Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

IMPLEMENTATION OF PARALLELISM OF BACK PROPAGATION NEURAL NETWORK ALGORITHM ON A DISTRIBUTED COMPUTING SYSTEM

SHAHAZAD NIWAZI QURASHI *
Department of Health Informatics, College of Public Health and Tropical Medicine, Jazan University, Jazan, Jizan, Kingdom of Saudi Arabia. *Corresponding Author Email: squrashi@jazanu.edu.sa

MOHD SHAHNAWAZ ANSARI
Department of Computer Science and Engineering, School of Engineering, Eklavya University, Damoh, (M.P), India. Email: shahnawaznbd@gmail.com,

FARRUKH SOBIA
Department of Health Education & Promotion, College of Public Health and Tropical Medicine, Jazan University, Jazan, Jizan, Kingdom of Saudi Arabia. Email: fsobia@jazanu.edu.sa

RABINDRA K. BARIK
School of Computer Applications, Kalinga Institute of Industrial Technology, Bhubaneswar, India.
Email:rabindra.mnnit@gmail.com

Abstract

Neural networks are composed of many small processors that work simultaneously on the same task. They can learn from training data and use their knowledge to compare patterns in a dataset. Combining the strengths of parallel processing and distributed computing, the neural network can enhance the processing times for both the learning and execution stages to efficiently calculate the most probable output with a remarkable degree of accuracy. The aim of this research was to design, implement, and demonstrate the enhanced computational speed of a generalized large-scale neural network with broad-based applications using parallel computing. As a result, we evaluated distributed computing systems and compared the performance of different neural network training functions to determine which one worked best for good system performance. The results indicate that the proposed method outperforms other methods in terms of accuracy and convergence time. The study suggests that parallelism of the backpropagation neural network model can lead to faster training convergence time and higher accuracy of the results. The system takes input data and runs on different systems in parallel.

Keywords: ANN, Backpropagation, Parallel Processing, Distributed system, MATLAB.

1. INTRODUCTION

In some practical applications of neural networks, fast responses to external events within an extremely short time are highly demanded and expected. However, the extensively used gradient descent-based learning algorithms obviously cannot satisfy real-time learning needs in many applications, especially large-scale applications, and when higher learning accuracy and generalization performance are required. This developed neural network has a parallel-distributed information processing structure that consists of a collection of simple processing elements, which are interlinked by means of signal channels or connections. The most useful property of neural network design is that it has the ability to recognize input it did not see before while maintaining the basic demands of

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

its specific applications. A MATLAB-based face recognition using PCA with a back propagation neural network was proposed [Dhoke *et al.* 2014]. The characteristics that determine the efficiency of this program are Speed, Accuracy, and real-time capability (time). It is possible also to compare the performance of two parallelization strategies for a backpropagation neural network on a cluster computer: exemplar parallel and node parallel strategies [Pethick e*t al.*2003]. With the integration of parallel computing, the computational speed of the neural network can be further increased and as the computing power of personal computers increases with the times, the reality of achieving real-time computations of extremely large problems becomes almost a possibility. Moreover, the benefits of training several ANNs in parallel compared to other forecasting

methods used in the competition. Indeed, training several ANNs in parallel yields a better fitting of the weights of the network and allows for training in a short time many ANNs for different time series [Cruz-López *et al*. 2017]. Backpropagation is a widely used method for calculating derivatives inside deep feedforward neural networks. It is short for backward propagation of errors. The algorithm is used to train the neural network by calculating the gradient of the loss function with respect to each of the weights of the network. This enables every weight to be updated individually to gradually reduce the loss function over many training iterations. Backpropagation involves the calculation of the gradient proceeding backward through the feedforward network from the last layer through to the first. The practical implementations of such a powerful tool span across various zones are the most interests in engineering applications. From basic image recognition problems to time-critical military installments in the form of target tracking and identification to more complex uses in strand recognition, there is a huge commercial potential for developing a system.

Section 2 outlines about the artificial neural networks and the literature based on implementing distributed environments, Section 3 describes neural network applications in parallel computing, and Section 4 discusses the approach to solving the present problem and the importance of neural networks to solve the problem, Section 5 Provides the details of the training algorithm that has been used, and the architecture of the parallel neural network, and implementation of distributed environment system. Section 6 shows the results of training and testing in the case of single processing and parallel processing of the data, and finally, Section 7 Concludes the study and future prospects.

2. RELATED WORK

Many researchers have been dedicated to implementing computationally expensive ANN algorithms on parallel or distributed computing systems. An ANN consists of an enormous number of massively interconnected nonlinear computational elements (neurons). Each neuron receives inputs from other neurons, performs a weighted summation, applies an activation function to the weighted sum, and outputs its results to other neurons in the network. To address large-scale neural network training problems, a customized parallel computing platform called cNeural was proposed [Gu *et al*. 2013].

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

Artificial Neural Networks (ANNs) need as much as possible data to have high accuracy, whereas parallel processing can help us to save time in ANNs training. [Sharif *et al.* 2018]. The simulation of an ANN comprises a simulation of the learning phase and the recall phase. The learning and recall of neural networks can be represented mathematically as linear algebra functions that operate on vectors and matrices [Means *et al.* 1994]. Thus, standard parallelization schemes can be exploited for both. However, the focus of parallel neural simulations has been more on the learning phase, which is the most computation-intensive part of neuroprocessing. Parallel architectures for simulating neural networks can be subdivided into general-purpose parallel computers and neurocomputers.

