Parallelized Backpropagation Neural Network Algorithm using Distributed System

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Computer Science and Engineering

Thapar University, Patiala

By

Kiran kumar kaki
(80632009)

Under the supervision of:
Dr. V. P. Singh
CSED

January 2009

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Parallelized Backpropagation Neural Network Algorithm using Distributed System"**, submitted by me in partial fulfillment of the requirements for the award of degree of Master of Engineering in Computer Science and Engineering in the Department of computer science and engineering of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. V. P. Singh and refers other researcher's works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

(**kiran kumar .kaki***)*

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(**Dr.V.P.Singh)**
Computer Science and Engineering Department
Thapar University
Patiala

Countersigned by

(Dr. SEEMA BAWA)
Professor & Head
Computer Science & Engineering Department
Thapar University
Patiala

(Dr. R.K.SHARMA)
Dean (Academic Affairs)
Thapar University,
Patiala.

Acknowledgment

I express my sincere and deep gratitude to my guide Dr.V.P.Singh, Department of Computer Science and Engineering, for the invaluable guidance, support and encouragement. He provided me all resource and guidance throughout of the thesis work.

I am thankful to Dr.Seema Bawa, Head of Department of Computer Science and Engineering, Thapar University, Patiala, for providing me adequate environment, and facility for carrying thesis work.

I would like to like to say thanks from deep inside my heart to my friends who had helped me at every moment of my study and is constantly guiding me and strengthening me. They are with me all the time.

Kiran kumar.k

(80632009)

ABSTRACT

With rapid increase of computer system performance during past decade, parallel processing becomes a crucial issue for computer system. Different parallel computing based methods have been proposed in recent years for the development of system performance. In the present research work, back propagation neural network training algorithm has been parallelized using distributed environment. The neural network model has been trained and tested for the vibration analysis data. Parallel processing using back propagation neural network algorithm helps to increase the performance of the system and to decrease the convergence time for the training of the neural network.

In this work a three-layer architecture of back propagation neural network is considered in distributed environment for the plate vibration problem. The input layer consists of two inputs, specified boundary condition and aspect ratio (m) of the elliptic plate. The output layer of the artificial neural network architecture consists of one output in the form of the corresponding frequency parameter (f). However, the number of nodes in the hidden layer has been taken as 10 and 15 for comparison the results.

The parallelism of back propagation neural network has been trained and tested for the analysis of vibration data. The implementation of parallelism of back propagation neural network algorithm on a distributed computing system with good performance has been demonstrated. The convergence time for the training of back propagation neural network by parallel processing is faster as compared to the single processing. The application of the proposed research work gives a faster estimation for the analysis of vibration data. In this research work, parallel processing of back propagation neural network has achieved the performance of the system with desired accuracy of the results.

Keywords: parallel processing, Backporpagation, Artificial Neural Networks, Distributed System.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Introduction

In some practical applications of neural networks, fast response 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 application 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. Combining the strengths of parallel processing and the use of distributed computing, the neural network is able to enhance the processing times for both the learning and execution stages so as to efficiently calculate the most probable output with a remarkable degree of accuracy. As such, the objective of this project is to demonstrate the enhanced computational speed of a generalized large- scaled neural network with broad-based applications using parallel computing. 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 its specific applications. The characteristics which determines the efficiency of this program are

- Speed
- Accuracy
- Real-Time capability (time)

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 increase with the times, the reality of achieving real-time computations of extremely large problems become almost a possibility. 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 such as this.

1.2 Motivation and Aim

Neural networks are specified with many small processors working simultaneously on the same task. It has the ability to ’learn’ from training data and use its’knowledge’ to compare patterns in a data set. The aim of this research is to design, implement, and evaluate an distributed computing systems and to compare performance of different neural network training functions which one works better for good system performance. This project describes an artificial neural network based system for distributed computing system. The system will take input data and run on different systems parallely.

1.3 Disposition

Chapter 2 Introduces the state art of artificial neural network and the literature based on to implement distributed environment

Chapter 3 Introduces the state art of parallel computing

Chapter 4 Describes approach to solve the present problem and importance of neural network to solve the present problem

Chapter 5 Provides the details of the training algorithm that has been used, architecture of the parallel neural network, and implementation of distributed environment system.

Chapter 6 Discusses about the results of training and testing in the case of single processing and parallel processing of the data

Chapter 7 Conclusion and further enhancement of work

CHAPTER 2 Artificial Neural Networks

INTORDUCTION

This chapter describes Artificial Neural Networks in detail, it explain how artificial neural network will help to implement in parallel processing.

2.1 Biological Neuron

The elementary nerve cell, called a neuron, is the fundamental building block of the biological neural network. Its schemantic diagram is shown in figure 2.1. Typical cell has three major regions: the cell body, which is also called the soma, the axon, and the dendrites, dendrites form a dendrites tree, which is a very fine bush of thin fibers around the neuron's body. Dendrites receive information from neurons through axons-long fibers that serve as transmission lines. An axon is a long cylindrical connection that carries impulses from the neuron. The end part of an axon splits into a fine arborization. Each branch of it terminates in a small end bulb almost touching the dendrites of neighboring neurons. The axon-dendrite contact organ is called synapse. The synapse is where the neuron introduces its signal to the neighboring neuron. The signal reaching a synapse and received by dendrites are electrical impulses. The inter neuronal transmission is sometimes electrical but is usually effected by the release of chemical transmitters at the synapse. Thus, terminal buttons generate the chemical that affects the receiving neuron. The receiving neuron either generates an impulse to its axon, or produces no response. The neuron is able to respond to the total of its inputs aggregated within a short time interval called the period of latent summation. The neuron’s response is generated if the total potential of its membrane reaches a certain level. The membrane can be considered as a shell, which aggregates the magnitude of the incoming signals over some duration. Specifically, the neuron generates a pulse response and sends it to its axon only if the conditions necessary for firing are fulfilled

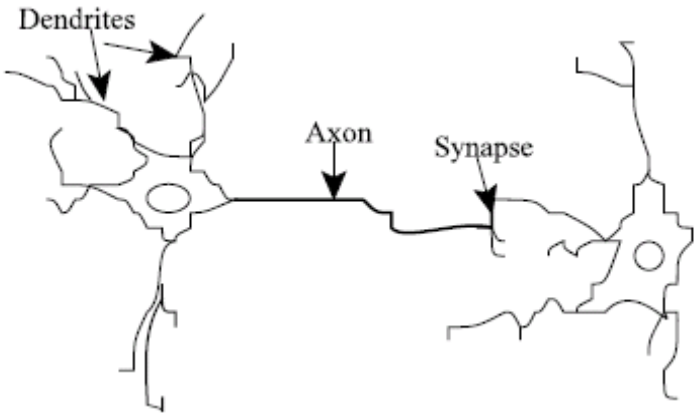


Figure 2.1: Basic Neuron

The necessary conditions for the firing of a neuron, incoming impulses can be excitatory if they cause the firing, or inhibitory if they hinder the firing of the response. A more precise condition for firing is that the excitation should exceed the inhibition by the amount called the threshold of the neuron, typically a value of about 40V. Since a synaptic connection cause the excitatory or inhibitory reactions of the receiving neuron, it is practical to assign positive and negative unity weight values, respectively, to such connections. This allows to reformulating the neuron's firing condition. The neuron fires when the total of the weights to receive impulses exceeds the threshold value during the latent summation period.

The incoming impulse to a neuron can only be generated by neighboring neurons and by the neuron itself. Usually, a certain number of incoming impulses are required to make a target cell fire. Impulses that are closely spaced in time and arrive synchronously are more likely to cause the neuron to fire. As mentioned before, observations have been made that biological networks perform temporal integration and summation of incoming signals. The resulting spatio-temporal processing performed by natural neural networks is a complex process and much less structured than digital computation. The neural impulses are not synchronized in time as opposed to the synchronous discipline of digital computation. The characteristic feature of the biological neuron is that the signals generated do not differ significantly in magnitude; the signal in the nerve fiber is either absent or has the maximum value. In other words, information is transmitted between the nerve cells by means of binary signals.

After carrying a pulse, an axon fiber is in a state of complete non-excitability for a certain time called the refractory period. For this time interval the nerve does not conduct any signals, regardless of the intensity of excitation. Thus, we may divide the time scale into consecutive interval, each equal to the length of the refractory period. This will enable a discrete-time description of the neurons performance in terms of their states at discrete time instance. Excitement neuron can be described by taking with example for the neurons that will fire at the instant k+1 based on the excitation conditions at the instant k. the neuron will be excited at the present instant if the number of excited excitatory synapses exceeds the number of excited inhibitory synapses at the previous instant by at least the number T, where T is the neuron's threshold value.

2.2 Artificial Neural Network

Artificial neural network (ANN) is a discipline that draws its inspiration from the incredible problem-solving abilities of nature's computing engine, the human brain, and endeavors to translate these abilities into computing machine which can then be used to tackle difficult problems in science and engineering. However, all Artificial neural network paradigms involve a learning phase in which the neural network is trained with a set of examples of a problem. For practical problems where the training data is large, training times of the order of days and weeks are not uncommon on serial machines. This has been the main stumbling block for artificial neural network use in real-world applications and has also greatly impeded its wider acceptability.

The problem of large training time can be overcome either by devising faster learning algorithms or by implementing the existing algorithms on parallel computing architectures.

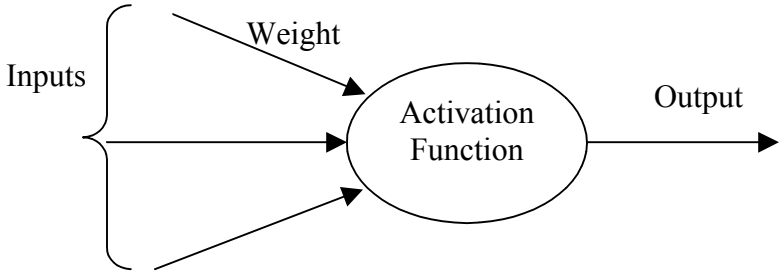


Figure 2.2: Artificial Neural Network

2.2.1 Single Layer Perceptron

A single layer perceptron (SLP) is the simplest form of artificial neural network that can be built. It consists of one or more artificial neurons in parallel. Each neuron in the single layer provides one network output, and is usually connected to all of the external inputs. The diagram shown in figure 2.3 illustrates a very simple neural network; it consists of a single neuron in the output layer. There are n neurons in the input layer; each circle represents a neuron. The total input stimuli to the neuron in the output layer is

$$z_{in} = \sum_{i=0}^{n} x_i w_i = x_0 w_0 + x_1 w_1 + x_2 w_2 + \ldots x_n w_n$$

Output of the neuron = f (z_{in}), the input x is a special input, referred to as the bias input. Its value is normally fixed at 1. Its associated weight w is referred to as the bias weight. This approach to building a single layer perceptron encourages a greater understanding of the concepts relating to neural networks. The single layer perceptron, implements a form of supervised learning. Supervised neural networks are trained to produce desired outputs when specific inputs are used in the system. Supervised neural networks are particularly well suited for modeling and controlling dynamic systems, classifying noisy data, and predicting future events. In this case, building without the toolbox creates a less powerful but functioning SLP. When designing the SLP structure the weights are assigned small random values, input and target output patterns are also applied. The output of the perceptron is calculated from the equation

$$y(k) = f(\sum W_k * X_k)$$

Y output, w weights, x inputs

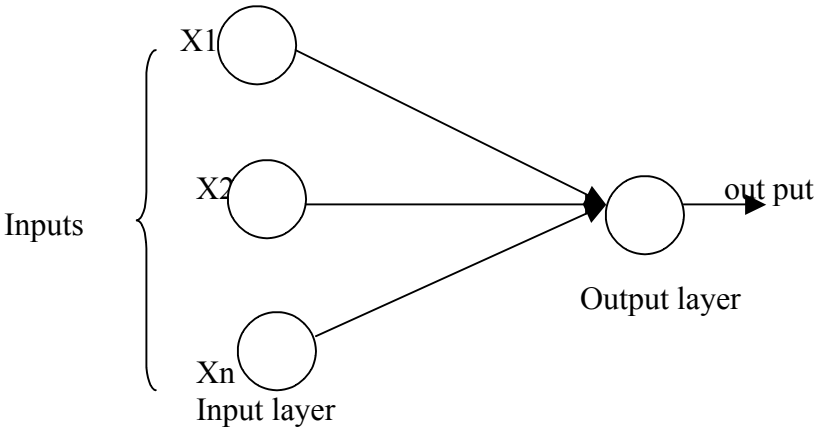


Figure 2.3: Single Layer Neural Network

2.2.2 Multi Layer Perceptron

This class of networks consists of multiple layers of computational units, usually interconnected in a feed-forward way. Each neuron in one layer has directed connections to the neurons of the subsequent layer as shown in figure 2.4. In many applications the units of these networks apply a sigmoid function as an activation function. The "universal approximation theorem" for neural networks states that every continuous function that maps intervals of real numbers to some output interval of real numbers can be approximated arbitrarily closely by a multi-layer perceptron with just one hidden layer. This result holds only for restricted classes of activation functions, e.g. for the sigmoid functions. Multi-layer networks use a variety of learning techniques, the most popular being "back-propagation". Here, the output values are compared with the correct answer to compute the value of some predefined error-function. By various techniques, the error is then fed back through the network. Using this information, the algorithm adjusts the weights of each connection in order to reduce the value of the error function by some small amount. After repeating this process for a sufficiently large number of training cycles, the network will usually converge to some state where the error of the calculations is small. In this case, one would say that the network has "learned" a certain target function. To adjust weights properly, one applies a general method for non-linear Optimization (mathematics) that is called gradient descent. For this, the derivative of the error function with respect to the network weights is calculated, and the weights are then

changed such that the error decreases (thus going downhill on the surface of the error function). For this reason, back-propagation can only be applied on networks with differentiable activation functions. In general, the problem of teaching a network to perform well, even on samples that were not used as training samples, is a quite subtle issue that requires additional techniques. This is especially important for cases where only very limited numbers of training samples are available. The training data and fails to capture the true statistical process generating the data. Computational learning theory is concerned with training classifiers on a limited amount of data. In the context of neural networks a simple heuristic, called early stopping, often ensures that the network will generalize well to examples not in the training set. Other typical problems of the back-propagation algorithm are the speed of convergence and the possibility of ending up in a local minimum of the error function.

