to the

optimisation problem play the role of individuals in a population, and the fitness function determines the quality of the solutions. The evolution of the population takes place after the repeated application of the above operators. Therefore, taking each local BPNN convergent on the splits of a Map task as an individual, the evolution of local BPNNs is an ideal way to generate the global BPNN candidate. Moreover, because EA has a great advantage in global optimum search and is much less sensitive to initial conditions, though it is rather inefficient in fine-tuned local search, so along with the gradient descent algorithms of BPNN [14], it can improve the efficiency to find the global convergent BPNN candidate and has the strong abilities to avoid falling into local optimum.

Therefore, in this paper, based on the evolution of local BPNNs, we intend to find an ideal solution to train BPNNs in MapReduce on high-dimensional big datasets to improve the training efficiency and accuracy.

The contributions of this paper are as follows. (1) A 3-stage mechanism for MapReduce training of a BPNN on a high-dimensional big dataset is proposed. Merging local BPNNs on the same node and the evolution of the merged BPNNs from all cluster nodes make the mechanism very effective. The concept may be used to facilitate the training of other ANNs on other data-parallel computing platforms. (2) Based on the evolution of local BPNNs produced by all Map tasks, a reasonable measure to generate the global convergent BPNN candidate is proposed. It can greatly reduce the number of iterations to find the global convergent BPNN candidate and has high abilities to avoid the training to get trapped into a local optimum. (3) An algorithm for MapReduce training of BPNNs based on the evolution of local BPNNs is designed. Random Project is introduced to improve the training efficiency further. Experiments show that the algorithm can improve the training efficiency and accuracy remarkably on a high-dimensional big dataset.

The next sections of the paper are arranged as follows. In section II, related work about the parallelism of ANN training is discussed. Then, BPNN and its traditional training are analysed in section III. In section IV, focusing on the challenges in MapReduce training of BPNN, the proposed approach based on the evolution of local BPNNs is discussed in details. The experimental verification and related analyses are shown in section V, and conclusions are given in section VI.

II. RELATED WORK

A. MPI BASED PARALLEL LEARNING OF ANNs

Message Passing Interface (MPI) is widely used for the communication among processes that model a parallel program running on a distributed memory system. Experiments have shown that in a distributed memory sharing environment, BPNNs implemented with MPI have good efficiencies [7].

Using different parallel programming standards or message passing mechanisms, such as MPI, OpenMP and MPICH-G2, many studies have explored the training of ANNs in different computing environments [15]–[18]. For example, a parallel batch-pattern back propagation training algorithm for Recirculation Neural Network on many-core high performance computing systems is proposed in [15]. An approach to training ANNs in a grid distributed memory environment and that to accelerating neuromorphic models on a cluster of GPGPUs are proposed in [16] and [17] respectively.

A parallelisation scheme for the computation of the fitness function is proposed in [19], which leverages the advantages of differential evolution to improve the training of feed-forward neural networks. The approach proposed in [20] incorporates distributed parallel computing technique to modelling neural networks. MPI is utilised for both data generation and neural network training in parallel. In [21], load balancing on heterogeneous LAM/MPI clusters is explored based on average evaluation time and communication delay feedback estimates from slaves. It shows that considerable speed-ups can be achieved using the proposed fuzzy controller.

According to the analyses above, MPI based approaches can improve the learning efficiency of ANNs remarkably, and show advantages in load balancing and speed-up as well. However, because processes on different nodes need to communicate each other during the running of an MPI program, if one process halted abnormally, the whole program would trap into waiting. Moreover, because all intermediate data keep stay in the memory, if any fault happens on a node, an MPI computation needs to restart from the beginning. To a big data application, such cost may be very high sometimes. At the same time, MPI programs regularly run on supercomputers with distributed memory. Compared with MPI, MapReduce has high fault tolerance even on an elementary cluster composed of personal computers. So, it is still meaningful to explore the effective approaches to training ANNs in MapReduce.

B. GPU BASED ANN LEARNING

Graphics Processing Unit (GPU) has been widely used in the implementation of algorithms that are not related to computer graphics. Especially, lots of studies utilised GPU to improve the learning of ANNs in the past decades.

Most of these proposed approaches use GPU to implement the multiplication between the weights and the input vectors in each layer, which is well known as the inner-product, but to do the training phase off-line based on the CPU implementation of the training algorithm [8]. For example, a GPU-based implementation of matrix algebra operations is employed in [22], and is used to implement a GPU simulator of a feedforward complex-valued neural network. It shows big advantages in speeding up the learning of ANNs on big datasets. Many studies concentrate on the performance improving of convolutional neural networks using GPU [23], especially on exploring low-overhead and efficient hardware mechanisms that can skip multiplications always giving zero results regardless of input data [24]. In [25], the implementation of a multichannel structure of Hopfield Network on

GPU platforms is explored, and a kind of parallelism is also proposed to explore the peak computing capacity of a GPU device. Current studies also show that GPU performs well on scaling up neural networks [26], and running DNN applications on mobile device [27].

According to the discussion above, GPU shows its big advantages in computation performance on inner-product and other matrix algebra operations during the ANN learning, as well as in scaling up some ANNs owing to its hardware mechanism. Nevertheless, considering that the special GPU programming on neural network learning algorithms may be needed, and GPU devices sometimes are not so convenient for some users, our studies in this paper are still interesting and meaningful.

C. MAPREDUCE BASED ANN LEARNING

Just as it is mentioned in the introduction, MapReduce has been introduced to parallelise the training of BPNNs [9]–[12]. Combining with various domain applications, different approaches to BPNN learning based on MapReduce are explored in [28]–[30]. These studies show that MapReduce based approaches can improve the efficiency of the BPNN training greatly, and the accuracy of a BPNN model learning from a big training dataset is much higher than that from a small one. There are also many studies to explore the training of various ANNs based on MapReduce, such as those represented in [31] and [32].

From the perspective of model learning process, these approaches fall into three categories as follows. One kind of them can be considered to share a global BPNN by all Map tasks [9], [10]. So, to adjust the connection weights of the global BPNN, Map/Reduce tasks may need to do synchronisations. Obviously, the training efficiency and the fault tolerance will be affected. Another type of these studies intends to get the global BPNN candidate based on the average of those produced by all Map tasks [11]. Though the idea is very interesting, there is little evidence to show that it can speed up the convergence process as the training iteration increase. The other type of novel approaches intends to provide an ensemble BPNN classifier after the distributed training, which usually leverages the bootstrapping of samples, and does future classification based on majority voting [12]. Though it can improve the training efficiency, it increases the cost in the prediction stage and makes final model more complex. More importantly, the approaches mentioned above give little concern on how to avoid the training process to get trapped into a local optimum.

Recently, other approaches such as those based on Spark and other data-parallel platforms are explored [33]–[35], but the challenges faced by the approaches based on MapReduce still exist. In [36]–[38], NN based decentralised control systems are represented. The ideas of these approaches are helpful to explore the ensemble classifiers based on the data-parallel training of BPNNs as shown in [12].

In this paper, we study how to better parallel BPNN training in MapReduce on high-dimensional big datasets. How to get the global BPNN effectively and how to avoid the training to get trapped into local optimum are the main challenges to be addressed. This paper extends our work in [39] through the following aspects. (1) We propose a 3-stage BPNN training approach in MapReduce, which merges the local BPNNs on the same node based on their weight average before the evolution of these merged local models. It can greatly improve the training efficiency and avoid the training to get trapped into a local optimum. At the same time, compared to our former work, local BPNN training and global evolution are finished within one MapReduce job. (2) In global evolution stage, we introduce Random Project to get samples for fitness computation from the original datasets in this paper. It can further reduce the training time without affecting the model's accuracy. (3) The extended policy for the evolution of local BPNNs is also discussed in detail, and the algorithm supporting the proposed approach is given. (4) In experiments, we compare our proposed approach with 23 other popular classification approaches to verify its effectivity. Moreover, a BPNN with 2 hidden layers is used in this paper to discuss whether our proposed approach is still feasible and efficient. (5) We further evaluate the speed-up ratio of our proposed approach on a Hadoop cluster in this paper.