3. NEURAL NETWORK APPLICATIONS

To efficiently utilize parallel processing for speeding up neural network training, we should be aware of which types of networks and training sets are used in today's neural applications. Mainly large applications, where parallel processing is of interest, will be described. The Feed-forward neural network with a single hidden layer is assumed unless otherwise stated.

3.1 Speech Recognition

Several research groups are working on the difficult task of continuous speech recognition. Promising results using neural networks are described in [15] however, the network and training set need to be large. For recognizing 300 sentences (speaker independently), the system achieved 4 to 5 percent error, which is competitive with a statistically based system. For a larger recognition problem, the error for the neural network system became twice that of the best mainstream system. However, the mainstream system is larger than the neural–network–based system. As the speech networks get larger, they tend more toward networks that are not fully connected [16]. This is in the form of fully connected subnets.

3.2 Parallel Implementation of Back Propagation Neural Network

Feed-forward neural network with backpropagation learning is the most widely used configuration.

Back-propagation neural network algorithm uses input training samples and their respective desired output values to learn to recognize specific patterns, by modifying the activation values of its nodes and weights of the links connecting its nodes [Joshi and Cheeran, 2014].

The backpropagation learning for a bigger network with a large training set takes a long time (in terms of days), it becomes imperative to look at parallel implementation schemes to reduce this large training time and the implementations depend very much on the type of parallelism used. The parallelism can be regarded as the practical solution in solving large workloads and in achieving an optimal training time and generalization ability, possessing the problem for generating a suitable comprehensive classifier will positively to the time and maintain accuracy at the same time [Mohamad *et al*., 2012].

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

3.3 Parallelization of Feed-Forward Neural Network

The basic terminology used in parallel implementation of back propagation neural networks is described in the following terms.

Training Set: It consists of several training patterns, each given by an input vector and the corresponding output vector.

Network Size: A network of N_i inputs units, N_i hidden units, and N_o output units is for short written N_i *N_i * N_o. Note that the word *network* is used for neural networks in this study, not for processor topology networks.

Training Iteration: It denotes one presentation of the whole training set.

Weight Updating Strategies: Three different approaches are used:

- Learning by pattern (LBP) updates the weights after each training pattern has been presented.
- Learning by block (LBB) updates the weights after a subset of the training patterns has been presented
- Learning by epoch (LBE) updates the weights after all patterns have been presented.

Weight Update Interval: The number of training patterns presented between weight updates is termed μ. For LBP, μ=1, whereas for LBE μ=p, where p is the number of training patterns in the training set. The backpropagation algorithm reveals for different kinds of parallelism

Training Session Parallelism: It starts training sessions with different initial training parameters on different processing elements.

Training Set Parallelism: It splits the training set across the processing elements. Each element has a local copy of the complete weight matrix and accumulates weight change values for the given training patterns. The weights are updated using learning by block.

Pipelining: It allows the training patterns to be “pipelined” between the layers, that is, the hidden and output layers are computed on different processors. While the output layer processor calculates output and error values for the present training pattern, the hidden layer processor processes the next training pattern. The forward and backward phases may also be parallelized in a pipeline.

Node Parallelism: It computes the neurons within a layer in parallel (neuron parallelism). Further, the computation within each neuron may also run in parallel. (Nord Strom [2] names this node parallelism as weight or synapse parallelism). In this method, the weights can be updated using learning by pattern.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

3.4 Distributed Computing for Backpropagation Parallelism

In this section, the first training set parallelism has been described. Then the other main parallel degree, network partitioning, is detailed by describing pipelining and node parallelism.

3.4.1 Training Set Parallelism

The great advantage of parallel implementations is the number of different ANNs that can be trained in the same amount of time. It is a good alternative when many different time series have to be forecasted [Cruz-López *at el*. 2017]. Training set parallelism is also called data parallelism because the training set is partitioned, not the training program. Each processing element (PE) has a local copy of the complete weight matrices and accumulates weight change values for the given training patterns. The neural network weights must be consistent across all the PEs, thus weights are updated in a global operation (learning by block, epoch). The weight change values of each PE are summed and used to update the local weight matrices.

4. METHODOLOGY

The objective was to run the system in a distributed environment to reduce time and to improve the performance of the system. The neural network has a parallel-distributed information processing structure that consists of a collection of simple processing elements, which are interlinked by means of signal channels or connections. The operation of the system was divided into five phases: Input Data, Data distribution among different processors, Training, Testing, and output.

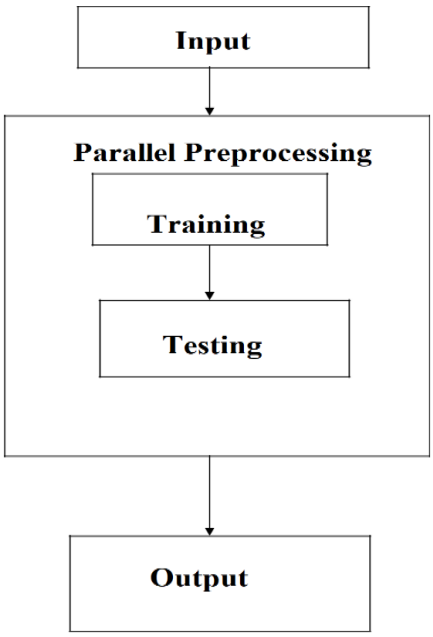

Figure 4.1: Block Diagram of System Overview

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

4.1 Input Data

In this phase, a three-layer architecture of a back propagation neural network is considered in a distributed environment for the plate vibration problem. The input layer consists of two inputs, (i) specified boundary condition and (ii) aspect ratio (*m*) of the elliptic plate. The digitized values for the input parameters corresponding to the boundary conditions are fed as 2 for clamped, 1 for simply supported, and 0 for free boundary condition. The output layer of the artificial neural network architecture consists of one output in the form of the corresponding frequency parameter (f) obtained from the *RR* method using the BCOPs. However, the number of nodes in the hidden layer has been taken as 10 and 15 for comparison of the results.