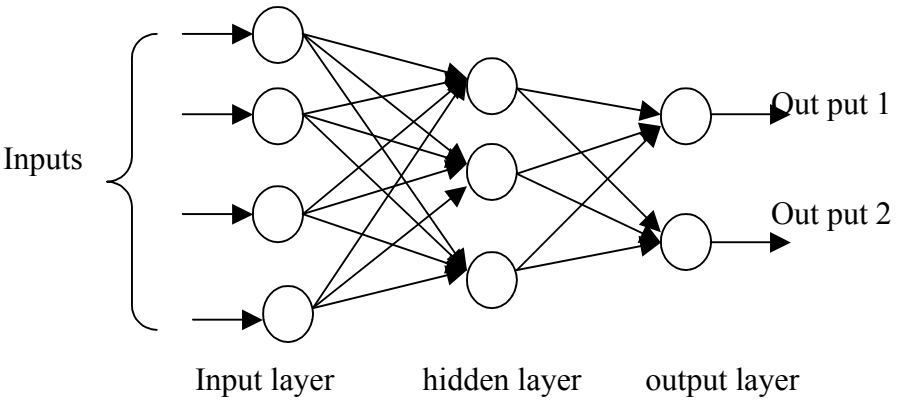


Figure2.4: Multi Layer Neural Network

2.3 Artificial Neuron and Activation Function

The artificial neuron like the biological neuron described in figures2.3 is a processing element. An output for this artificial neuron is calculated by multiplying its inputs by a weight vector. The results are then added together and an activation function is applied to the sum. The activation function is a function used to transform the activation level of a unit or rather a neuron into an output signal. Typically, activation functions have a “squashing” effect; they contain the output within a range.

2.3.1 Linear Activation Function

There are many activation functions that can be applied to neural networks; three main activation functions are dealt with in this project the first is the linear transform function, or purelin function. It is defined as follows

$$f(x) = X$$

Neurons of this type are used as linear approximators

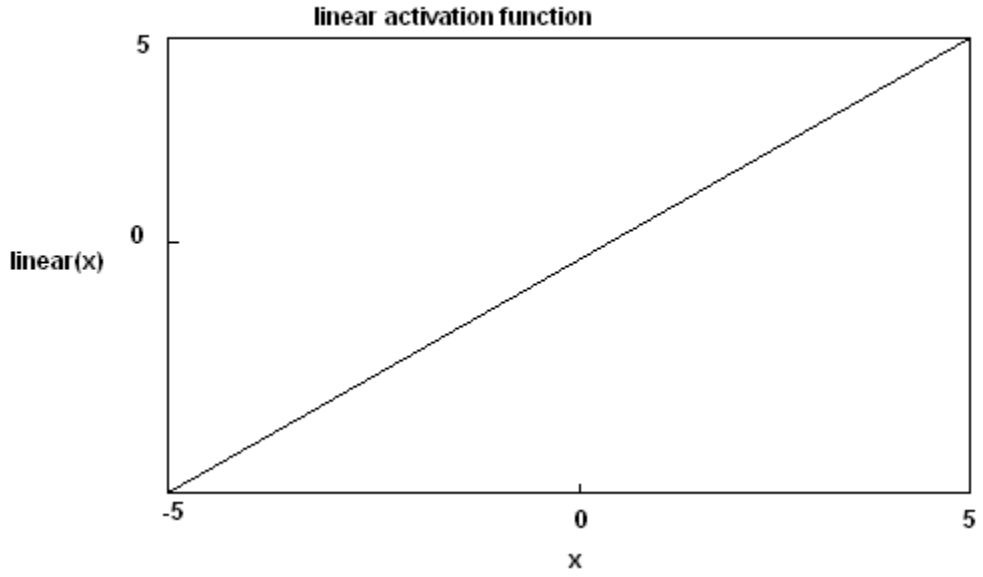


Figure 2.5: Linear Activation Function

2.3.2 Non Linear Activation Function

There are several types of non-linear activation functions; the two most common are the log-sigmoid transfer function and the tan-sigmoid transfer function. Plots of these differentiable, non-linear activation functions are illustrated in figure 2.6. They are commonly used in networks trained with back propagation. The networks referred to in this work are generally back propagation models and they mainly use log-sig and tan-sig activation functions. The logistic activation function; it is defined by the equation

$$\text{Logsig}(x) = 1/(1+\exp^{(-\sigma x)})$$

$\sigma = 1$ though it can be changed which in turn changes the shape of the sigmoid. As σ tends toward infinity it behaves more and more like a hard-limiter where the slope of the sigmoid is zero. In this case where the slope is not zero, the output range is contained between 0 and 1

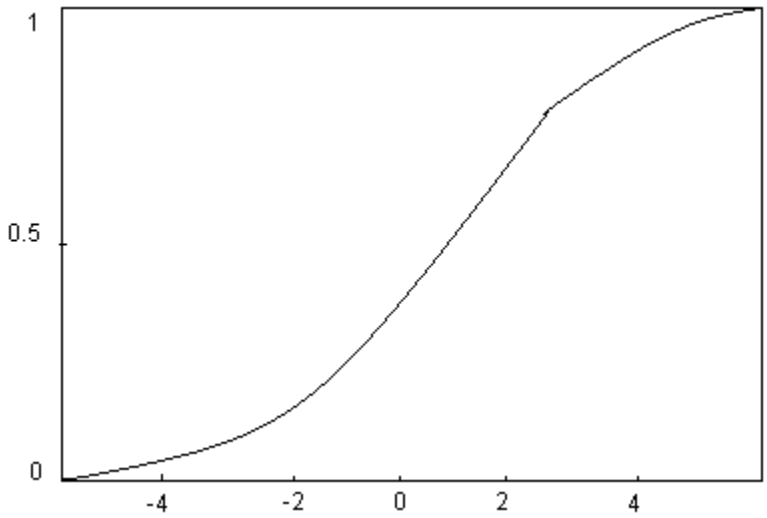


Figure 2.6: Non Linear Activation Function

2.4 Learning Methods

In general, learning is a relatively permanent change in behavior brought about by experience. Learning in neural networks is a more direct process, and we typically can capture each learning step in a distinct cause-effect relationship. The knowledge of a neural network is stored in the synapses, which are the weights of the connections between the neurons. These weights between two layers of neuron can be represented as matrices. If the neural network with the proper learning algorithm, which is determined by the analysis and preprocessing of the data, can produce a reasonable prediction on how much it is going to rain the next year. Another field, that is quite more interesting and way more challenging, is the stock market. Undoubtedly, investors will be interested in knowing how the value of a stock is going to develop in the short and long term. Based on the former developments of the stock for the passed years, a neural network model can be trained to predict when a stock will yield the largest profit. But the stock market is a very complex field and it is doubtful that such a model will be invented.

The definition of the learning process implies the following sequence of events:

1. The neural network is stimulated by an environment.
2. The neural network undergoes changes in its free parameters as a result of the stimulation.

3. The network responds in a new way to the environment due to the changes that occurred in its internal structure.

There are numerous algorithms available and as one would expect there is no unique algorithm for designing a neural network model. The difference between the algorithms lies in formulation of how to alter the weights of the neurons and in the relations of the neurons to their environment.

All learning methods can be classified into two major categories.

2.4.1 Supervised Learning

In this learning method an external teacher is incorporated, so that each output unit is told what its desired response to input signals ought to be. During the learning process global information may be required. Paradigms of supervised learning include error-correction learning, reinforcement learning and stochastic learning. Important issue concerning supervised learning is the problem of error convergence, i.e. the minimization of error between the desired and computed unit values. The aim is to determine a set of weights, which minimizes the error. One well-known method, which is common to many learning paradigms, is the least mean square (LMS) convergence.

2.4.2 Unsupervised Learning

Unsupervised learning uses no external teacher and is based upon only local information. It is also referred to as self-organization, in the sense that it self-organizes data presented to the network and detects their emergent collective properties. Hebbian learning and competitive learning are the paradigms of unsupervised learning. We say that a neural network learns off-line if the learning phase and the operation phase are distinct. A neural network learns on-line if it learns and operates at the same time. Usually, supervised learning is performed off-line, whereas unsupervised learning is performed on-line.

2.5 Parallel Processing for Simulating Ann's

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. Simulation of an ANN comprises 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 [1]. 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. Neurocomputers are designed as boards and system for high-speed ANN simulations [25]. Neurocomputers can be classified as general purpose of special purpose [22]. Most neurocomputers are based on processing elements computing in parallel.

2.6 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. Feed –forward neural network with a single hidden layer is assumed unless otherwise stated.

2.6.1 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 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.

2.6.2 Image Processing

Satellite images Remote sensing is of interest for agricultural, environmental, and weather-control applications. For some of these it is necessary to identify human-mad structures like roads of residential areas,

2.6.3 Face Recognition

High order neural networks have been used for human face recognition [17]. The network consists of a preprocessing network followed by a feed-forward network with one or two hidden layers. Four image planes, each of 16 X 16 elements are input to the network. The originally proposed network had to be reduced to make computer simulations possible on a i486 personal computer: the recognition rate varied according to the transforming (scaling and rotation) of the input image. In the best case, over 96 percent of the transformed training patterns were classified correctly.

2.6.4 Medical Imaging

Much research is published within the field of medical imaging [18] molecular biology is one area in which neural networks often outpace traditional tried methods. However, in most other areas, neural networks have not yet been shown to outperform statistical techniques. A feed-forward neural network used for cancer cell classification. A classification accuracy of as high as 96.8 percent was obtained. The method is based on using extracted features from the raw image as input to the network.

2.6.5 Road and Obstacle Recognition

Tsinas described a vision system for real-time recognition of traffic situations [19]. Separate Feed Forward Neural Networks are used for recognizing the driving lane and obstacles on the lane, respectively.

2.6.6 Image Compression

Multi layer neural network can be used to transform image data into compressed data by reducing the spatial redundancy [20]. The number of neurons in the hidden layer is chosen to be smaller than in the input and output layer. Thus, the output of the hidden layer is a compressed image, and the output layer decompresses the image. In order to increase the generalization properties, the image is dived into blocks. By dividing the original image into 8 X 8 pixel blocks and using an adaptive compression ratio according to the complexity of each block

2.6.7 Process Control

Neural networks are ideal for process control because they can build predictive models of the process directly from multidimensional data collected from sensors [21]. The network Ids need history, which is often abundant, which is often absent. Thus neural networks are will-suited for predicting, controlling, and optimizing industrial processes.

CHAPTER 3 Parallel Artificial Neural Network

INTRODUCTION

This chapter describes overview of parallel architectures, need of parallel processing, parallel artificial neural network.

3.1 General Aspects of Parallel Processing

A parallel computer usually consists of a number of processing elements (PEs). Each processing element consists of a processor and memory. The memory can either be on the processing chip or on separate chips. Neural circuits have been produced containing several processing elements on a single chip. Because the processing elements may have to exchange data with their neighbors, a communication module may be required for each PE. A number of topologies exist for interconnecting processing elements [6]. The most common are shown in the figure 3.1. For example finer grained parallel processing can be realized by using a multidimensional topology. Parallel computers can be classified according to following classification, based on the number of simultaneous instruction and data streams [6].

SISD (single instruction stream single data stream): - A sequential computer with a single processor.

SIMD (single instruction stream multiple data streams): - A single program controls multiple execution units.

MISD (multiple instruction streams single data stream): - Systolic arrays with pipelined execution.

MIMD (multiple instruction streams multiple data streams): - Computers with more than one processor and the ability to execute more than one program simultaneously. Computers in this category are also called multiprocessors or multi computers depending on shared or distributed memory, respectively.

SPMD (Single programs operating on multiple data streams): - The same program is downloaded onto all the processing elements. The processors are usually performing the same operations but on different parts of the data [5].

Several different methods are used for interprocessor communication. The two major switching methods for communication [6] are the following.

Circuit switching: - A physical path is established between the source and the destination before the message is sent. SIMD machines frequently use circuit switching.

Packet switching: - A message is split into fixed or flexible-sized packets. Each packet is routed through the interconnection network independent of other packets. As such, packets may take different routes through the network. MIMD machines are usually based on packet switching.