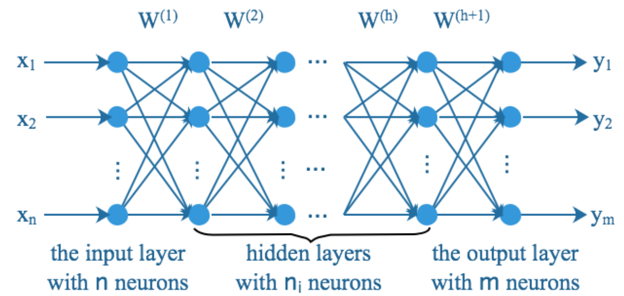


FIGURE 1. An example of a BP neural network.

III. BPNN AND ITS TRADITIONAL TRAINING

A. BPNN AND BP ALGORITHM

Fig. 1 illustrates a BPNN model with 1 input layer, h hidden layers and 1 output layer, which receives a vector $(x_1, x_2, \ldots, x_n)$ as input and outputs $(y_1, y_2, \ldots, y_m)$ after a series of computation. As shown in Fig. 1, the input layer is composed of n neurons, the output layer has m neurons, and the ith hidden layer has n_i neurons, where $1 \leq i \leq h$. For a BPNN, the neurons of a layer will connect to all those of the next layer. So, the input layer is the preceding of the first hidden layer, and the output layer is next to the last hidden layer. The connection weights between neurons of two neighbouring layers can be denoted as a matrix $W^{(i)}$, where $1 \leq i \leq h+1$. Obviously, $W^{(1)}$ is the connection weight matrix between the input layer and the first hidden layer, and $W^{(h+1)}$ is that between the last hidden layer and the output layer. Because the structure of the BPNN can be specified by a series of such weight matrices, to be more concise, a BPNN can be denoted as $BP = (W^{(1)}, W^{(2)}, \ldots, W^{(h+1)})$.

The learning process of BP algorithm contains two steps. (1) *Forward propagation*: the output of a specific layer

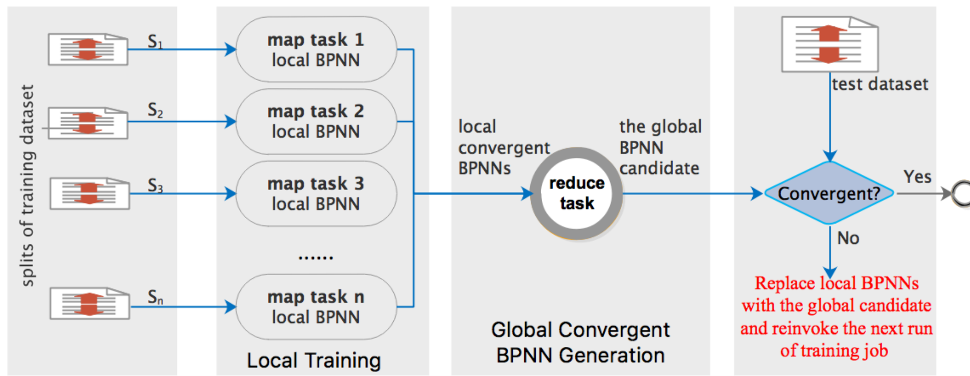


FIGURE 2. Traditional MapReduce training of BPNNs [39].

transfers to the next layer as its input. For example, for a three layer BPNN, the output of the jth neuron of the hidden layer is $o_j = f(\sum w_{ji}x_i + \theta_j)$, where w_{ji} is the element of the matrix $W^{(1)}$, x_i is an element of the input vector, and θ_j is the threshold of the jth neuron of the hidden layer. Importantly, f is an activation function, which usually uses Sigmoid function $f(x) = 1/(1 + e^{-x})$. Similarly, for the output layer, there is $o_k = f(\sum w_{kj}o_j + \theta_k)$, where w_{kj} is the element of the matrix $W^{(2)}$, o_j is the output of the jth neutron of the hidden layer, and θ_k is the threshold of the kth output neuron. (2)*Reverse propagation of error*: the error signal will be returned along the original neural network. During the returning, each weight value will be modified, so as to minimise the sum of squares error $E = \frac{1}{2}\sum_p \sum_k (y_{pk} - o_{pk})^2$ between the actual output and supposed output of the network, where p is the index of a sample, k is the index of an output neuron, y_{pk} is the supposed output of the kth neuron with the pth sample and o_{pk} is the actual output correspondingly. When E satisfies the expected threshold, we call the BPNN is convergent on the training dataset. Then, a test dataset will be used to verify the accuracy of the BPNN.

B. TRADITIONAL MAPREDUCE TRAINING

Given a small size BPNN, when the training dataset is also in a small scale, it usually can learn with a hight efficiency stand-alone (on one single machine). However, for a large scale BPNN with a large training dataset, it often takes a very long time to finish learning stand-alone. Just as our discussion above, MapReduce training of BPNNs will be a good choice in such a condition.

The traditional training process of BPNNs in MapReduce is shown in Fig. 2. The training dataset is divided into a series of splits, and then Map tasks are scheduled to each machine node of a cluster where some splits are distributed. Each Map task will get an initial BPNN whose connection weights are generated randomly in the first run of training, and finish the training of its BPNN with a specific split on the node. When all the Map tasks finish the training of their own BPNN, BPNNs of all these Map tasks are gathered by a Reduce task. Obviously, in Fig. 2, the ith local BPNN may be only convergent on the ith split of the training dataset.

Therefore, how to generate a global BPNN that is convergent on the whole training dataset from all these local convergent BPNN reasonably is a big challenge. To the best of our knowledge, there is still no such an effective approach. Current approaches usually realise this with the average of local BPNNs' connection weights. However, there is little solid arguments to show their advantages in speeding up the global convergency of the BPNN candidate. So, their effectivity and efficiency still need to be verified, and the problem is still essential to be addressed.

IV. DATA-PARALLEL TRAINING WITH THE EVOLUTION OF LOCAL BPNNs

Considering the good performance of the evolution algorithm in global optimum search, in this paper, we introduce the evolution of local BPNNs into the data-parallel training of a BPNN to conquer the challenges mentioned above, and propose a novel approach called MREvolution.

A. OVERVIEW OF THE PROCESS

As shown in Fig. 3, the proposed approach can be divided into three stages logically. All input and intermediate data generated during the BPNN training are all stored in a Distributed File System like Hadoop HDFS, besides weight matrices of local BPNNs.

Stage 1: Local Training Stage. In this stage, the training dataset is divided into lots of splits and distributed on nodes of a cluster with a specific policy. Then, the MapReduce engine will schedule Map tasks onto these nodes of the cluster. Each Map task reads a split on the local node and an initial global BPNN candidate from a specific file in the DFS, and takes each sample in the split to train the BPNN with Gradient Descent Algorithm(see in section III-A). When a local training process is finished by a Map task, a local convergent BPNN will be serialised into the DFS. The initial global BPNN candidate read by each Map is generated randomly in the first run of training, and will be replaced by the current global BPNN candidate in the next turn of training (see in the Test stage in Fig. 3).