Table 4.1: Training Patterns for Artificial Neural Network

	Boundary Condition		
	2	**1**	**0**
Aspect ratio (m)	1.00	1.00	1.00
	0.90	0.90	0.90
	0.80	0.80	0.80
	0.70	0.70	0.70
	0.60	0.60	0.60
	0.50	0.50	0.50
	0.40	0.40	0.40
	0.30	0.30	0.30
	0.20	0.20	0.20
	0.10	0.10	0.10
Frequency parameter (f)	10.216	4.9351	5.3583
	11.442	5.5282	5.8381
	13.229	6.3935	6.1861
	15.928	7.7007	6.4185
	20.195	9.7620	6.5712
	27.377	13.213	6.6705
	40.646	19.514	6.7321
	69.147	32.813	6.7654
	149.66	69.684	6.7778
	579.36	262.98	6.7781

4.2 Parallel Computing

Parallel computing is the simultaneous use of multiple computing resources to solve a computational problem. There are several different forms of parallel computing bit level, instruction level, and task parallelism. The test on larger distributed systems would help to accurately measure the systems' performance in more diverse environments Mohamad *et al*., (2012). A distributed Computing environment enables us to coordinate and execute independent operations simultaneously on a cluster of computers, speeding up the execution of jobs that contain large amounts of data. A job is some large operation that needs be to performed in a distributed environment. A job is broken down into segments called tasks. The job is divided into identical tasks, but tasks do not have to be identical, in a distributed environment the job and its tasks are defined in the client processor or user processor. The client uses a Distributed environment to perform the definition of jobs

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

and tasks. The job manager is the part of the environment that coordinates the execution of jobs and the evaluation of their tasks. The job manager distributes the tasks for evaluation to the different systems called workers as shown in figure 4.2.

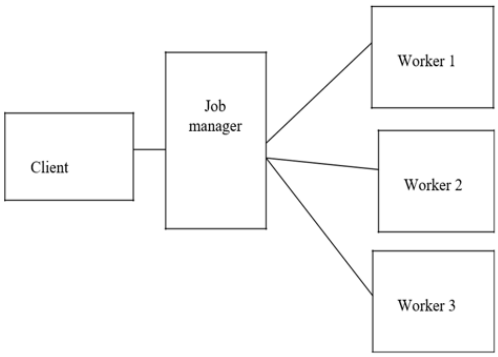

Figure 4.2: Basic Parallel Computing Configuration

4.3 Training

The propagation neural network is used for behavior classification. The neural network was constructed for this research. It has two units in the input layer and one unit in the output layer. The network consisted of only one hidden layer and selected the number of hidden nodes in the hidden layer. In this phase, the network has been trained and tested by the trainlm training function of neural network tool in MATLAB environment.

4.4 Testing

In the training phase, the input pattern and output pattern were given to the network. But in this phase, only the input pattern is given. After the initialization of the weights, the input units of the input layer are activated with the input patterns that have been taken from the input file. The outputs of the input layer are propagated toward the output layer similarly to training. The calculated output from the output unit of the output layer was considered as the desired output pattern.

4.5 Parameters for the Neural Network

A back propagation artificial neural network was used in this research work. The parameter setting for the training of the neural network is of most important. Number of inputs i.e. how many units are there in the input vector. Number of outputs i.e. how many units are there in the output vector. RMS error is a value that can be adjusted, for the accuracy of the output

Table 4.5: Parameters for the neural network

Parameters	Values
Number of Input	2
Number of Output	1
Number of hidden layers	15
RMS-Error	0.000001

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

5. IMPLEMENTATION

5.1 Multi-Layer Feed-Forward Networks with Bp Learning

A three-layer feed-forward network is shown in Figure 5.1(a); the network is called fully connected. Because there are all-to-all connections between two adjacent neuron layers. The number of neurons (also called units) in each layer is N_i, N_h, and N_o for the input, hidden, and output neuron layers, respectively. The network can be extended to any number of layers; however, because most applications use two-weight layers, the description here has been restricted to two-layer networks. The BP learning phase for a pattern consists of a forward phase followed by a backward phase. The training algorithm of backpropagation involves four stages: Initialization of weights, Feedforward, Backpropagation of errors, and Updation of the weights and biases.

There are two types of learning in backpropagation: sequential learning and batch learning. In sequential learning a given input pattern is propagated forward, the error is determined and back propagated, and the weights are updated. In batch learning the weights are updated only after the entire set of training networks has been presented to the network. Thus, the weights update is only performed after every epoch.

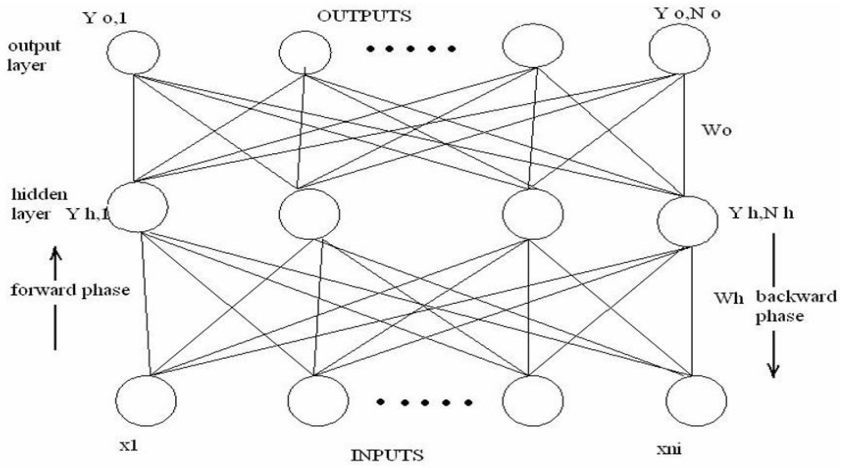

Figure 5.1(a): A Three Weight Layer Feed-Forward Neural Network

To train the network, the proposed training algorithm used in the backpropagation algorithm with set of steps. The main steps are as follows:

- Initialize the weights to small random values.
- Select a training vector pair (input and the corresponding output) from the training set and present the input vector to the inputs of the network.
- Calculate the actual outputs in the forward phase.
- According to the difference between actual and desired outputs (error). Adjust the weights W_o and W_h to reduce the difference this is the backward phase.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

- Repeat from step 2 for all training vectors.
- Repeat from step 2 until the error is acceptably small.

In the forward phase the hidden layer weight matrix W_h is multiplied by the input vector $X= (X_1, X_2, X_3,.....X_n)^T$ to calculate the hidden layer output

$$Y_{h,j}=f(\sum W_{hji}*X)$$

Where W_h,ji is the weight connecting input unit I to unit j in the hidden neuron layer. The function f is a nonlinear activation function. Normally the S-shaped sigmoid function $F(\alpha) =1/1+e^{-\alpha}$ is used. It compresses the output value to lie in (0, 1), as shown in Figure 5.1(b). Moreover, the function is differentiable, which is a demand of the training algorithm.

The output from the hidden layer $Y_{h,j}$ is used to calculate the output of the network $Y_{o,k}$

$$Yo,k=f(\sum Wo,kj*Yh,j)$$

The error measure E_p for a training pattern p is given by

$$E_p=1/2\sum (d_{p,k}-Y_{p,o,k})^2$$

The overall error measure for a training set of P patterns is $E=\sum E_p$

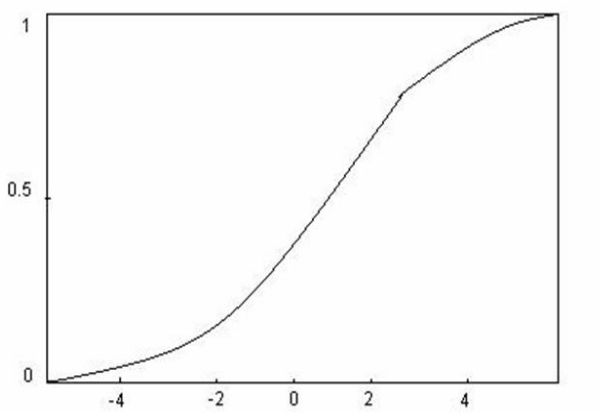

Figure 5.1(b): The Sigmoid Function $f(\alpha) =1/1+e^{-\alpha}$

In the following expressions, the pattern index p has been omitted on all variables to improve clarity. In the backward phase the target, d, and output, Y_o, are compared and the difference (error) is used to adapt the weights to reduce the error. The error used to update the weights can be shown to be

$$\delta_{o,k} =Y_{o,k}(1-Y_{o,k})(d_k-Y_{o,k})$$

Like computing the output delta error, the hidden delta error value for neuron j is

$$\delta h,j =Yh,j(1-Yh,j)\sum \delta o,k\ Wo,kj$$

The error is not explicitly given and is computed based on the impact of the fan-in of the output delta errors. To perform the steepest descent in the weight space, the weight changes become

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

$$W_{o,kj} = \eta \delta_{o,k} Y_{h,j}$$

$$W_{h,ji} = \eta \delta_{h,j} X_i$$

Where η is the learning rate coefficient.

If learning by pattern is applied, the output layer weights are changed to $W_{o,kj}$ $W_{o,kJ}=W_{o,kj}+\eta \delta_{o,k}*Y_{h,j}$

The hidden layer weights are updated accordingly $W_{h,ji}=W_{h,ji}+\eta\delta_{h,j}*X_i$

The training continues for each vector in the training set until the error for the entire set becomes acceptably small.

5.2 MATLAB Configurations for Distributed Computing

Generally, there is not much difficulty in deciding which machines will run worker processes and which will run client processes. Worker sessions usually run on the cluster of machines dedicated to that purpose. The client session of MATLAB usually runs where MATLAB programs are run, often on a user's desktop. The job manager process should run on a stable machine, with adequate resources to manage the number of tasks and amount of data expected in our distributed computing applications. The following table shows what products and processes are needed for each of these roles in the distributed computing configuration.

Table 5.2: Distributed Computing Configuration

Session	Products	Process
Client	Distributed Computing Toolbox	MATLAB with toolbox
Worker	MATLAB Distributed Computing Engine	worker; mdce service(if using a job manager)
Job manager	MATLAB Distributed Computing Engine	mdce service; job manager

5.2.1 Worker Configuration

The different workers running on different machines can be connected using the following configurations. The 3 workers running on a machine with IP address 172.31.5.105 and the 4 workers running on the machine with IP address 172.31.5.96.