Two major concerns in parallel implementations are the following:

Load balancing: - To minimize idle time, it is necessary to keep the processors active. Each processor should be given an equivalent computation load.

Communication: - To maximize the time processors perform computation, communication should be minimized. Moreover, the communication should be distributed as evenly as possible over all the communications links [23].

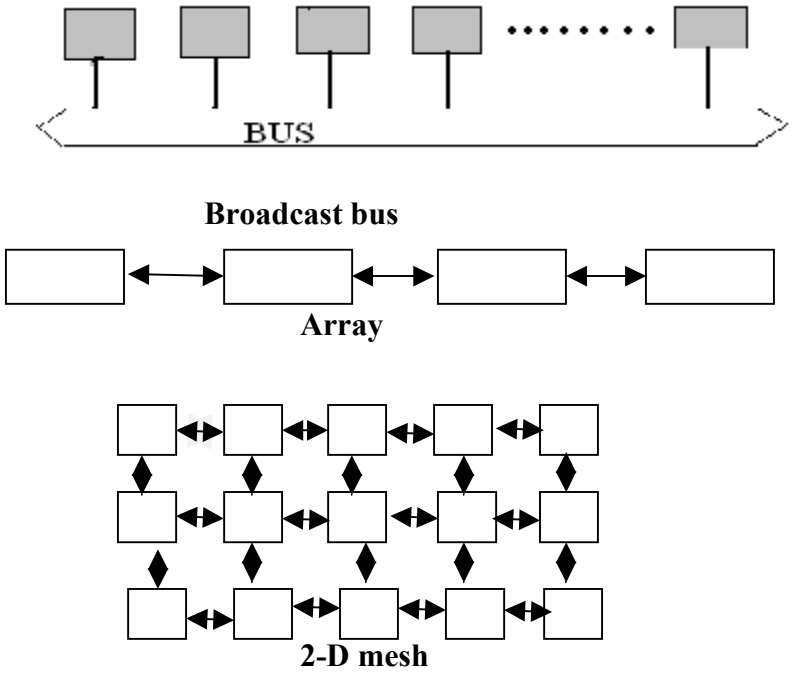


Figure 3.1: Processor Topologies for Simulating Ann's

3.2 Parallel Implementation of Back Propagation Neural Network

3.2.1 Introduction

Feed forward neural network with back propagation learning is the most widely used configuration. Because the back propagation learning for a bigger network with a large training set takes a long time (in term of days), it becomes imperative to look at parallel implementation schemes to reduce this large training time. Because the implementations depend very much on the type of parallelism used.

3.2.2 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 a number of training patterns, each given by an input vector and the corresponding out put 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 network in this study, not for processor topology network.

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 (that is, one training iteration).

Weight update Interval: -

The number of training patterns that is presented between weight updates is termed μ. For LBP, μ=1, where as for LBE μ=p, where p is the number of training patterns in the training set.

The back propagation 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 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 phase 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.

3.3 Distributed Computing for Back Propagation Parallelism

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

3.3.1 Training Set Parallelism

Training set parallelism is also called data parallelism because the training set is partitioned, not the training program. An example is given in figure 3.2 for training of the English alphabet. Each processing element (PE) has a local copy of the complete weight matrices and accumulate 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.

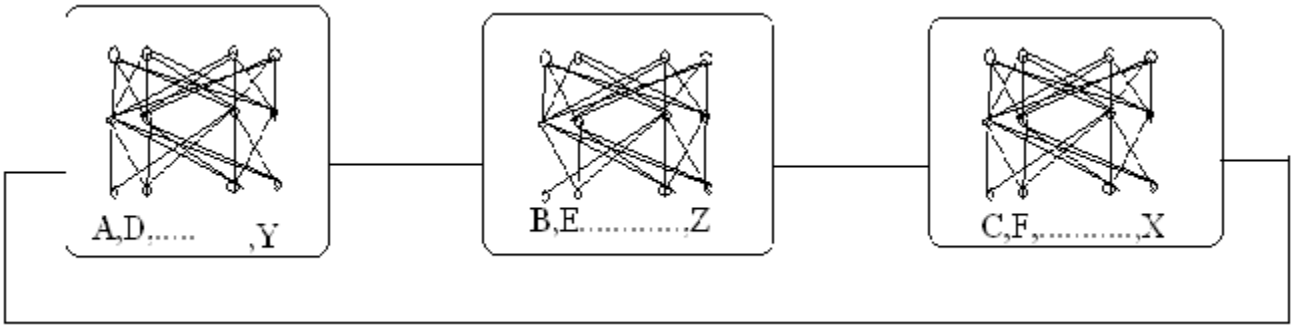


Figure 3.2: For Training of The English Alphabet

3.3.2 Pipelining

In pipelining, the different weight layers are computed in different PEs as illustrated in figure 3.3 shows a pipelining example. First the hidden layer processor computes output values of training pattern. The output processor reads the values and computes output and error values of (a). The hidden processor concurrently processes the next training pattern (b). Then, it reads the hidden error for (a) and both processors accumulate the weight change values for (a). This method interleaves the forward phase with the backward phase. The extension is to execute the two training phase in parallel [10].

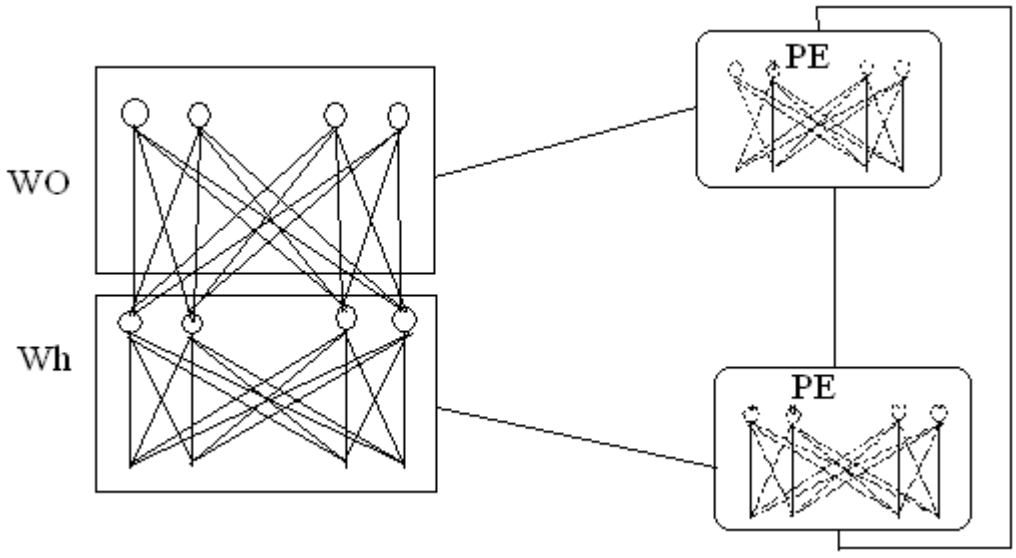


Figure 3.3: Pipelining

3.3.3 Node Parallelism

Node parallelism contains two sub degrees of network parallelism: neuron parallelism and synapse parallelism. The parallel mapping schemes, which use a mixture of these two, are named node parallel methods. Next, each of the two sub degrees are explained.

3.3.3.1 Neuron Parallelism

The most common way to parallelize a feed-forward network is by using neuron parallelism, also called vertical slicing. The example in figure 3.4 applies the principle to three processing elements. All incoming weights to one hidden and one output neuron are mapped to each PE. That is, each PE stores all the incoming weights to the neuron assigned to that PE. The network slicing corresponds to storing one row of the weight matrix in each PE. The output of the neurons is computed by the matrix-vector product;

$$L= \{layer \mid layer \{h, o\}\}.$$

First, the value of one hidden neuron is computed in each PE. Then, each PE exchanges the values, for example, over a ring bus, and continues by computing the value of the output neuron. The hidden layer error is computed based on the output error. This can be formulated as a vector matrix product because the weights are stored in row order, the sum of products is computed by adding partial product. However, a more effective summation could be achieved if the weights were stored in column order. Thus some implementations store the weights twice, both by row order for forward pass and column order for backward pass. Either duplicated weight updates of communication of the weights is by this method required. The former has been shown to be the most efficient [24].

3.3.3.2 Synapse Parallelism

Instead of mapping the rows of the weight matrices to each PE, the columns may be mapped. In synapse parallelism, each PE computes a partial sum of the neuron output as indicated in figure 3.4. The computation is more fine grained than for neuron parallelism. The sub results from each PE have to be added and broadcast to all PEs before the next layer can be computed. The advantage is that the hidden layer error can be computed without communication. Thus, some implementations use neuron parallelisms in the first layer add synapse parallelism in the second layer.

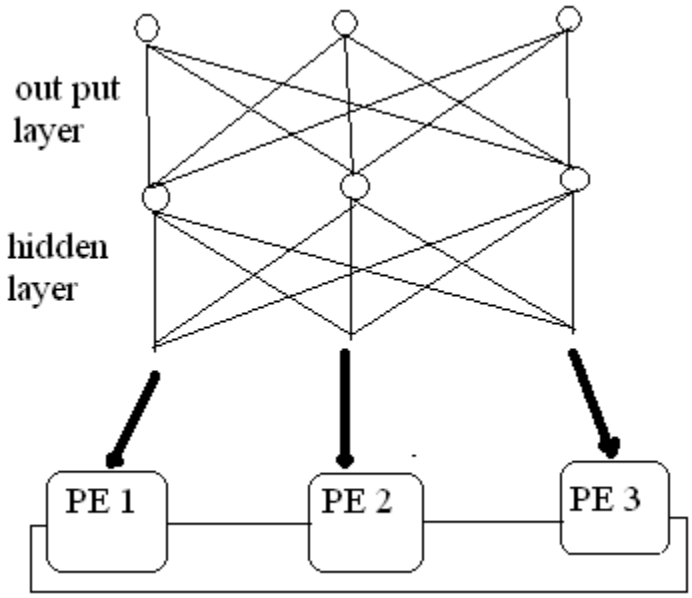


Figure 3.4: Synapse Parallelism

3.4 Need of Parallel Processing

The parallel computation with communication networks and distributed database systems will be one of the main supporting technologies for future digital systems. The tremendous rate of development in hardware/software technology means that any conclusions based on present capabilities will be outdated during the life cycle of any short term project. Therefore real computational bottlenecks need to be identified. For spatial digital systems, there are a number of aspects from which serious computational bottlenecks arise. For example

1. The enormous, and rapidly growing, quantity of raw digital datasets
2. The demand for increased resolution and size of images both for browsing and for scientific applications
3. The distributed nature of both data and potential users
4. The increase in the sophistication of queries.

The future of spatial digital libraries depends in part on the availability of parallel means to overcome such computational problems. Our aim is to develop the necessary

theory and computational tools to this end, as well as to validate our developments and algorithms on real-world data. When computations on very large data sets are involved, load balancing, routing, data partitioning, locality, and pre fetching become important implementation parameters that can drastically improve the system's response time. These issues exist in a distributed system of clients/servers, especially where one or more of the servers is itself a multiprocessor. However it is virtually impossible to create a meaningful application if a system designer is confronted with all the complexities of dynamic load balancing, data migration, and their resulting communication patterns. Functional designers need to be shielded from the complexities of the underlying architecture to the greatest extent possible, at the same time being able to develop applications that get reasonable performance

3.5 Survey of Different Parallel Implementations

Many mappings of back propagation onto parallel computers have been proposed and implemented, for both general purpose and special purpose computers. Mapping neural network models onto parallel architectures may be categorized into two general groups: heuristic mapping and algorithmic mapping [3]. Most of the proposed mappings are of the heuristic kind. They are generated by trial and error based on knowledge about the algorithm and the target machine. The algorithmic technique relies on a systematic approach to the implementation. The work by fujimoto and colleagues [4] and Kumar and colleagues [7] belong to the algorithmic category moreover, some theoretical studies of parallel algorithms have also been done.

3.5.1 Data Driven Systems

Data driven computations is based on asynchronous parallel execution of computations, which may be represented by directed graphs. A processor instruction is executed based on the availability of its data operands. Q-x is a floating-point data-driven processor. A message-passing multiprocessor is able to be configured of up to 1024 processor. A 2-D torus communication network processing interconnects the processors and storage in the processor is carried out as packets flowing through elastic pipelines. A description of the mapping of BP onto the architecture is reported in [8].

3.5.2 Mesh Topology

A fine grain back propagation algorithm using node parallelism for mesh connected multiprocessor is proposed [9]. It combines neuron and synapse parallelism, that is, each cell computes a partial sum for a single neuron. The weight matrices are divided into rectangular blocks or sub matrices that are distributed among the cells. The method is similar to the checkerboard partitioning method [11], described in the section on CM-5 implementations. The forward phase of the parallel BP algorithm can be illustrated by using a simple, fully connected network of four input, four hidden, and four output neurons and a system of four cells.

3.6 Parallel Artificial Neural Network

The simultaneous use of more than one system to execute a program ideally, parallel processing makes a program run faster because there are more systems running it. In practice, it is often difficult to divide a program in such a way that separate systems can execute different portions without interfering with each other. It is possible to perform parallel processing by connecting the computers in a network. However, this type of parallel processing requires very sophisticated software called distributed processing software. Note that parallel processing differs from multitasking, in which a single system executes several programs at once. Parallel processing is also called parallel computing.