Because the BPNN can be specified by all its connection weights, it can be notated as weight matrices between all

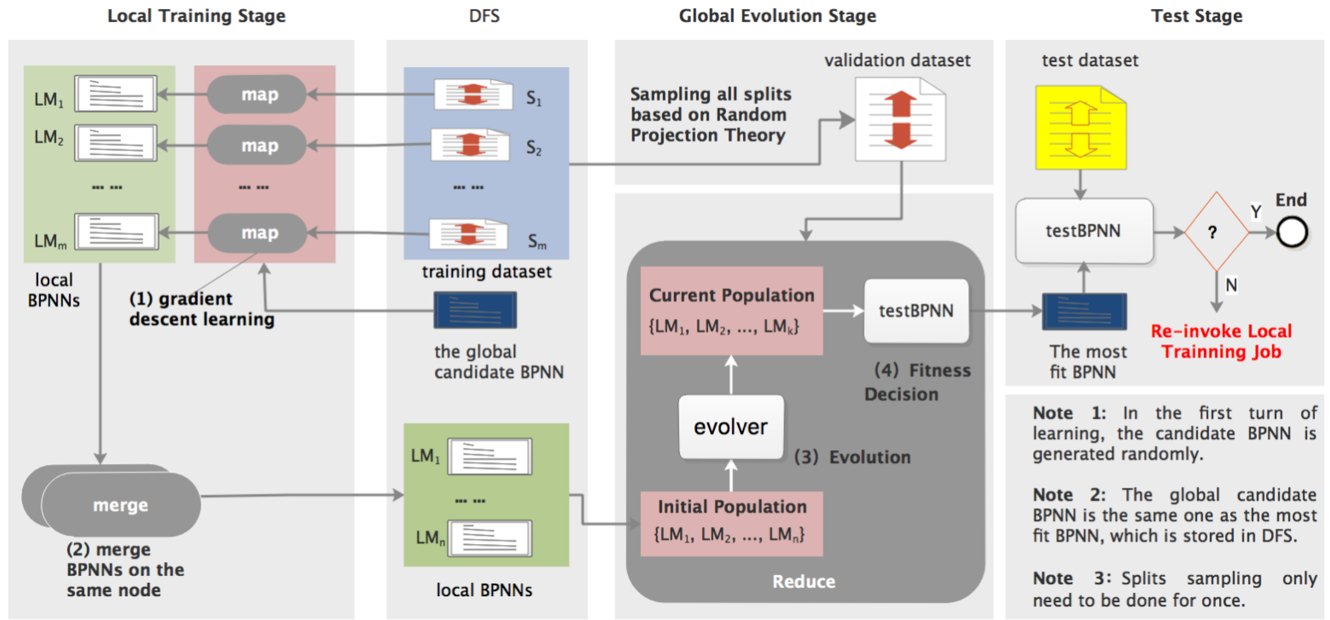


FIGURE 3. BPNN training in MapReduce driven by evolution.

neighbouring layers. To be convenient to discuss, the BPNN for each Map task is notated as LM (Fig. 3). So, BPNN and its weight Matrices have the same meaning in this paper.

Therefore, if the training dataset is divided into m splits $\{S_1, S_2, \ldots, S_m\}$, m local BPNNs $\{LM_1, LM_2, \ldots, LM_m\}$ will be produced by Map tasks when the training process finished. Each LM_i produced by the ith Map task is only convergent on the specific split S_i. Before a Reduce task pulls local convergent BPNNs to generate the global convergent BPNN candidate, local BPNNs on the same node will be merged with the average of their connection weights in the Merge stage of the MapReduce job. This can reduce the I/O cost and improve the efficiency highly. Eventually, n pairs $\{< K_i, LM_i > |1 \leq i \leq n\}$ will be written into files in the DFS, where K_i corresponding to the *ID* of the current node is the key of the local BPNN LM_i, and n is the number of nodes of the current cluster.

Because all Map tasks run in data-parallelism on the nodes of a cluster, this process is highly efficient. At the same time, because all Map tasks need not to communicate each other, when a Map task halted abnormally, it will be scheduled to another node transparently. So, the process is very highly fault-tolerant.

Stage 2: Global Evolution Stage. In this stage, a Reduce task pulls all local BPNNs produced by Merge tasks, $\{< K_i, LM_i > |1 \leq i \leq n\}$, from the DFS as the initial population of an Evolver. Each LM_i is called an individual of the population. The Evolver then uses its operator *Selection*, *Mutation* and *Crossover* to produce new individuals whose fitness satisfying the threshold will be put into the population of the next generation. An individual is considered to have higher fitness if its error computed by *testBPNN* function based on a validation dataset is smaller.

Because the objective in this stage is to generate a global BPNN convergent on the whole training dataset, the validation dataset that is also the input of *testBPNN* is from all splits of the training dataset. However, if the whole training dataset is taken, the computing efficiency will be affected. Therefore, we introduce Random Project (see in section IV-B) to sample the training dataset beforehand for the fitness computing in this stage. The big dataset with high dimension is projected to a new one with much lower dimension that still remain the relations among the samples based on the approach. Then, referring to the lower dimension space, a validation dataset will be extracted from the whole training dataset based on the probability distribution of the samples. The validation dataset is then used for the selection of the global BPNN candidates based on the fitness computing of individuals. Because the sampling needs to be done only once beforehand, its cost can be ignored. The fitness computing approach is also acceptable because the validation samples can reflect the features of the original dataset.

When the new generation of individuals are generated, a set of pairs, $\{< K_i, < GM_i, f_i \gg |1 \leq i \leq k\}$, will be put into a file in the DFS, where K_i is a key, GM_i is a global convergent BPNN candidate whose fitness is f_i, and k is the size of the population.

Stage 3: Test Stage. In this sage, all test samples will be loaded from DFS, and the error e_i between the supposed output and the actual output of *testBPNN* according to each sample is computed. The $GM \in \{< K_i, < GM_i, f_i \gg |1 \leq i \leq k\}$ that has the smallest e_i and $e_i \leq \delta$ will be selected as the eventual global BPNN, where δ is a threshold. Specially, if no element in $\{< K_i, < GM_i, f_i \gg |1 \leq i \leq k\}$ satisfies the condition $e_i \leq \delta$, the one with the smallest fitness f_i will be selected. Then, the one selected is saved as the global BPNN

candidate into the file in the DFS to replace the old one, and the MapReduce training job is re-invoked. Thus, all Map tasks will get a better initial BPNN in the new turn of training.

After some turns of such distributed data-parallel training, the final BPNN will become globally convergent on the whole training dataset and satisfied the expected accuracy. Compared with our former work [39], the mechanism proposed finishes the local training and the global evolution in one MapReduce job, and can reduce the cost for MapReduce job configuration, initialisation, I/O operations and so on.

B. DATASET SAMPLING FOR FITNESS EVALUATION

Just as shown in Fig. 3, the fitness of each new individual needs to be computed to support the evolution of local BPNNs. To do so, the test dataset should be the whole training data. When the training dataset is very large, the fitness computing really can be evaluated with data-parallel computing to maintain an acceptable efficiency. However, if a smaller test dataset that can reflect the features of the whole training data well can be found, it can highly improve the training efficiency of BPNNs. The similar concept is used in some sampled-data neural-network-based systems [40]. Therefore, sampling the splits of the training dataset for all Map tasks in the local training stage to provide validation dataset for the global evolution stage is meaningful. The challenge is how to guarantee the quality of the sampled dataset.

According to the Random Projection theory [41], given a dataset $S = \{x_i\}$, where each $x_i \in S$ is an $n-dimension$ sample, if $S \in R^{n\times k}(k < n)$ is a random matrix, for each x_i and $x_j (i \neq j)$, let $\bar{x}_i = S^T x_i$ and $\bar{x}_j = S^T x_j$, $\bar{x}_i, \bar{x}_j \in \bar{S}$, the distance between x_i and x_j in a high dimension can be simulated by that between $\bar{x}_i$ and $\bar{x}_j$ in a low dimension. So, the lower dimension dataset can reflect the relations among data in the original high dimension to some extent.

Therefore, we can sampling the training data taking the advantages of this theory. (1) Get a lower dimension dataset $\bar{S} = S \times R^{n\times k}(k < n)$ from the primary $n-dimension$ dataset S. (2) Then, Centralisation L2 Discrepancy [42] is used to mark uniform samples from $\bar{S}$. (3)The corresponding samples in S according to the marked ones in $\bar{S}$ will be selected into the eventual test dataset.