5.2.2 Job Manager Configuration

For the job Manager Configuration First Parallel pull-down menu must be selected on the MATLAB desktop. to open the Configurations Manager, we Click the Parallel Manage Configurations. The first time when the Configurations Manager opens, it lists only one configuration called local, which at first is the default configuration and has only default settings.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

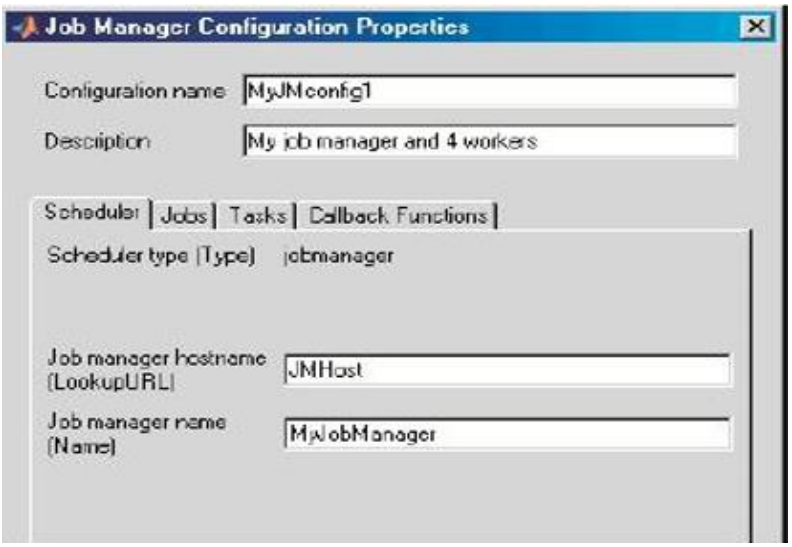

Figure 5.2.2(a): Job Manager Configuration Properties 1

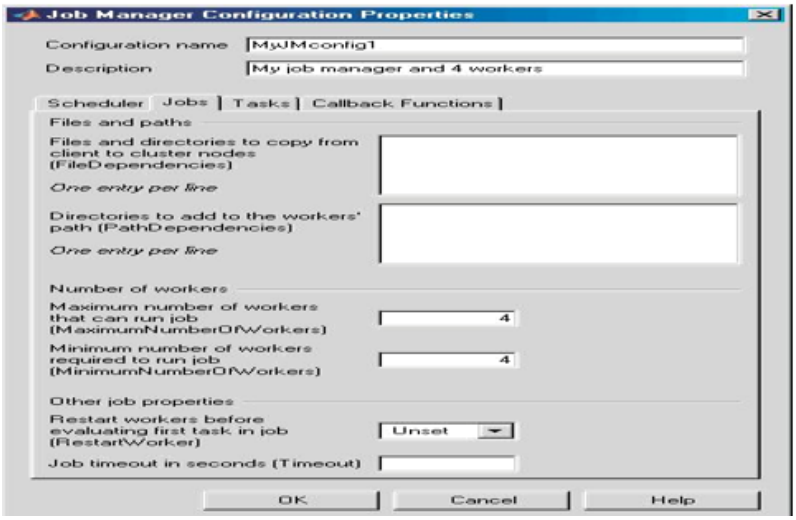

Figure 5.2.2(b): Job Manager Configuration Properties 2

To create a new configuration whose type of scheduler is a job manager in the Configurations Manager, click New Job Manager. This opens a new Job Manager Configuration Properties dialog box. Enter a configuration name, such as “MyJMconfig1”, and a description as shown in Figure 5.2.2. In the Scheduler tab, enter the hostname for the machine on which the job manager is running and the name of the job manager. If entering information for an actual job manager already running on your network, enter the appropriate text. In the Jobs tab, enter 4 and 4 for the maximum and minimum number of workers. This specifies that jobs using this configuration require at least four workers and use no more than four workers. Therefore, the job runs on exactly four workers, even if it has to wait until four workers are available before starting. After creating a job, apply either configuration to that job as a way of specifying how many workers it should run on.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

5.3 Training

The training process requires a set of examples of proper network behavior - network inputs p and target outputs t. In the previous section, it has been already mentioned that the backpropagation neural network is used for behavior classification. In this work neural network is constructed with 2 units in the input layer and one unit in the output layer. Only one hidden layer is used in this work and there is an option to choose a different number of hidden nodes for the system. This can be simply done by changing the value of the variable for hidden units in the implementation. A weight value is associated with each of the connections. The output of the neural network will be our desired target output.

5.4 Different Training Algorithms

According to [Gu et al. 2013], a more complex algorithm leads to increased complexity of memory management, synchronization details, and tracking of processes involved in tasks, especially when tasks are generically distributed. Different test runs were made for each of the following training algorithms with varying numbers of hidden nodes. Table 5.4 in this work shows the training function used along with some other training functions.

Table 5.4: Description of Different Neural Network Training Functions

Function	Description
Trainbfg	BFGS quasi-Newton method. Requires storage of approximate Hessian matrix and has more computation in each iteration than conjugate gradient algorithms, but usually converges in less iteration.
Trainoss	One-step secant method. Compromise between conjugate gradient methods and quasi-Newton methods.
Trainlm	Levenberg-Marquardt algorithm. Fastest training algorithm for networks of moderate size. Has a memory reduction feature for use when the training set is large.

5.5 Testing

The system has already been trained with the normal given data. During the training phase, both the input pattern and output pattern are given to the network. However, in this phase, only the input pattern is given. After initializing the weights, the input units of the input layer are activated with the input patterns taken from the input file. The outputs of the input layer are propagated towards the output layer, like training. The calculated output from the output unit of the output layer is considered as the desired output pattern. This output will indicate whether the given input pattern represents the desired output or not.