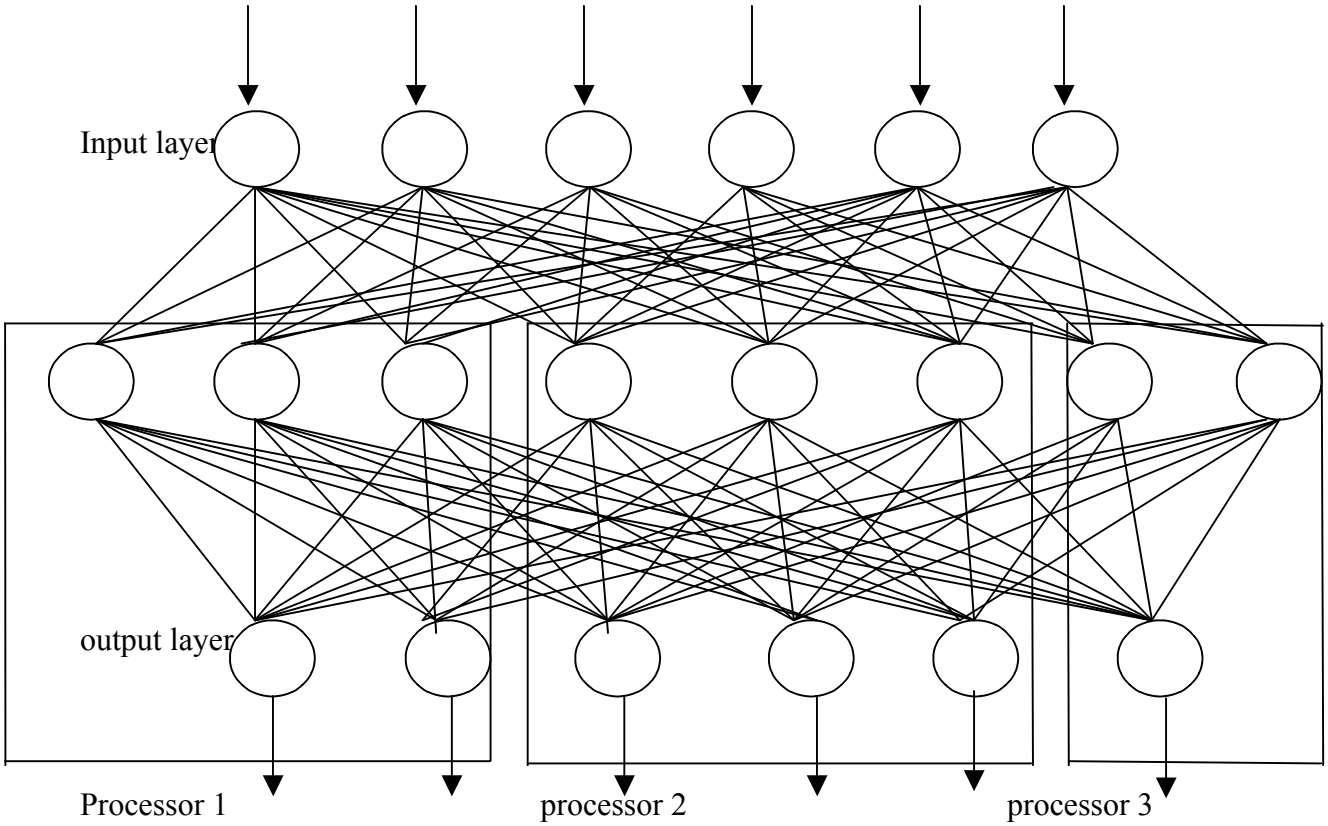


Figure 3.5: Network Partitioning for Array of Processors

A network based parallelism with vertical partitioning is used as the parallelizing method, each processor will have some neurons from each layer of the neural network. But because of heterogeneity, the number of neurons contained in each processor will in general be different from the other processor as shown in the figure 3.5. In this implementation, processors hold the incoming and outgoing weights of the hidden and output layer neurons that reside in them. Such a scheme is used in [12] for a homogeneous processor network to improve the speed of the forward and backward passes of back propagation. In the forward pass each processor computes the activations for the neurons contained in it. In order to do this it is necessary to have the activation values of all neurons from the previous layer. This means each processor has to transfer the activations (of its neurons) of each layer to all other processors once these have been computed. Because the processor network is heterogeneous, all processors will not complete their computations at the same time, and this causes processors to wait before

sending data to or receiving data from their neighbors. A similar reaction occurs while calculating the deltas during the backward pass of the back propagation algorithm

CHAPTER 4 Research Methodology

INTRODUCTION

This chapter presents in details of the research methodology, used for a back propagation neural network in the distributed environment.

4.1 Overview

The work Objective is to run the system in distributed environment to reduce the time and to improve the performance of the system. 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. Operation of the system is divided into five phases as shown in figure 4.1

- Input Data
- Data distribution among different processors
- Training
- Testing
- Output

Distributed computing is a science which solves a large problem by giving small parts of the problem to many computers to solve and then combining the solutions for the parts into a solution for the problem The simultaneous use of more than one system to execute a program Ideally, parallel processing makes a program run faster because there are more systems are running the program. In practice, it is often difficult to divide a program in such a way that separate systems can execute different portions without interfering with each other. It is possible to perform parallel processing by connecting the computers in a network. However, this type of parallel processing requires very sophisticated software called distributed processing software. Note that parallel processing differs from multitasking, in which a single system executes several programs at once. Parallel processing is also called parallel computing.

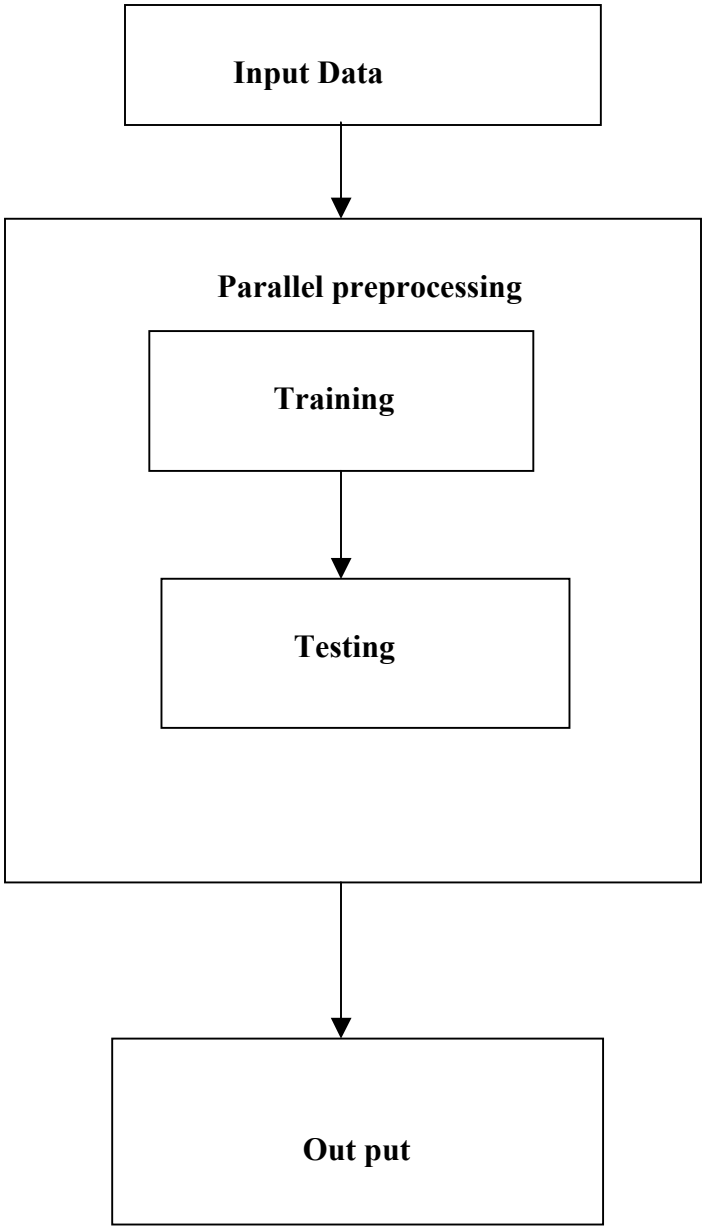


Figure 4.1: Block Diagram of System Overview

4.2 Why use Neural Networks?

A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. ,with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends, which are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. This expert can then be used to test with the new situations. Other advantages include:

- Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
- Self-Organization: An ANN can create its own organization or representation of the information it receives during learning time.
- Real Time Operation: ANN computations carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
- Parallel organization permits solutions to problems where multiple constraints must be satisfied simultaneously

4.3 Input Data

In this work a three-layer architecture of back propagation neural network is considered in 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 the results.

Table 4.1: Training patterns for Artificial Neural Network

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

4.4 Parallel Computing

Parallel computing is the simultaneous use of multiple compute resources to solve a computational problem. Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently (in parallel). There are several different forms of parallel computing bit level, instruction level and task parallelism. Distributed Computing environment enable us to coordinate and execute independent operations simultaneously on a cluster of computers, speeding up execution of job which contain large amount of data. A job is some large operation that need be to perform in distributed environment. A job is broken down into segments called tasks. The job divides into identical tasks, but tasks do not have to be identical, in distributed environment the job and its tasks are defined in client processor or user processor. The client uses Distributed environment to perform the definition of jobs 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 the figure 4.2.

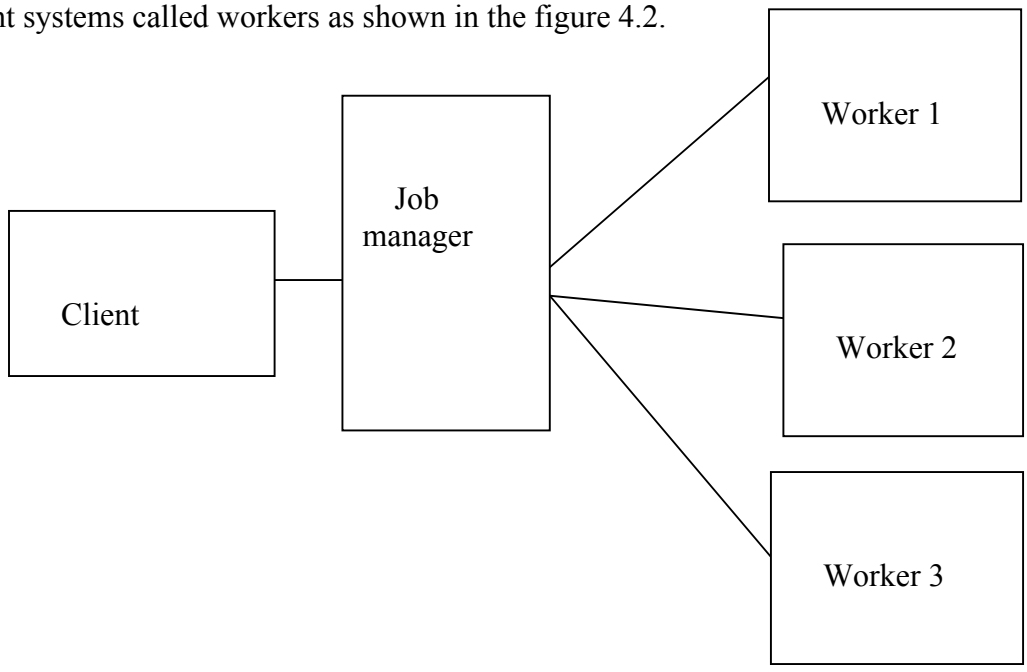


Figure 4.2: Basic Parallel Computing Configuration

4.5 Training

As stated in previous chapter back propagation neural network is used for behavior classification. The neural network that is going to construct for this work will be of two units in the input layer and one unit in the output layer. The network consists of only one hidden layer and can choose the number of hidden nodes in the hidden layer. In this work the network has been trained and tested by trainlm training function of neural network tool in mat lab environment.

4.6 Testing

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

4.7 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 4.2: Used Parameters for Neural Network

Parameters	Value
Number of inputs	2
Number of outputs	1
Number of hidden units	15
RMS-error	0.000001

CHAPTER 5 Implementation

INTRODUCTION

In this chapter the implementation of the parallel neural network has been discussed along with its architecture.

5.1 Multi layer Feed-Forward Networks with BP Learning

A three-layer feed-forward network is shown in figure 5.1, 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 back propagation involves four stages is as follows:

- Initialization of weights
- Feed forward
- Back propagation of errors
- Updation of the weights and biases.

There are two types of learning in back propagation: 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 network has been presented to the network. Thus the weights update is only performed after every epoch. To train the network, the proposed training algorithm used in the back propagation algorithm is given below.

The main steps are as follows:

1. Initialize the weights to small random values.

2. 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.

3. Calculate the actual outputs-this is the forward phase.

4. 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.