C. THE EVOLUTION OF LOCAL BPNNs

As the discussion above, it is an effective solution to generate the global convergent BPNN based on the evolution of local convergent BPNNs using EAs. The fitness (or error) function does not have to be differentiable or even continuous since EAs do not depend on gradient information [14].

Given an individual X, which is a BPNN, its fitness is defined as $f(X) = 1/E(X)$, in which $E(X) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}(o_{ij} - \bar{o}_{ij})^2}$, where n is the number of test samples, m is the number of neurons in the output layer, o_{ij} is the output of the jth output neuron with the ith test sample, and $\bar{o}_{ij}$ is the expected output value corresponding to o_{ij}.

During the evolution, an individual X is considered to be composed of a series of chromosomes, each of which is either a connection weight or a threshold. So, each individual X is encoded into a series of real numbers. For example, given a three-layer BPNN, supposing its input layer, hidden layer and output layer has n_0, n_1 and n_2 neurons respectively, each connection weight $w_{jk}^{(i)}$ will be treated as a chromosome, where $i = [1, 2]$, $1 \leq j \leq n_i$ and $1 \leq k \leq n_{i-1}$. As shown in Fig. 1, $w_{jk}^{(1)}$ is the connection weight of the jth neuron in the hidden layer to the kth neuron in the input layer. Similarly, $w_{jk}^{(2)}$ is the element of the weight matrix $W^{(2)}$ between the hidden layer and output layer. Importantly, a threshold $\theta_j^{(i)}$ ($i = [1, 2]$, $1 \leq j \leq n_i$) is also be treated as a chromosome, where $\theta_j^{(1)}$ is the threshold for the jth neuron in the hidden layer, and $\theta_j^{(2)}$ is that in the output layer.

Then, an individual will be encoded into a concatenation of all connection weights and threshold like $(W^{(1)}, W^{(2)}, \Theta^{(1)}, \Theta^{(2)})$, where $\Theta^{(1)}$ and $\Theta^{(2)}$ are a vector of threshold values for the hidden layer and the output layer respectively. We note that each connection weights will be flattened. Because hidden nodes in BPNN are in essence feature extractors and detectors, separating inputs to the same hidden node far apart during the encoding would increase the difficulty of constructing useful feature detectors because they might be destroyed by crossover operators [14]. Therefore, connection weights to the same hidden/output node are put together. That is to say, for each $i = [1, 2]$, $W^{(i)}$ will be flattened as $W^{(i)} = (w_1, w_2, w_3, \ldots, w_t, \ldots)$, $1 \leq t \leq n_{i-1} \times n_i$, and the element $w_{jk}^{(i)}$ in the original weight matrix $W^{(i)}$ refers to the element w_t in the flattened vector, of which $t = (j-1) \times n_{i-1} + k$, where $1 \leq j \leq n_i$, $1 \leq k \leq n_{i-1}$.

The selection operator can be defined simply by the fitness of an individual. Given an individual X, if $f(X) > \beta$, it may be selected, where β is a threshold. To control the number of the individuals entering the next generation, we define p_s as a selection rate. The individual satisfying$f(X) > \beta$ is selected by a probability p_s.

Let p_c is a crossover rate, two individuals will enter the next generation after crossover as a probability p_c. Given two individuals $X_i^{(t)}$ and $X_j^{(t)}$ in current generation t, supposing $f(X_i^{(t)}) > f(X_j^{(t)})$, if their chromosomes at the corresponding location are denoted as $x_i^{(t)}$ and $x_j^{(t)}$, the new chromosome $x_i^{(t+1)}$ and $x_j^{(t+1)}$ after the crossover of $x_i^{(t)}$ and $x_j^{(t)}$ to enter the next generation $t + 1$ can be generated as following (Eq. (1), (2)), where α is a value in [0, 1].

$$x_i^{(t+1)} = x_i^{(t)} + \alpha(x_i^{(t)} - x_j^{(t)}) \tag{1}$$

$$y_i^{(t+1)} = x_i^{(t)} - \alpha(x_i^{(t)} - x_j^{(t)}) \tag{2}$$

Let p_m is a mutation rate, then an individual will mutated with a probability p_m. Given an individual X, its new chromosome $x_i^{(t+1)}$ of $x_i^{(t)}$ after mutation is determined according to Eq.(3), in which $r = (x_i^{(t)} - x^l)/(x^u - x^l)$, x^u and x^l are the upper and lower bounds of the real coded value of the chromosome, r_0 is a uniform random number between 0 and 1, s is a number created randomly following the power

Algorithm 1 Local-Training

function map(key_i, S_i)

//S_i is a split of the training dataset identified by key_i
LM_i = loadLocalBPNNFromDFS($bpFile$)

//LM_i is a local weight matrix for the current Map task
initializeBPNN ($BPNN_i$, LM_i)
$samples = S_i$.format ()

for each $s \in samples$ **do**
e_i = gradientDescentLearning($BPNN_i$, s)
//e_i is training error //see in section III-A
end for

key_i = getHostID()
output(key_i, $< LM_i, e_i >$)
end function

function merge(key_i, $\{< LM_j, e_j >\}$)
//Elements in $\{< LM_j, e_j >\}$ have the same key, key_i
n = 0;
LM = O; //zero matrix
e = 0;
for each $< LM_j, e_j > \in \{< LM_j, e_j >\}$ **do**
$LM = LM + LM_j$
$e = e + e_j$
n = n + 1
end for

$LM = LM/n$
$e = e/n$
output(key_i, $< LM, e >$)
end function

Algorithm 2 Global-Evolution

function reduce($\{key_i, < LM_i, e_i >\}$)
//$\{key_i, < LM_i, e_i >\}$ is produced by MERGE() and is pulled here by Reduce task of the MapReduce job
$P_0 = \{LM_i\}$ //initialize the population

P_1 = select(P_0, β, p_s)
//copy individuals whose fitness no smaller than β with a probability p_s into the next generation (section IV-C)

$P_1 = P_1 \cup$ crossover(P_0, p_c) //Eq.1 and 2
//generate new individuals by crossover, p_c is the crossover rate

$P_1 = P_1 \cup$ mutate(P_0, p_m) //Eq.3
//generate new individuals by mutation, p_m is the mutation rate

$validationSamples$ = sampling (S)
//see in section IV-B, S is the whole training dataset

$f = 0$
for each $LM_i \in P_1$ **do**
$BPNN_i$ = InitializeBPNN (LM_i)
f_i = testBPNN($BPNN_i$, $validationSamples$)
//see in section III-A
if $f < f_i$ **then**
$LM = LM_i$
end if
end for

output(LM)
//output the BPNN with the highest fitness
end function

distribution [43].

$$x^{(t+1)} = \begin{cases} x_i^{(t)} - s(x_i^{(t)} - x^l), & r < r_0 \\ x_i^{(t)} + s(x^u - x_i^{(t)}), & r \geq r_0 \end{cases} \tag{3}$$

Usually, p_m and p_c are set as static values according to experiences in applications. Some approaches to generate p_m and p_c self-adaptively are also proposed. However, considering its weight adjustment ability of a BPNN, in this paper, p_m and p_c are set according to the experiences.

D. THE LEARNING ALGORITHMS

Based on the discussion above, the algorithms for distributed data-parallel training of a BPNN with the evolution of local BPNNs can be described as follows.

In algorithm 1, MAP() and MERGE() refer to the two stages in the execution process of a MapReduce job. The key_i in MAP(key_i, S_i) is the offset of the first character in the split S_i of the source data. It also can be the fingerprint of the data split, i.e., $key_i = fingerprint(S_i)$. Differently, key_i in MERGE(key_i, $\{< LM_j, e_j >\}$) is the host ID of the current machine node.

Algorithm 2 generates the candidate global convergent BPNN based on the evolution of local BPNNs. The input parameter of REDUCE() pulls from the output of MERGE(). Because the evolution tends to generate the global BPNN candidates convergent on the whole training dataset, S in the algorithm 2 is the whole training dataset.