6. RESULT EVALUATION

This section discusses experimental issues and compares single and parallel data processing with the trainlm training function. The experiment examines the number of hidden layers required to train the neural network and compares data processing in the case of serial and parallel processing. The experiments use different data for plate vibration analysis. They compare single processing and parallel processing to increase system performance using a distributed system in the MATLAB environment.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

In a preliminary experiment, a training function with 10 hidden units was used to train the network until the RMS error value was reduced to an acceptable level. The training sessions were collected for plate vibration analysis. The results of the training for plate vibration analysis are shown in Table 6.1 with 10 and 15 hidden nodes. In the table, the number of hidden units denotes the number of neurons in the first layer.

The number of epochs represents the number of iterations needed to converge the network with the desired accuracy. The performance goal status indicates whether the desired goal is achieved or not. Table 6.1 compares single and parallel data processing for training the network to increase system performance using a distributed system in the Matlab environment.

6.1 Training

Table 6.1: Comparison of Training Patterns between Single and Parallel System

	No. of Epocs/sec	Hidden Units	Performance goal
Single	8520	10	Achieved
Parallel	1326	10	Achieved
Single	2761	15	Achieved
Parallel	821	15	Achieved

6.2 Experiment

6.2.1 Single processing with hidden units 10

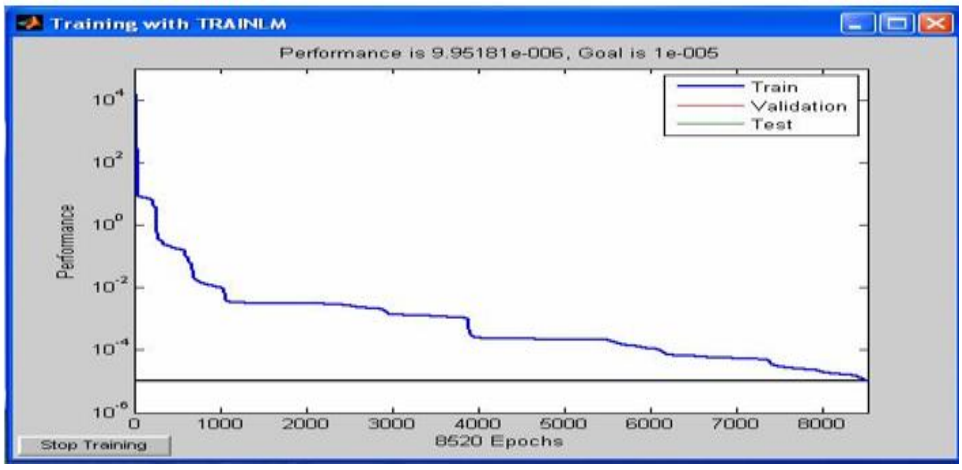

Figure 6.1.1: Output of Training Dataset-using Trainlm Training Function

The graph shown in Figure 6.1.1 represents the output of the training of the network and 8520 epochs have been taken to get trained the network using the trainlm train function. and the performance goal of the network has been achieved.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

6.2.2 Parallel processing with hidden units 10

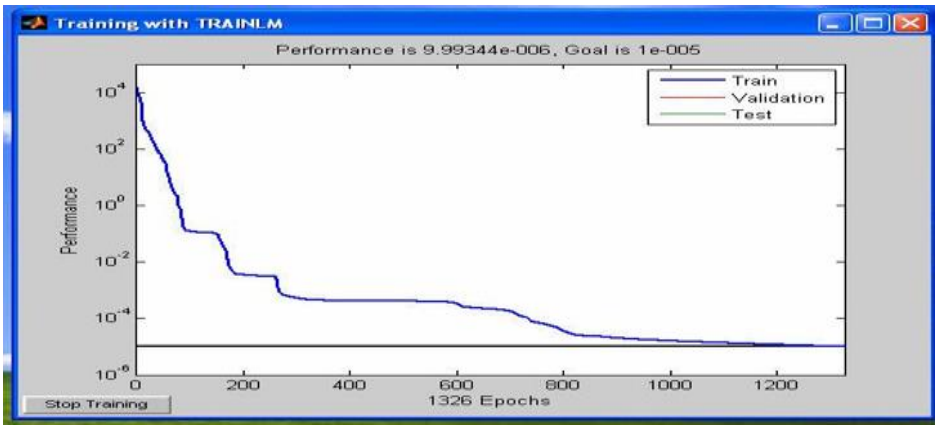

Figure 6.2.2: Output of Training Dataset-using Trainlm Training Function

The graph shown in Figure 6.2.2 represents the output of the training of the network and 1326 epochs have been taken to get train the network using the trainlm train function. In this case, the performance goal of the network has been achieved.

6.2.3 Single processing with hidden units 15

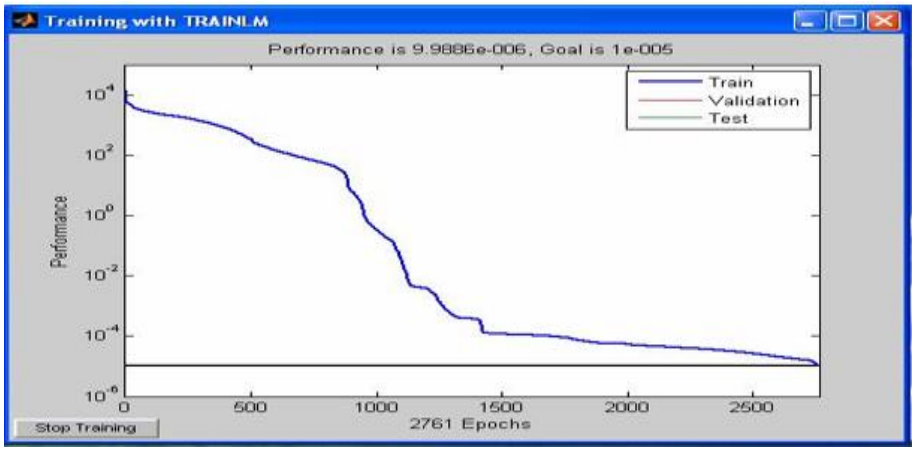

Figure 6.2.3: Output of Testing Dataset-using Trainlm Training Function

The graph shown in Figure 6.2.3 represents the output of the training of the network and 2761 epochs have been taken to get trained the network using the trainlm train function. In this case, the performance goal of the network has been achieved.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