5. Repeat from step 2 for all training vectors.

6. Repeat from step 2 until the error is acceptably small.

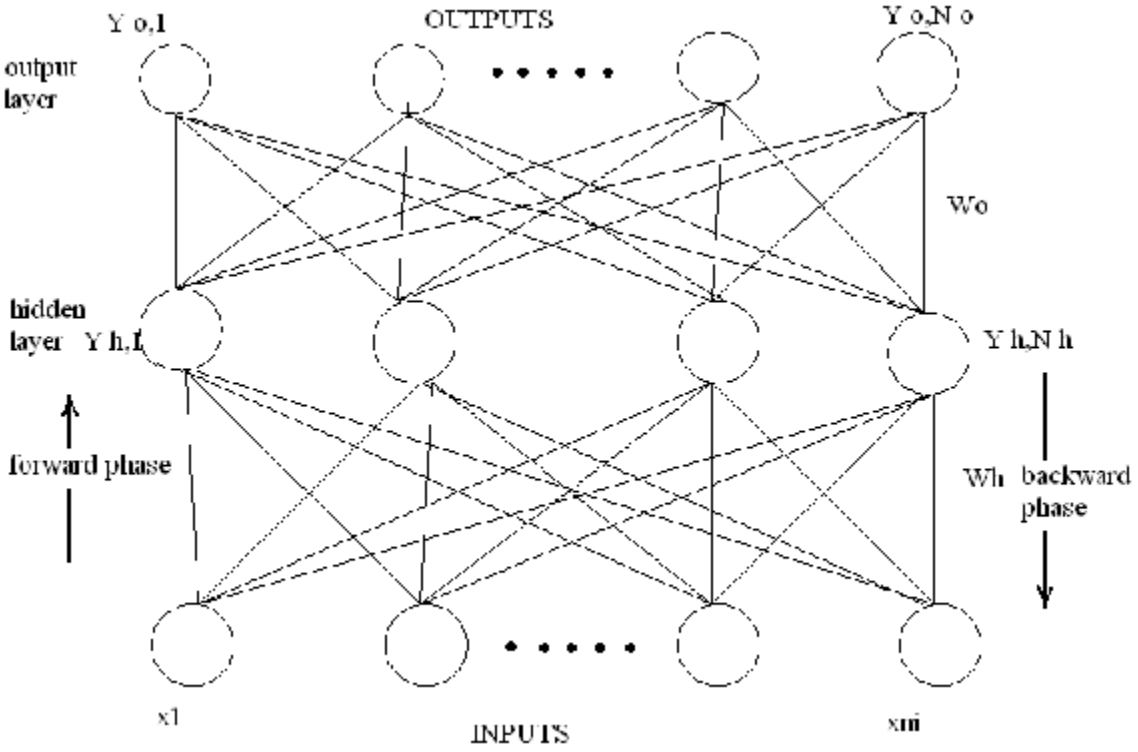


Figure 5.1: A Three Weight Layer Feed-Forward Neural Network

Back Propagation learning algorithm. 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

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

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 (α) =1/1+$e^{-\alpha}$ is used. It compresses the output value to lie in (0, 1), as shown in figure 5.2. Moreover the function is differentiable, which is a demand of the training algorithm.

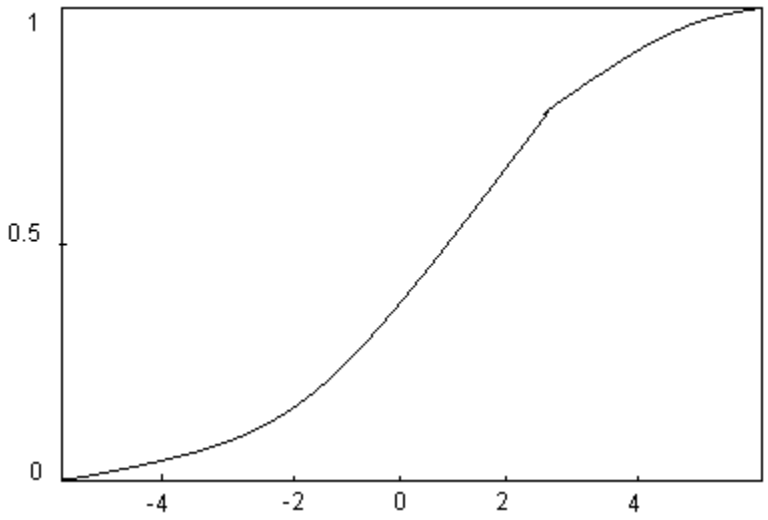


Figure 5.2: The Sigmoid Function f (α) =1/1+$e^{-\alpha}$

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

$$Y_{o,k} = f\left(\sum W_{o,kj} * Y_{h,j}\right)$$

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

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

The overall error measure for a training set of P patterns is

$$E = \sum E_p$$

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

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

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

$$\delta_{h,j} = Y_{h,j}(1 - Y_{h,j}) \sum \delta_{o,k} W_{o,kj}$$

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

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

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

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 Parallel Processing

Distributed Computing environment enable to coordinate and execute independent operations simultaneously on a cluster of computers, speeding up execution of job which contain large amount of data. A job is some large operations that need to be perform in distributed environment. A job is broken down into segments called tasks. Each task assigned to different worker. Every worker does their processing on task which they have been allocated. The main program runs on client system.

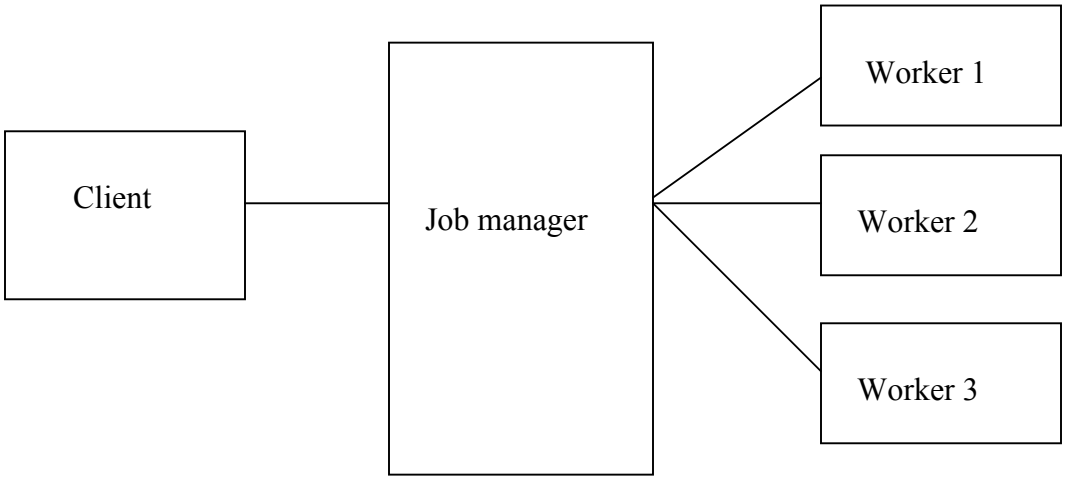


Figure 5.3: Block Diagram of Parallel Computing

5.3 Interactions of Distributed Computing Sessions

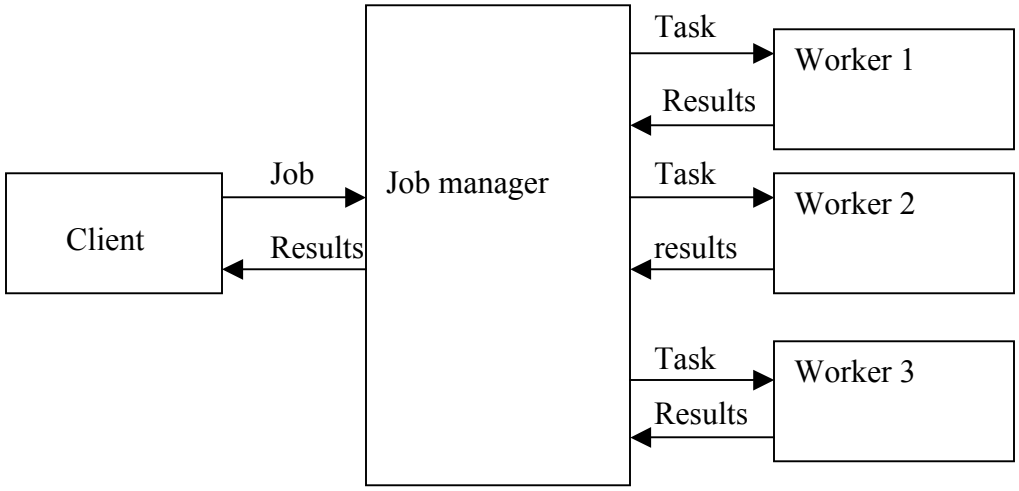


Figure 5.4: Interactions among Different Systems

A large network might include several job managers as well as several client sessions. Any client session can create, run, and access jobs on any job manager, but a worker session is registered with and dedicated to only one job manager at a time as shown in the figure 5.4, the optional job manager can run on any machine on the network. The job manager runs jobs in the order in which they are submitted, unless any jobs in its queue are promoted, demoted, canceled, or destroyed. Each worker receives a task of the running job from the job manager, executes the task, returns the result to the job manager, and then receives another task. When all tasks for a running job have been assigned to workers, the job manager starts running the next job with the next available worker. It is generally not important which worker executes a specific task. Each worker evaluates tasks one at a time, returning the results to the job manager. The job manager then returns the results of all the tasks in the job to the client session. Note that when a job is finished, it remains in the job manager or Data Location directory, even if we clear all the objects from the client session. The job manager or scheduler keeps all the jobs it has executed, until restart the job manager in a clean state. Therefore, information can be retrieved from a job at a later time or in another client session, so long as the job manager has not been restarted with the clean option. To permanently remove completed jobs from the job manager or scheduler's data location, we have to use the destroy function.

5.3.1 Job Manager Information

Job manager Information

==========

Type : job manager

ClusterOsType : pc

Data Location : database on hec107job@hecl07

- Assigned Jobs

Number Pending: 2

Number Queued : 0

Number Running : 1

Number Finished : 1

- Job manager Specific Properties

Name : hec107job

Hostname : hecl07

HostAddress(s) : 172.31.5.92

State : running

ClusterSize : 5

NumberOfIdleWorkers : 0

NumberOfBusyWorkers : 0

5.3.2 Stages of a Job

5.3.2.1 Pending: First a job is executed on the scheduler with the create Job function in the client session of Parallel Computing Toolbox software. When the job is assigned by tasks the job's first state is pending.

5.3.2.2 Queued: Execution time of the submit function on a job, the scheduler places the job in the queue, and the job's state is queued. The scheduler executes jobs in the queue in the sequence in which they are submitted, all jobs moving up the queue as the jobs before them are finished. The order of the jobs can be changed in the queue with the promote and demote functions.

5.3.2.3 Running: When a job reaches the top of the queue, the scheduler distributes the job's tasks to worker sessions for evaluation. The job's state is running. If more workers

are available than necessary for a job's tasks, the scheduler begins executing the next job. In this way, there can be more than one job running at a time.

5.3.2.4 Finished: When all of a job's tasks have been evaluated, a job is moved to the finished state. At this time, we can retrieve the results from all the tasks in the job with the function get All Output Arguments.

5.3.2.5 Failed: In the case of the third-party scheduler, a job might fail if the scheduler encounters an error when attempting to execute its commands or access necessary files.

5.3.2.6 Destroyed: When a job's data has been removed from its data location or from the job manager, the state of the job in the client is destroyed. This state is available only as long as the job object remains in the client.

The optional job manager can run on any machine on the network. The job manager runs jobs in the order in which they are submitted, unless any jobs in its queue are promoted, demoted, canceled, or destroyed. Each worker receives a task of the running job from the job manager, executes the task, returns the result to the job manager, and then receives another task. When all tasks for a running job have been assigned to workers, the job manager starts running the next job with the next available worker. It is generally not important which worker executes a specific task. Each worker evaluates tasks one at a time, returning the results to the job manager. The job manager then returns the results of all the tasks in the job to the client session.