Algorithm 3 reads the test dataset to verify whether the BPNN has satisfied the accuracy. If it has not yet, the whole training process will be re-invoked. In the next turn of training, all Map tasks will load the global BPNN candidate generated by the Reduce task in the last turn as its initial BPNN.

The whole training process of the proposed approach in this paper, MREvolution, will be a loop of Local-Training(Alg. 1), Global-Evolution(Alg. 2) and Test(Alg. 3).

Algorithm 3 Test

function test(LM)
$testSamples$ = loadTestSamples()

n = 0
m = length(testSamples)
for each $s \in testSamples$ **do**
BPNN = InitialBPNN (LM)
$e_i = testBPNN(BPNN, testSamples)$
if $e_i \leq \delta$ **then**
$n = n + 1$
end if
end for

if $n/m > r$ **then**
GM = LM
//GM is the final global convergent BPNN
output(GM)
return //The whole training process finished!
else
saveIntoDFS(LM, $bpFile$)
//LM will be read by Map tasks in next turn of training
invoke (Local-Training)
//begin the new turn of training
end if
end function

V. EXPERIMENTS AND EVALUATION

To evaluate the approach proposed, three BPNN training tools, MREvolution, MRAverage and StandAlone are implemented. MREvolution uses our BPNN training approach proposed in this paper, and MRAverage is based on the BPNN training approach in MapReduce with local weight matrices average. StandAlone does BPNN training with the traditional non-parallel algorithm.

The training datasets in experiments come from the scenario mentioned in section I. Both MREvolution and MRAverage do their BPNN training on the same Hadoop cluster with 1 master node and 9 data nodes. We note that only the data nodes of the cluster undertake training tasks. To make it fair, one of the cluster nodes is selected for StandAlone. Each cluster node has 32GB RAM, 2T disk storage and two 8-core CPUs of 2.4GHz.

A. EFFICIENCY AND ACCURACY ANALYSES

Firstly, a 3-level BPNN model is designed, which has 53 input neurons, 20 hidden neurons and 1 output neurons. To compare the training efficiency and accuracy of these three approaches, 3 different datasets are used. The first training dataset contains 0.95 million samples, and the other two have 1.9 million and 3.8 million samples respectively.

Fig. 4 shows the training time of each tool on the specific training dataset. In the training process, we set the error threshold as 0.02. We find that the tool StandAlone has not finished the BPNN model training yet after about 7 hours when we stopped the training process manually. Even so, the training time of MREvolution is only 7% to 10% of that of StandAlone. That is to say the data-parallel approaches make it feasible for BPNN to work on a big and high-dimensional training dataset for its high training efficiency.

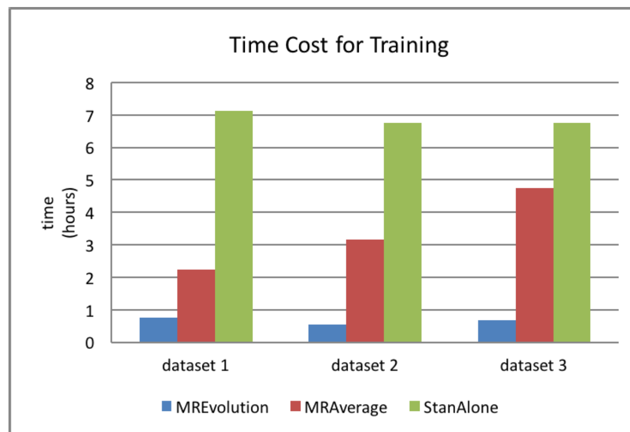


FIGURE 4. Training time taken by the three approaches to finish the training.

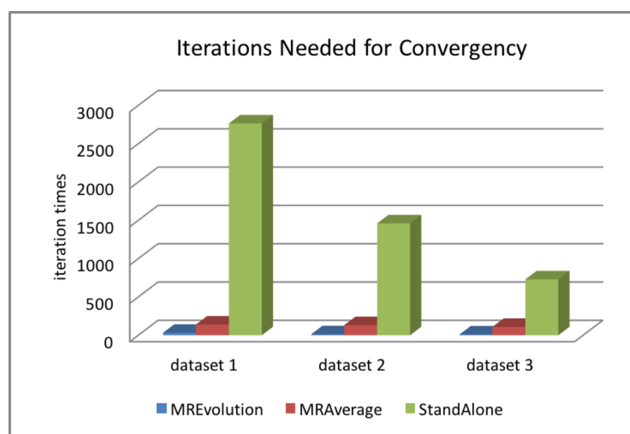


FIGURE 5. Iteration times needed by the three approaches to finish the training.

It also shows that MREvolution can reduce the training time by 66.8%, 82.5% and 85.9% compared to MRAverage on the three datasets respectively (Fig. 4). Fig. 5 interprets why MREvolution can improve the training efficiency so largely. It shows that MREvolution only needs to iterate 10 to 32 times to make the BPNN convergent, but MRAverage and StandAlone need to iterate hundreds or thousands of such turns. The reason is that MREvolution introduces the evolution into the generation of the global convergent BPNN. So, the approach proposed in this paper provides a feasible and effective measure for the global BPNN candidate generation in MapReduce training of BPNNs.

Fig. 6 shows the comparison of the accuracy of the BPNN when finished its training based on three different approaches respectively. It shows that on the smallest dataset, the accuracy of MREvolution is slightly lower than that

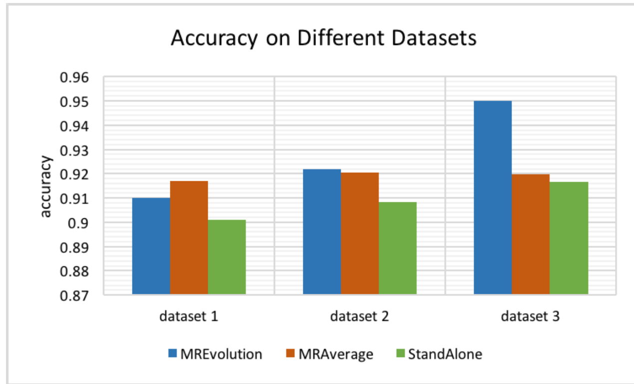


FIGURE 6. Accuracy of the BPNN when finished its training using the three approaches.

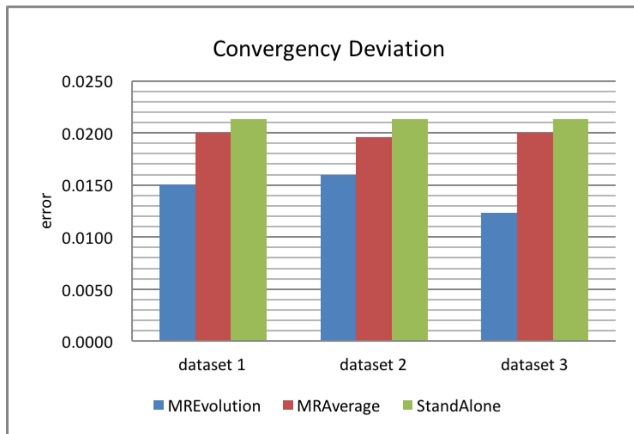


FIGURE 7. Average error when using the three approaches to make the BPNN convergent.

of MRAverage, but MREvolution has much higher efficiency according to the discussion above. Moreover, as the dataset become lager, MREvolution approximately improves the accuracy by 2% to 3% on the other two datasets compared to MRAverage. It means that as the training data become bigger, the model can learn some potential rules in the data that cannot be found with smaller sample sets. The reason is that MREvolution introduces the average of local BPNNs on the same node in the local training stage before leveraging the evolution of local BPNNs. This makes MREvolution has a high ability to avoid the training to get trapped into a local optimum.

Fig. 7 shows the error comparison when the BPNN finished training by the three tools. Apparently, the tool MREvolution has the smallest training error and StandAlone even does not reach the error threshold 0.02 after a 7-hour training. In the experiment, the tool StandAlone just gets a convergence error of 0.0213 when we stop the training process manually.