6.2.4 Parallel processing with hidden units 15

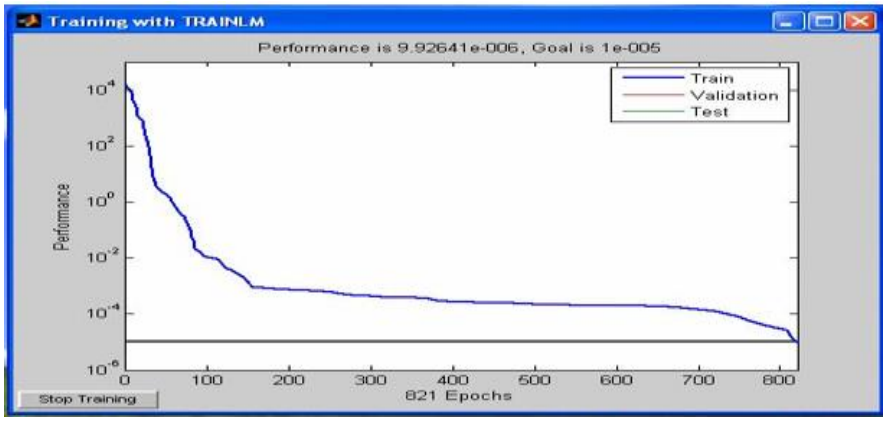

Figure 6.2.4: Output of Testing Dataset-using Trainlm Training Function

The graph shown in Figure 6.2.4 represents the output of the training of the network and 821 epochs have been taken to get train the network using the trainlm train function. In this case, the performance goal of the network has been achieved.

6.3 Different Node Status in Distributed Environment

6.3.1 Nodestatus1

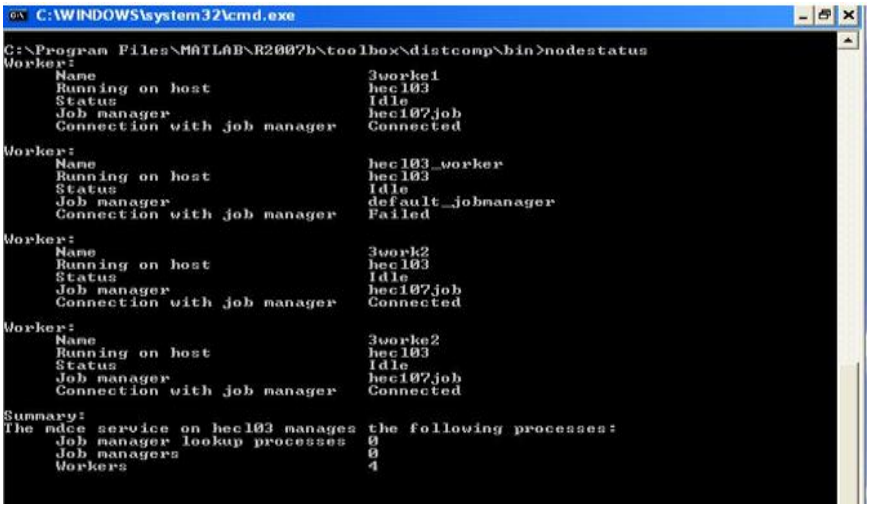

Figure. 6.3.1: Node Status of the Worker on System hec103

As shown in Figure 6.3.1 this is one of the node statuses in a distributed environment, there are four workers are running in this hec103 host, and three workers are (3worke1, 3work2, 3worke2) related to the job manager name with hec107job on the system 172.31.5.92. And one worker's name with hec103_worker is not connected with the job manager; the connection is failed because the default_ job manager is not running on the system 172.31.5.92.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

6.3.2 Nodestatus2

As shown in figure 6.3.2 this is one of the node statuses in a distributed environment, there are two workers are running in this hec104 host and two workers are (4worker1,4worker2) relate to job manager name with hec107job on the system 172.31.5.92.

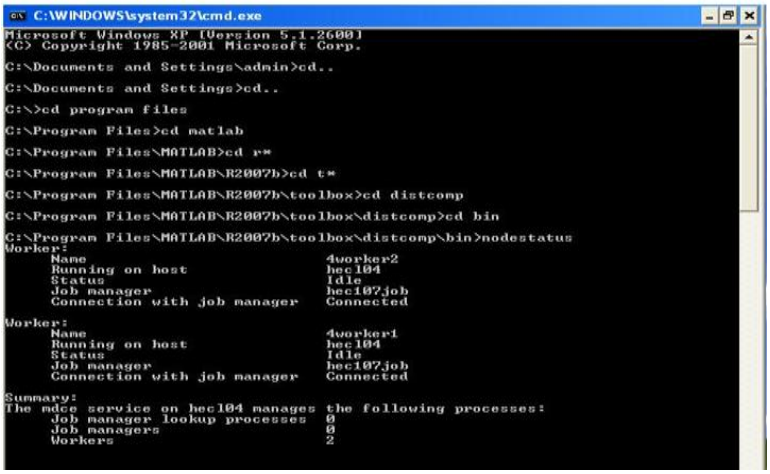

Figure 6.3.2: Node Status of the Worker on System hec104

6.3.4 Nodestatus3

As shown in figure 6.3.3 this is the job manager node status in a distributed environment, there are on job manager lookup processes and one job manager and one worker, hec107_worker is not connected with the default job manager, and the connection is lost, because the default job manager is not running on this node hec107 and five other workers relate to this job manager name with hec107job as described in the above node statuses.