5.4 Mat lab Configurations for Distributed Computing

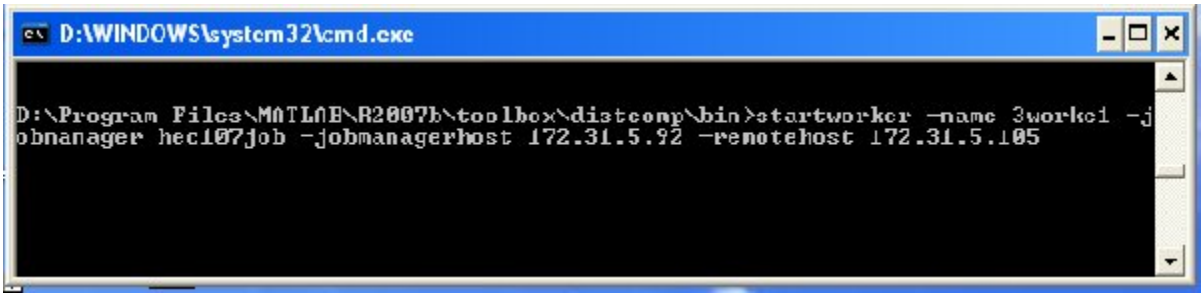
Generally, there is not much difficulty in deciding which machines will run worker processes and which will run client processes. Worker sessions usually run on the cluster of machines dedicated to that purpose. The client session of MATLAB usually runs where MATLAB programs are run, often on a user's 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 5.1: Distributed Computing Configuration

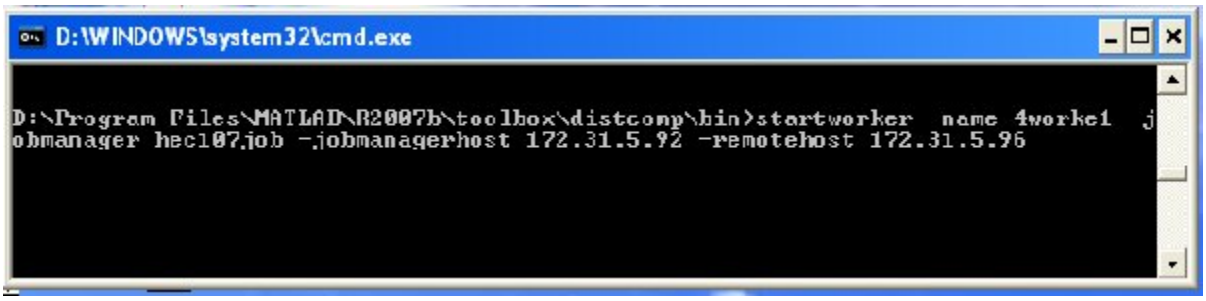
Session	Product	Processes
Client	Distributed Computing Toolbox	MATLAB with toolbox
Worker	MATLAB Distributed Computing Engine	worker; mdce service (if using a job manager)
Job manager	MATLAB Distributed Computing Engine	mdce service; job manager

5.4.1 Worker Configuration

The different workers running on different machines can be connected using the following configurations.
3worke1 running on machine with IP address 172.31.5.105



4worke1 running on machine with IP address 172.31.5.96



5.4.2 Job Manager Configuration

For the job Manager Configuration first Parallel pull-down menu has to be select on the MATLAB desktop. Click Parallel→ Manage Configurations to open the Configurations Manger as showing in the following figure 5.5.

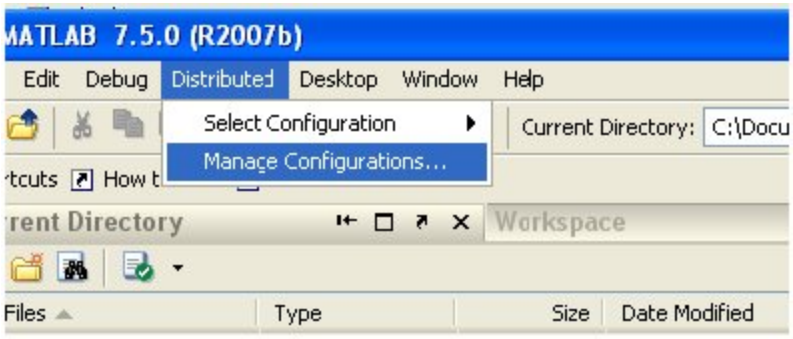


Figure 5.5: Mat lab Desktop

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.

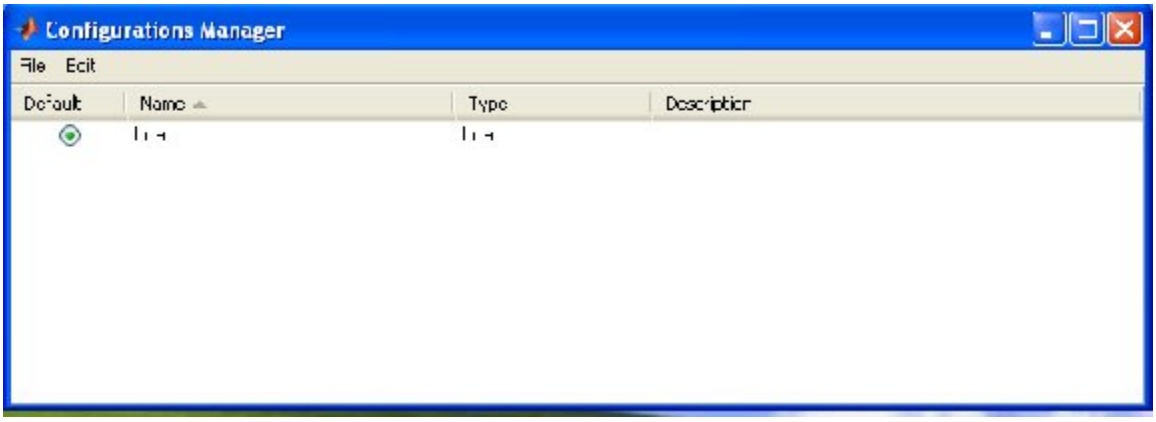


Figure 5.6: Configuration Manager 1

The following example provides instructions on how to create and modify configurations using the Configurations Manager and its menus and dialog boxes.

5.4.2.1 Creating and Modifying User Configurations

To create a configuration to set several properties for some jobs being run by a job manager.

1 In the Configurations Manager, click new → job manager. This specifies that we want a new configuration whose type of scheduler is a job manager.

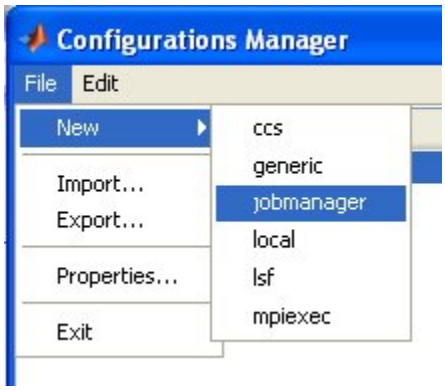


Figure 5.8: Configuration Manager 2

This opens a new Job Manager Configuration Properties dialog box.

2 Enter a configuration name MyJMconfig1 and a description as shown in the following figure 5.9. In the Scheduler tab, enter the host name 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 our network, enter the appropriate text.

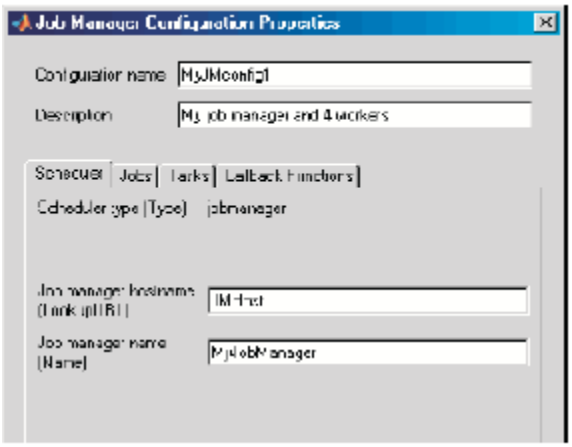


Figure 5.9: Job Manager Configuration Properties 1

3 In the Jobs tab, enter 4 and 4 for the maximum and minimum number of workers. This specifies that for jobs using this configuration, they 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.

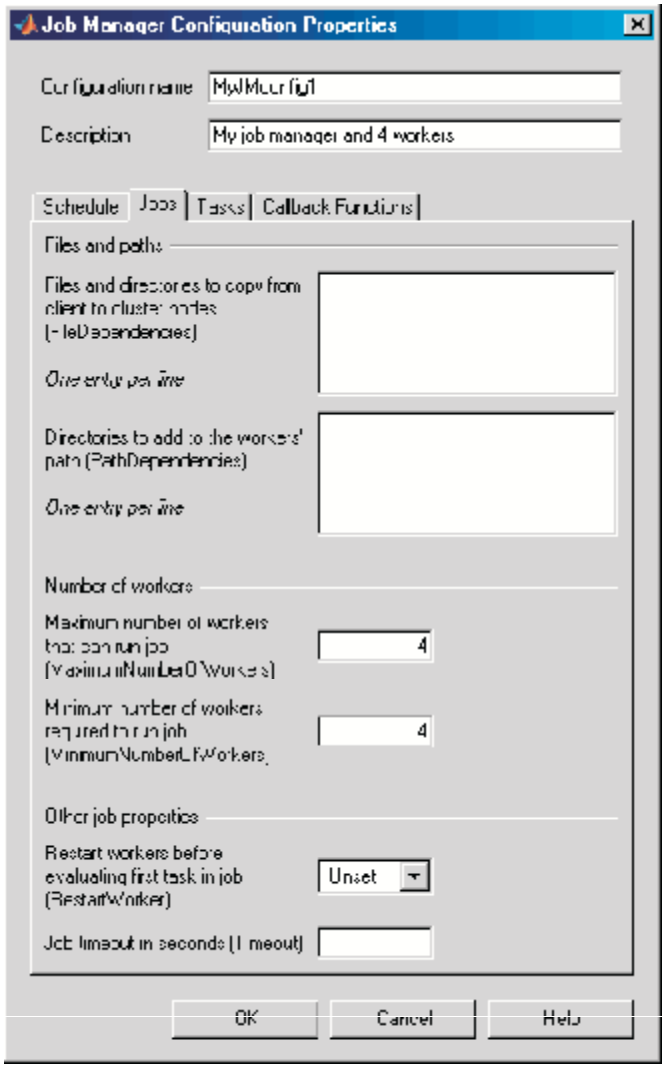


Figure 5.10: Job Manager Configuration Properties 2

5.4.2.2 New Configuration Manager

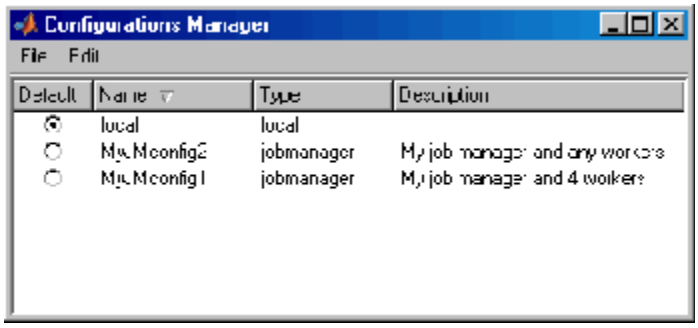


Figure 5.11: New Configuration Manager

After creating a job, apply either configuration to that job as a way of specifying how many workers it should run on.

5.5 Training

The training process requires a set of examples of proper network behavior - network inputs p and target outputs t. In previous chapters it has been already mentioned that the back propagation 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 option to choose 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.

5.6 Different Training Algorithms

Different test runs were made for each of the following training algorithm and also tried with different number of hidden nodes. The training function that has been used in this work is given in the table 5.2 with some other training functions.

Table 5.2: 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 memory reduction feature for use when the training set is large.

5.7 Testing

The system is already trained with the normal given data. In the training phase the input pattern and output pattern is given to the network but in this phase only the input pattern is given. After initialization of the weights, the input units of input layer are activated with the input patterns that have been taken from the input file. The outputs of the input layer are propagated towards the output layer similarly as training. The calculated output from the output unit of output layer is considered as the desired output pattern. This output will show us whether the given input pattern represents the desired output has come or not.

CHAPTER 6 Evaluation

INTRODUCTION

In This chapter experimental issues has been discussed, and Comparison between single and parallel data processing with trainlm training function.

6.1 Experiments

From this experiment it can be examined that the number of hidden layers that were needed to train the neural network and the trainlm training functions that has been used to train with the same dataset .by using trainlm training function, data processing is compared in the case of serial processing and parallel processing in this research work. In these experiments taking different data is used for plate vibration analysis. The following experiments has been performed to compare between single processing and parallel processing to increase the system performance in the case of parallel processing using distributed system in Mat lab environment.

6.2 Experiment 1

As preliminary experiment trainlm training function is used with 10 hidden units. The network has been trained till the RMS error value is reduced to an acceptable level. For training we used sessions that were collected for plate vibration analysis. The results of training for plate vibration analysis have been shown in table 6.1 with 10 and 15 hidden nodes. In the table number of hidden units denotes the number of neurons in the first layer. Number of epochs is nothing but number of iterations needed to converge the network with desired accuracy. Performance goal status represents that desired goal is achieved or not. The table 6.1 shows the comparison between single and parallel data processing for the training of the network.