In Fig. 8, the convergence process of the MREvolution training is given. It shows that the error of the BPNN training has a descending trend and lower the threshold 0.02 eventually.

At the same time, to further verify the effectivity of the approach proposed in the paper, up to 23 popular

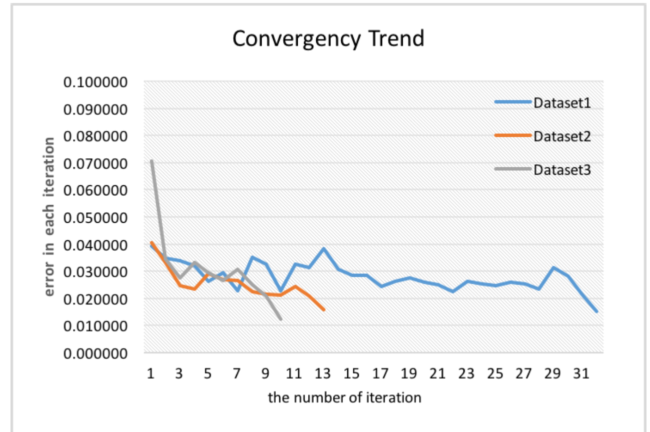


FIGURE 8. The error variation during the training process of MREvolution.

approaches for classification are chosen for accuracy evaluation. These approaches include those based on SVM, Decision Trees(DT), Ensemble Learning(EL), KNN, Logistic Regression(LR) and Random Forests(RF). Our approach, MREvolution, is assigned a category label called MRE (Fig. 9). All these approaches use the same dataset from the application mentioned in the introduction. Because the objective of these experiments is to verify that MREvolution is effective on such high-dimensional big datasets, we let SVM-Fine Gaussian act as the base approach, and the accuracy improving rate of each of the other approaches is evaluated. The accuracy improving rate is calculated by a formula $(a - b)/b$, where a is the accuracy of a specific approach and b is the accuracy of the base approach. Fig. 9 shows the accuracy improving rate of each approach relative to the base approach, SVM-Fine Gaussian. So, the ordinate of Fig. 9 means the accuracy improving rate of each approach that falls into one of the given categories. The trend line in Fig. 9 is only used to highlight the performance of each approach in accuracy improving.

As shown in Fig. 9, MREvolution has the best performance in accuracy improving compared with others. It improves the accuracy about 27.1% relative to that of the base approach. Though Random Forests also shows big advantages to most of the other approaches, MREvolution still gets an accuracy improving rate about 0.7% higher than Random Forests does. At the same time, it shows that traditional BPNN also has a better performance compared with all KNN and SVM based approaches, and has an accuracy improving rate no smaller than those of most approaches based on DT and EL. Nevertheless, the accuracy improving rate of MREvolution is about 6% higher than that of the traditional BPNN. That is to say MREvolution is feasible and effective to be applied in such applications.

B. THE PERFORMANCE OF MREVOLUTION ON 2-HIDDEN-LAYER BPNNs

To analyses the feasibility of MREvolution on BPNNs with multiple hidden-layers, a 4-layer BPNN is designed.

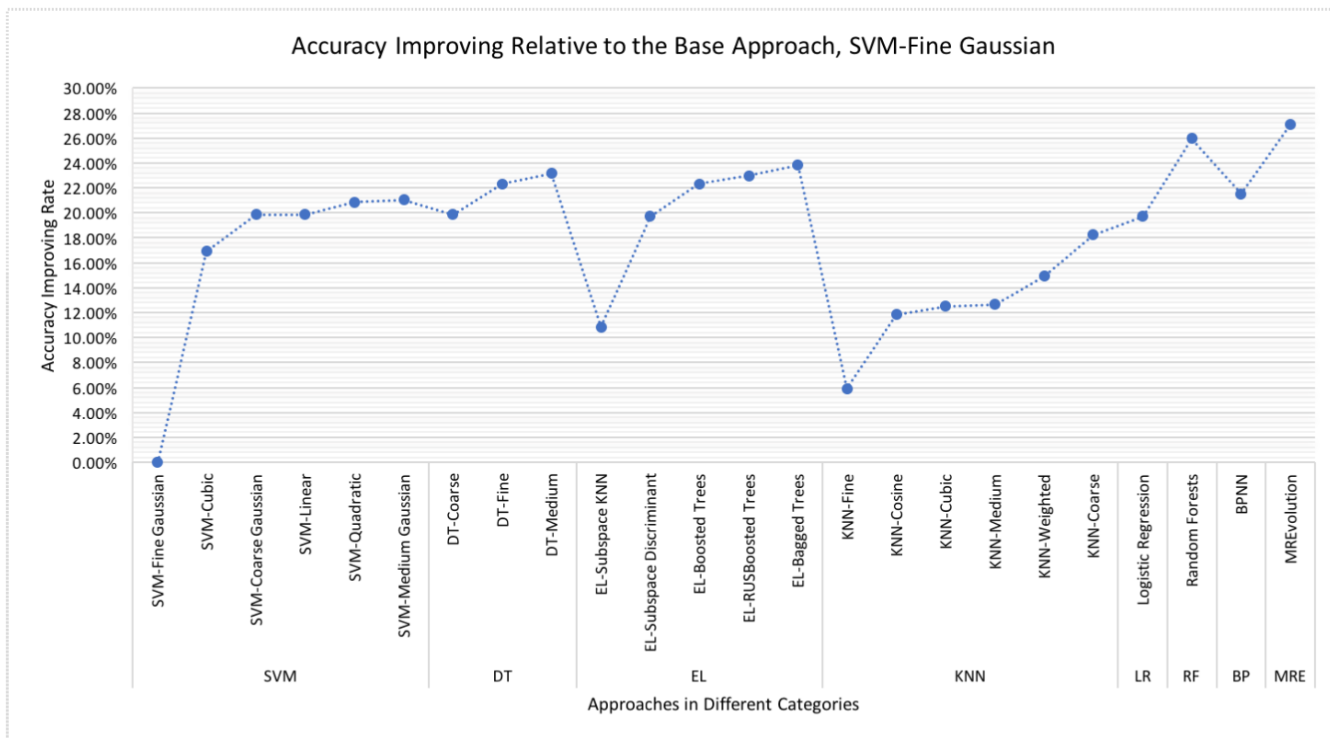


FIGURE 9. Accuracy improving evaluation with SVM-fine gaussian as the base approach.

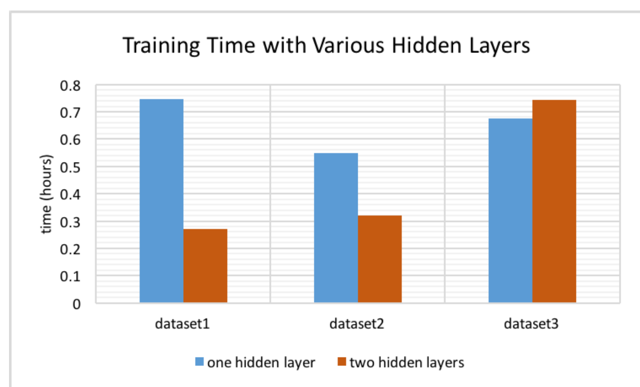


FIGURE 10. Training time taken by MREvolution.

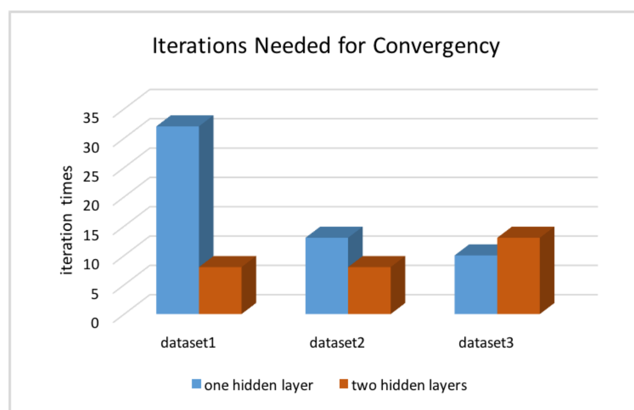


FIGURE 11. Training iteration times needed by MREvolution.