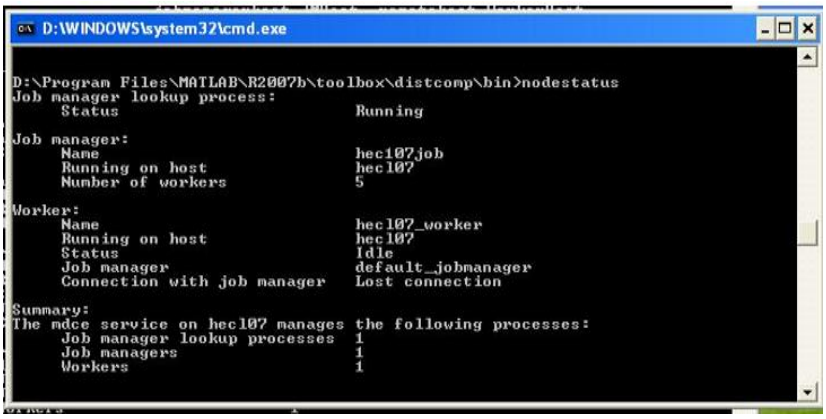

Figure 6.3.3: Node Status of the Job Manager on System hec107

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

7. CONCLUSION

The implementation of parallelism of back propagation neural network algorithm on a distributed computing system with good performance has been demonstrated. The parallelism of the backpropagation neural network has been trained and tested for the analysis of vibration data. It has been observed that the convergence time for the training of backpropagation neural networks by parallel processing is faster as compared to single processing. This is because the data has been processed parallelly. In the case of parallel processing, training of back propagation neural network has achieved the performance goal with the desired accuracy of the results.

The training of the backpropagation neural network algorithm has been performed by using trainlm training function of the Mat Lab environment with 10 and 15 nodes in the hidden layer of the network model. It has also been observed that with 15 numbers of nodes in the hidden layer of the network takes less time to converge. It is also tested that while increases the number of nodes in the hidden layer the accuracy of the results does not increase. The proposed research work gives a faster estimation for the analysis of vibration data. It is well documented that parallelism of the backpropagation neural network model gives the faster training convergence time and higher accuracy of the results.

7.1 Limitations and Future Work

In this research work, there is no fixed size of the cluster of the distributed systems for the particular to problem under test. Only one can assign a number of workers to the Job manager depending on the achievement of the higher performance of the system. In future work, this can be trained with different training functions of the Mat Lab environment. As shown in this research work, back propagation neural networks can be successfully implemented in the distributed environment system for data processing. The same experiments should also be conducted with other types of neural networks to see if the different types can improve the performance of the system as we got the experiment results with the backpropagation neural network.

References

1) Rumelhart, D. E., Durbin, R., Golden, R., & Chauvin, Y. (1995). Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*, 1-34.

2) Kashem, M. A., Akhter, M. N., Ahmed, S., & Alam, M. M. (2011). Face recognition system based on principal component analysis (PCA) with back propagation neural networks (BPNN). *Canadian Journal on Image Processing and Computer Vision*, *2*(4), 36-45.

3) Meng, Y. (2004). Speech recognition on DSP: Algorithm optimization and performance analysis. *The Chinese University of Hong Kong*, 1-18.

4) Le, C. G. (1993). *Application of a Back-propagation neural network to isolated-word speech recognition* (Doctoral dissertation, Monterey, California. Naval Postgraduate School).

5) Mehrotra, K., Mohan, C. K., & Ranka, S. (1997). *Elements of artificial neural networks*. MIT Press.

Tianjin Daxue Xuebao (Ziran Kexue yu Gongcheng Jishu Ban)/
Journal of Tianjin University Science and Technology
ISSN (Online):0493-2137
E-Publication: Online Open Access
Vol: 56 Issue: 11:2023
DOI: 10.5281/zenodo.10223972

6) Dhoke, P., & Parsai, M. P. (2014). A MATLAB-based Face Recognition using PCA with Back Propagation Neural network. *International Journal of Innovative Research in Computer and Communication Engineering*, *2*(8), 5291-5297.

7) Joshi, S. C., & Cheeran, A. N. (2014). MATLAB-based back-propagation neural network for automatic speech recognition. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, *3*(7), 10498-10504.

8) Mohamad, M., Saman, M. Y. M., & Hitam, M. S. (2012). Parallel Training for Back Propagation in Character Recognition. *University Malaysia of Terengganu*.

9) Cruz-López, J. A., Boyer, V., & El-Baz, D. (2017, May). Training many neural networks in parallel via back-propagation. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (pp. 501-509). IEEE.

10) Pethick, M., Liddle, M., Werstein, P., & Huang, Z. (2003, November). Parallelization of a backpropagation neural network on a cluster computer. In *International conference on parallel and distributed computing and systems (PDCS 2003)*.

11) Gu, R., Shen, F., & Huang, Y. (2013, October). A parallel computing platform for training large scale neural networks. In *2013 IEEE International Conference on big data* (pp. 376-384). IEEE.

12) Sharif, M. H., & Gursoy, O. (2018). Parallel computing for artificial neural network training using java native socket programming. *Periodicals of engineering and natural sciences*, *6*(1), 1-10.