6.2.1 Training

Table 6.1: Comparison of Training Pattrens between Single and Parallel system

	No of epochs/sec	hidden units	Performance goal
Single	8520	10	Achieved
Parallel	1326	10	Achieved
Single	2761	15	Achieved
Parallel	821	15	Achieved

6.2.1.1Single Processing

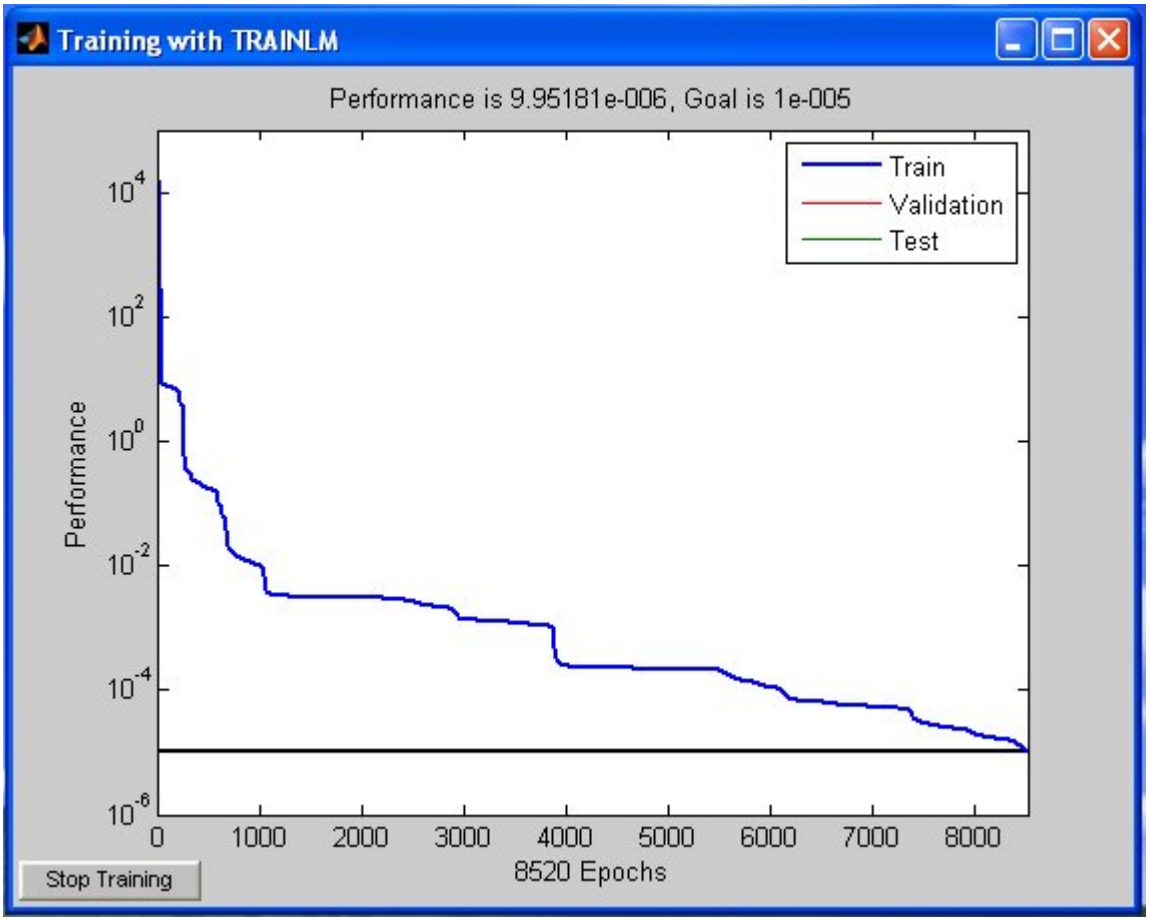


Figure 6.1: Output of Training Dataset-using Trainlm Training Function

The graph shown in figure 6.1 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. In this case the performance goal of the network has been achieved

6.2.1.2 Parallel Processing

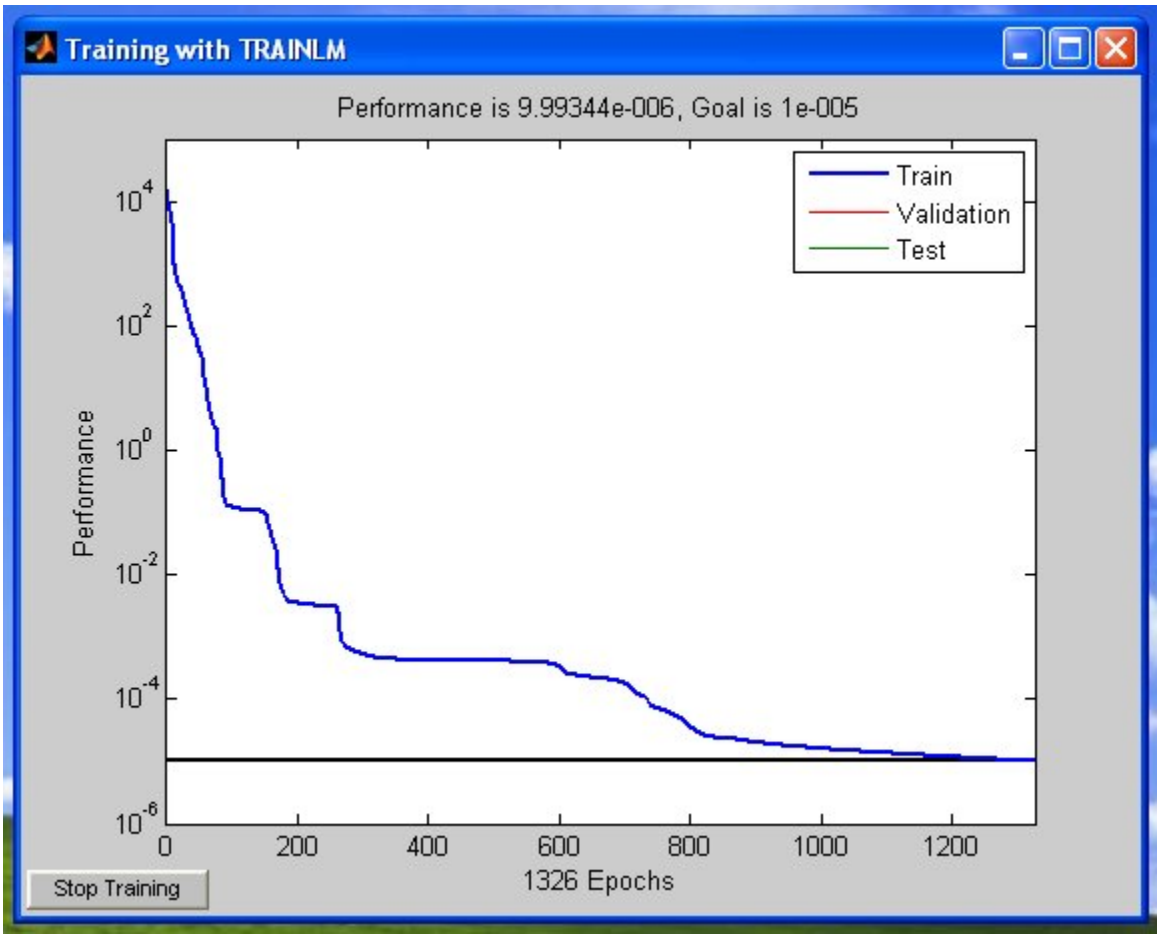


Figure 6.2: Output of Training Dataset-using Trainlm Training Function

The graph shown in figure 6.2 represents the output of the training of the network and 1326 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

6.2.1.3 Single Processing

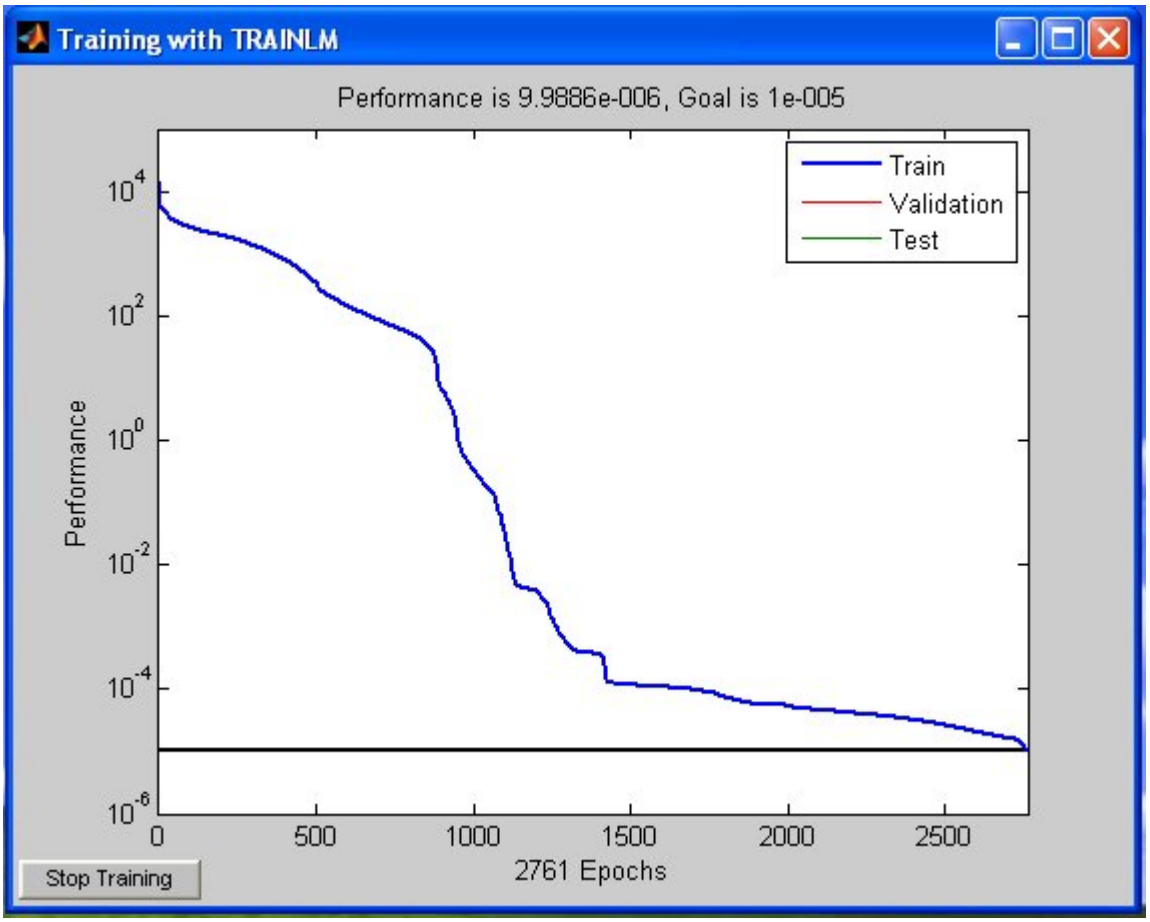


Figure 6.3: Output of Testing Dataset-using Trainlm Training Function

The graph shown in figure 6.3 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

6.2.1.4 Parallel Processing

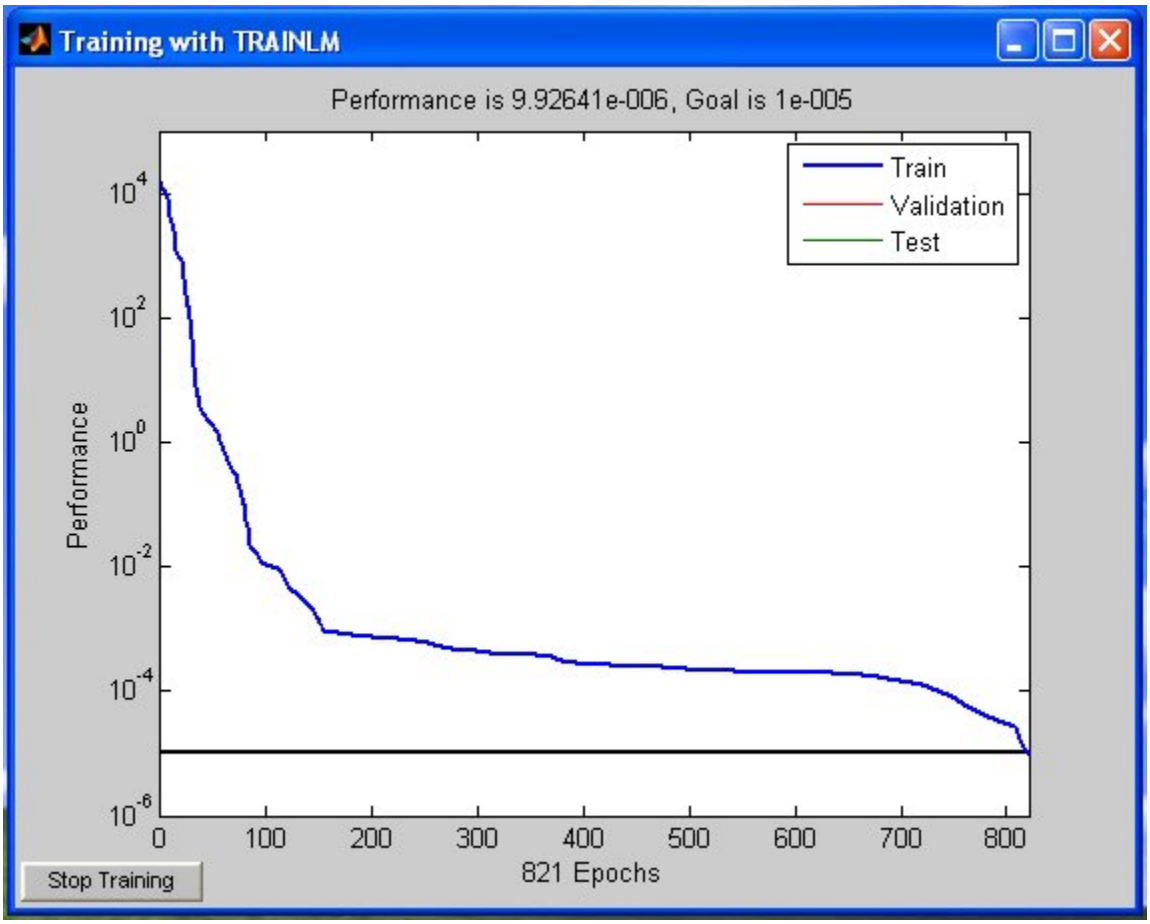


Figure 6.4: Output of Testing Dataset-using Trainlm Training Function

The graph shown in figure 6.4 represents the output of the training of the network and 821 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