Then, MREvolution is used to train it on the same Hadoop cluster and datasets mentioned above.

Fig. 10 shows that its training time does not increase largely compared with that of the 3-layer BPNN. Instead, it has a decline of 41% and 64% respectively on the first and second datasets. On the third dataset, though the training time of the 4-layer BPNN increases, the ratio is only about 5%. The iteration times for convergency shown in Fig. 11 can interpret the variations of the training time. In reality, as the structure of a BPNN becomes more complex, its precision usually may increase. So, it is possible to become convergent with less iterations.

According to Fig. 12, the accuracy of the 4-layer BPNN declines by about 0.8% to 1.3% compared with that of the 3-layer one on the three datasets. This is really a backward of BPNN, because it may lead a BPNN into overfitting as its structure become more complex and if the dataset has noise [44]. This will be taken into consideration in our future work. Nevertheless, the proposed approach has a good trade-off in efficiency and accuracy.

C. SPEED-UP ANALYSES OF MREVOLUTION

To evaluate the speed-up of our proposed approach MREvolution, the 3-level BPNN mentioned above is trained on different size of clusters with the biggest dataset. Supposing the training time of the BPNN on a cluster with n computing nodes is t and that on a cluster with one computing node is t_0, the speed-up ratio on this size of cluster will be t/t_0, where n means the size of the cluster.

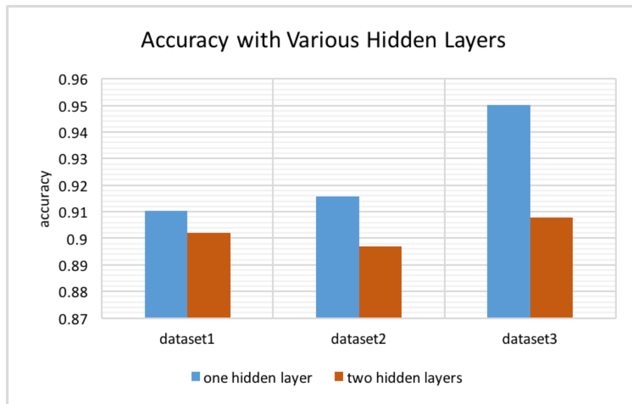


FIGURE 12. **Accuracy comparison of the 1- and 2-hidden-layer BPNNs finished training by MREvolution.**

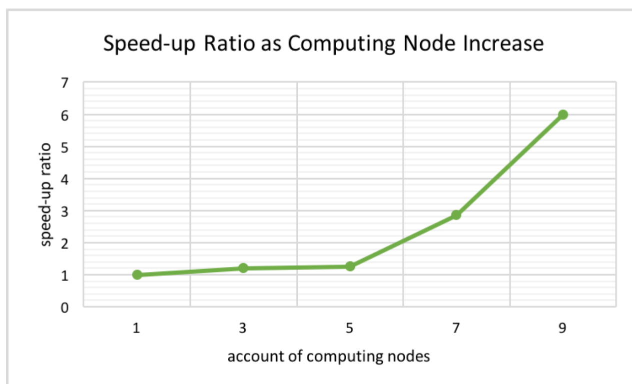


FIGURE 13. **The speed-up ratio of MREvolution on the biggest dataset.**

Fig. 13 shows the variation of the speed-up ratio of MREvolution. It shows that the speed-up ratio on a 9-node cluster is over 6 and the approach proposed will have a high performance. This may provide a guidance to determine the size of the cluster.

D. EVALUATION SUMMARY

According to the experiment results, our proposed approach, MREvolution, shows great advantages in both efficiency and accuracy to current approaches because it introduces the evolution of local BPNNs into the global convergent BPNN generation.

VI. CONCLUSION AND FUTURE WORK

The evolution of the local BPNNs generated by Map tasks is an ideal way to enhance the MapReduce training of a BPNN. It can improve either the training efficiency or the model accuracy on high-dimensional big datasets. At the same time, it verified by the real-world scenario that the approach proposed can work well on big data applications.

In future, the approach proposed will be enhanced in following aspects. (1) Supporting either the structure evolution or weight matrix evolution of local models. Thus, it will be applied to the distributed data-parallel learning of other NNs. (2) Applying the approach into deep NNs and improving its abilities further to avoid model overfitting. (3) Leveraging other distributed data-parallel programming platforms, especially Spark, and analysing their impacts on learning efficiency.

REFERENCES

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362.

[2] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, vol. 4, no. 2, pp. 4–22, Apr. 1987.

[3] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Netw.*, vol. 4, no. 2, pp. 251–257, 1991.

[4] H. Wang, P. X. Liu, J. Bao, X. Xie, and S. Li, "Adaptive neural output-feedback decentralized control for large-scale nonlinear systems with stochastic disturbances," *IEEE Trans. Neural Netw. Learn. Syst.*, to be published.

[5] H. Wang, P. X. Liu, S. Li, and D. Wang, "Adaptive neural output-feedback control for a class of nonlower triangular nonlinear systems with unmodeled dynamics," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 8, pp. 3658–3668, Aug. 2018.

[6] T. C. A. Goh, "Back-propagation neural networks for modeling complex systems," *Artif. Intell. Eng.*, vol. 9, no. 3, pp. 143–151, 1995.

[7] K. Ganeshamoorthy and D. N. Ranasinghe, "On the performance of parallel neural network implementations on distributed memory architectures," in *Proc. 8th IEEE Int. Symp. Cluster Comput. Grid (CCGRID)*, May 2008, pp. 90–97.

[8] A. Brandstetter and A. Artusi, "Radial basis function networks GPU-based implementation," *IEEE Trans. Neural Netw.*, vol. 19, no. 12, pp. 2150–2154, Dec. 2008.

[9] J. Yusheng and Z. Qing, "A method for text categorization using BP network based on Hadoop," in *Proc. Int. Conf. Comput. Inf. Sci.*, Jun. 2013, pp. 818–821.

[10] B. Zhou, W. Wang, and X. Zhang, "Training backpropagation neural network in mapreduce," in *Proc. Int. Conf. Comput., Commun. Inf. Technol. (CCIT)*. Paris, France: Atlantis Press, 2014, pp. 22–25.

[11] Z. Liu, H. Li, and G. Miao, "MapReduce-based backpropagation neural network over large scale mobile data," in *Proc. Int. Conf. Natural Comput.*, vol. 4, Aug. 2010, pp. 1726–1730.

[12] Y. Liu, L. Xu, and M. Li, "The parallelization of back propagation neural network in mapreduce and Spark," *Int. J. Parallel Program.*, vol. 45, pp. 760–779, Aug. 2017.

[13] Z. Chenje and R. Ruonan, "The improved BP algorithm based on mapreduce and genetic algorithm," in *Proc. Int. Conf. Comput. Sci. Service Syst.*, Aug. 2012, pp. 1567–1570.

[14] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.

[15] V. Turchenko, G. Bosilca, A. Bouteiller, and J. Dongarra, "Efficient parallelization of batch pattern training algorithm on many-core and cluster architectures," in *Proc. IEEE Int. Conf. Intell. Data Acquisition Adv. Comput. Syst. (IDAACS)*, vol. 2, Sep. 2013, pp. 692–698.

[16] A. Gutierrez, F. Cavero, R. M. de Llano, and J. A. Gregorio, "Parallelization of a neural net training program in a grid environment," in *Proc. Euromicro Conf. Parallel, Distrib. Netw.-Based Process.*, Feb. 2004, pp. 258–265.

[17] B. Han and T. M. Taha, "Neuromorphic models on a GPGPU cluster," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2010, pp. 1–8.

[18] B. P. Gonzalez, G. G. Sánchez, J. P. Donate, P. Cortez, and A. S. de Miguel, "Parallelization of an evolving artificial neural networks system to forecast time series using OPENMP and MPI," in *Proc. IEEE Conf. Evolving Adapt. Intell. Syst.*, May 2012, pp. 186–191.

[19] W. Kwedlo and K. Bandurski, "A parallel differential evolution algorithm for neural network training," in *Proc. Int. Symp. Parallel Comput. Electr. Eng. (PARELEC)*, Sep. 2006, pp. 319–324.

[20] J. Zhang, K. Ma, F. Feng, Z. Zhao, W. Zhang, and Q. Zhang, "Distributed parallel computing technique for EM modeling," in *Proc. Int. Conf. Numer. Electromagn. Multiphys. Modeling Optim. (NEMO)*, Aug. 2015, pp. 1–3.

[21] L. Singh, A. Narayan, and S. Kumar, "Dynamic fuzzy load balancing on LAM/MPI clusters with applications in parallel master-slave implementations of an evolutionary neuro-fuzzy learning system," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Jun. 2008, pp. 1782–1788.

[22] C. Hacker, I. Aizenberg, and J. Wilson, "GPU simulator of multilayer neural network based on multi-valued neurons," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2016, pp. 4125–4132.

[23] T. Gong, T. Fan, J. Guo, and Z. Cai, "GPU-based parallel optimization and embedded system application of immune convolutional neural network," in *Proc. Int. Workshop Artif. Immune Syst. (AIS)*, Jul. 2015, pp. 1–8.

[24] H. Park, D. Kim, J. Ahn, and S. Yoo, "Zero and data reuse-aware fast convolution for deep neural networks on GPU," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, Oct. 2016, pp. 1–10.

[25] S. Mei, M. He, and Z. Shen, "Optimizing hopfield neural network for spectral mixture unmixing on GPU platform," *IEEE Geosci. Remote Sens. Lett.*, vol. 11, no. 4, pp. 818–822, Apr. 2014.

[26] B. Li, E. Zhou, B. Huang, J. Duan, Y. Wang, N. Xu, J. Zhang, and H. Yang, "Large scale recurrent neural network on GPU," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2014, pp. 4062–4069.

[27] P.-K. Tsung, S.-F. Tsai, A. Pai, S.-J. Lai, and C. Lu, "High performance deep neural network on low cost mobile GPU," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2016, pp. 69–70.

[28] L. Dongming, Y. Chao, L. Yan, L. Chaoran, P. Jiaqi, C. Guifen, and Z. Lijuan, "Research on the precise fertilization based on mapreduce model for BP neural network field," in *Proc. IEEE Int. Conf. Comput. Commun. Internet (ICCCI)*, Oct. 2016, pp. 173–176.

[29] G. Xu, M. Liu, F. Li, F. Zhang, and W. Shen, "User behavior prediction model for smart home using parallelized neural network algorithm," in *Proc. Int. Conf. Comput. Supported Cooperat. Work Design (CSCWD)*, May 2016, pp. 221–226.

[30] R. Zhang and C. Jiang, "The bank risk forewarning model of BP neural network based on the clound computing," in *Proc. Int. Conf. Comput. Netw. Technol.*, Aug. 2012, pp. 91–94.

[31] S. Richly, G. Pueschel, D. Habich, and S. Goetz, "MapReduce for scalable neural nets training," in *Proc. World Congr. Services*, Jul. 2010, pp. 99–106.

[32] S. Venkatraman and S. Kulkarni, "MapReduce neural network framework for efficient content based image retrieval from large datasets in the cloud," in *Proc. Int. Conf. Hybrid Intell. Syst. (HIS)*, Dec. 2012, pp. 63–68.

[33] H. Li, P. Su, Z. Chi, and J. Wang, "Image retrieval and classification on deep convolutional sparknet," in *Proc. IEEE Int. Conf. Signal Process., Commun. Comput. (ICSPCC)*, Aug. 2016, pp. 1–6.

[34] J. Zheng, Q. Ma, and W. Zhou, "Performance comparison of full-batch BP and mini-batch BP algorithm on spark framework," in *Proc. Int. Conf. Int. Conf. Wireless Commun. Signal Process. (WCSP)*, Oct. 2016, pp. 1–5.

[35] K. Grolinger, M. A. M. Capretz, and L. Seewald, "Energy consumption prediction with big data: Balancing prediction accuracy and computational resources," in *Proc. IEEE Int. Congr. Big Data (BigData Congr.)*, Jun./Jul. 2016, pp. 157–164.

[36] L. Ma, X. Huo, X. Zhao, B. Niu, and G. Zong, "Adaptive neural control for switched nonlinear systems with unknown backlash-like hysteresis and output dead-zone," *Neurocomputing*, vol. 357, pp. 203–214, Sep. 2019.

[37] X. Zhao, X. Wang, S. Zhang, and G. Zong, "Adaptive neural backstepping control design for a class of nonsmooth nonlinear systems," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 49, no. 9, pp. 1820–1831, Sep. 2019.

[38] X. Chang, R. Huang, and J. H. Park, "Robust guaranteed cost control under digital communication channels," *IEEE Trans. Ind. Informat.*, to be published.

[39] W. Chen, X. Li, J. Li, and J. Wang, "Enhancing the mapreduce training of BP neural networks based on local weight matrix evolution," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2017, pp. 2829–2835.

[40] Y. Wang, H. Shen, and D. Duan, "On stabilization of quantized sampled-data neural-network-based control systems," *IEEE Trans. Cybern.*, vol. 47, no. 10, pp. 3124–3135, Oct. 2017.

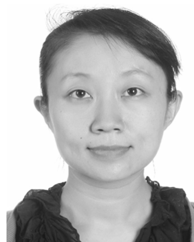
[41] D. Fradkin and D. Madigan, "Experiments with random projections for machine learning," in *Proc. Int. Conf. Knowl. Discovery Data Mining (SIGKDD)*, New York, NY, USA, 2003, pp. 517–522.

[42] K.-T. Fang, D. K. J. Lin, P. Winker, and Y. Zhang, "Uniform design: Theory and application," *Technometrics*, vol. 42, no. 3, pp. 237–248, 2000.

[43] K. Deep and M. Thakur, "A new mutation operator for real coded genetic algorithms," *Appl. Math. Comput.*, vol. 193, no. 1, pp. 211–230, Oct. 2007.

[44] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

WANGHU CHEN received the Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, in 2009. He was a Visiting Scholar with the University of California, San Diego, from 2013 to 2014. He is currently a Professor with the Institute of Computer Science and Engineering, Northwest Normal University, China. His research interests include big data and cloud computing. He is PIs of five research projects, including National Science Funds and has published 40 more than articles. He is also PC member of international workshops.



JING LI received the master's degree from Lanzhou University, China, in 2013. She is currently an Associate Professor with the Institute of Computer Science and Engineering, Northwest Normal University, China. Her research interests include big data, scientific workflow, and service-oriented computing.

XINTIAN LI received the bachelor's degree in computer science from Northwest Normal University, in 2016, where he is currently pursuing the master's degree. His research interests include big data and cloud computing.

LIZHI ZHANG received the bachelor's degree in computer science from Bohai University, in 2018. He is currently pursuing the master's degree with Northwest Normal University. His research interests include big data and cloud computing.

JIANWU WANG received the Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, in 2007. He is currently an Assistant Professor with the Department of Information Systems, University of Maryland, Baltimore County (UMBC). He has published 60 more than articles with more than 800 citations. His research interests include big data, scientific workflow, distributed computing, service-oriented computing, and end-user programming. He is also program committee member for over 30 conferences/workshops, and a Reviewer of over ten journals or books. He is an Associate Editor or Editorial Board Member of four international journals, a Co-Chair of three related workshops.