6.3 Different Node Status in Distributed Environment

6.3.1 Nodestatus1

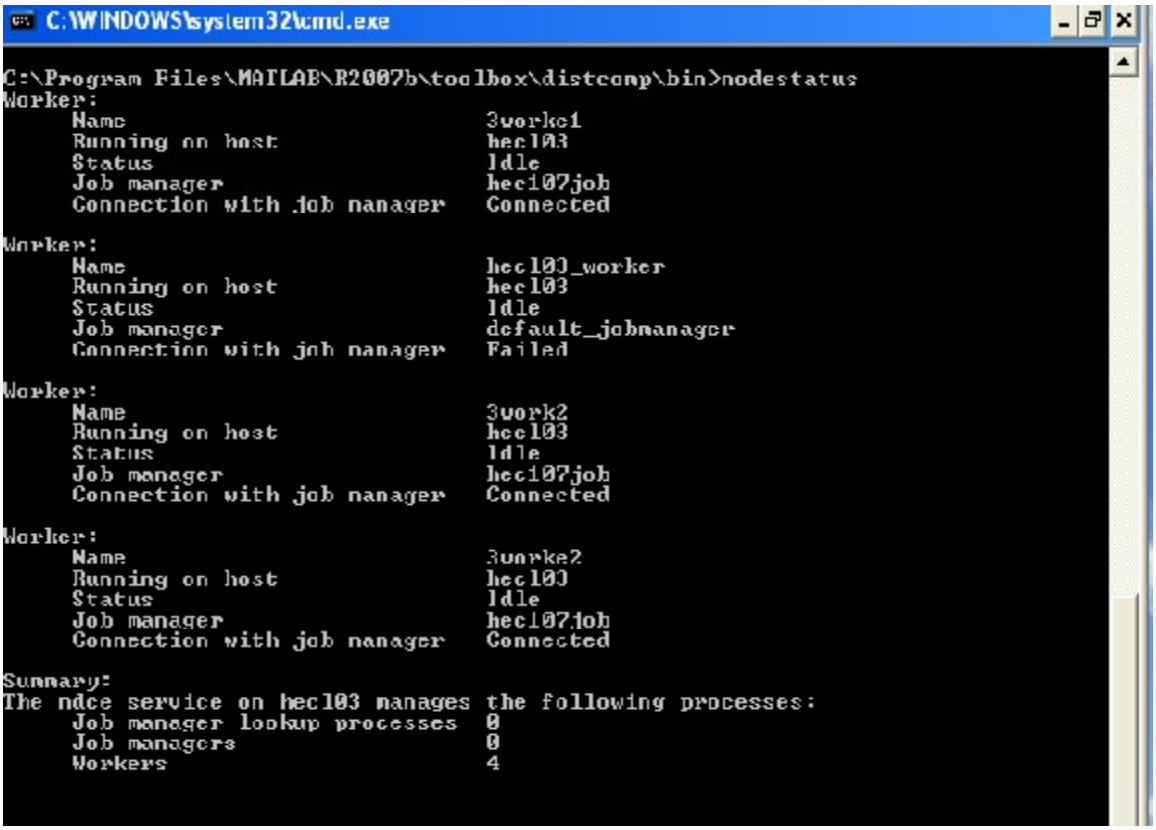


Figure 6.5: Node Status of the Worker on System hec103

As shown in the figure 6.5 this is one of the node status in distributed environment, there are four workers are running in this hec103 host and three workers are (3worke1, 3work2,3worke2) are connected with job manager name with hec107job on the system 172.31.5.92. And one worker name with hec103_worker is not connected with job manager, connection is failed because default_ job manager is not running on the system 172.31.5.92.

6.3.2 Nodestatus2

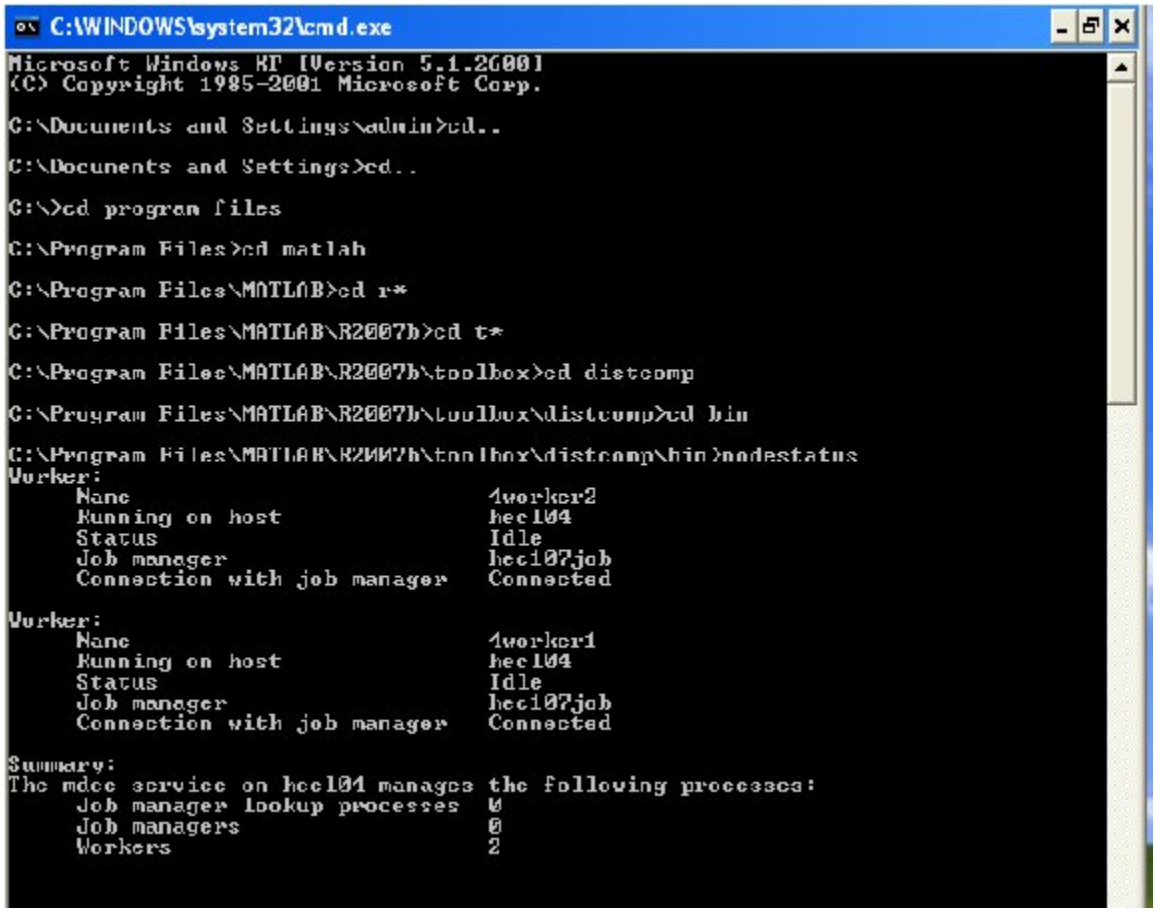


Figure 6.6: Node Status of the Worker on System hec104

As shown in the figure 6.6 this is one of the node status in distributed environment, there are two workers are running in this hec104 host and two workers are (4worker1,4worker2) are connected with job manager name with hec107job on the system 172.31.5.92

6.3.4 Nodestatus3

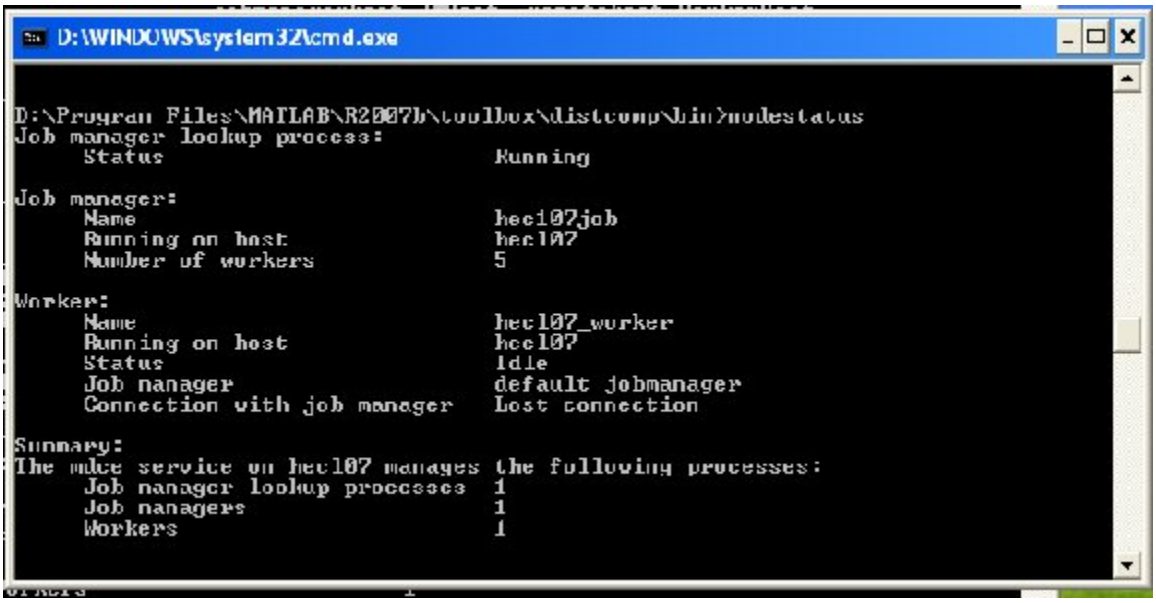


Figure 6.7: Node Status of the Job Manager on System hec107

As shown in the figure 6.7 this is job manager node status in distributed environment, there are on job manager lookup processes and one job manager and one worker, hec107_worker is not connected with default_jobmanager, connection is lost, because default_jobmanager is not running on this node hec107 and five other workers are connected with this job manager name with hec107job as described in the above node statuses

Chapter 7 Conclusion

INTRODUCTION

This chapter discusses about the conclusion that are made from the results obtained in the previous chapter and also about the limitation and future work.

7.1 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 back propagation 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 back propagation neural network by parallel processing is faster as compared to the single processing. This is because the data has been processed parallely. In case of parallel processing, training of back propagation neural network has achieved the performance goal with desired accuracy of the results.

The training of back propagation neural network algorithm has 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 back propagation neural network model gives the faster training convergence time and higher accuracy of the results.

7.2 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 number of worker in the Job manager depends to achieved higher performance of the system. In the future work this

system can be trained with different training function of the Mat Lab environment. As showed in this thesis work, back propagation neural networks can be successfully to implement the distributed environment system for data processing. The same experiments should also be conducted with other types of neural network to see the different types can improve the performance of the system as we got the experiments results with the back propagation neural network.

References

[1] R.W. Means,"High speed parallel hardware performance issues," in IEEE international conference on Neural network, pp 10-16, June 28-July 2 1994.

[2] T.Nordstrom and B.sevensson,"using and designing massively parallel computers for artificial neural networks," journal of parallel and distributed computing vol 14,pp 260-285,March 1992.

[3] W.M.Lin, V.K.Prasanna, and K.W.Przytula,"Algorithmic mapping of neural network models onto parallel SIMD machines," IEEE Transactions on computers. Vol.40, pp.1390-1401, December 1991.

[4] Y.Fujimoto, N.Fukuda, and T.Akabane,"Massively parallel architecture for large scale neural network simulation," IEEE Trans. On Neural networks, vol 3, pp 876-887, November 1992.

[5] N.Morgon, et.al."The ring array processor (RAP): A multiprocessing peripheral for connectionist applications," journal of parallel and distributed computing, vol 14, 1992.

[6] K.Hwang and F.A.Briggs, Computer architecture and parallel processing. McGraw-Hill Book Company, 5th ed., 1989.

[7] V.Kumar,"A scalable parallel formulation of the back propagation algorithm for hypercube and related architectures," IEEE trans.on parallel and distributed systems, vol 5, pp.1073-1090, October 1994.

[8] A.M.Alhaj and H.Terada,"Exploiting parallelism in neural networks on a dynamic data-driven system," IEICE Trans. On fundamentals. Pp.1804-1811, October 1993.

[9] T.yukawa and T.ishikawa,"optimal parallel back-propagation schemes for mesh connected and bus-connected multiprocessors," in proc.of IEEE int. conference on neural network, pp 748-753, 1993.

[10] C.Rosenberg and G.Blelloch,"An implementation of network learning on the connection Machine," in connectionist Models and their implications. Pp.329-340, Ablex, Norwood, NJ, 1988.

[11] V.kumar, A.grama, A.Gupta “introduction to parallel computing .ch 5.Benjamin – Cummings, 1993.

[12] H.yoon, j.H.nang and s. Maeng,”parallel simulation of multilayered neural networks on distributed-memory multiprocessors,” micro processing and microprogramming, vol 29, pp 185-190, 1990.

[13] E.franzi, “neural accelerator for parallelization of back propagation algorithm,” micro processing and microprogramming, vol 38,pp 689-696,1993.

[14] L.Utne, Design of a reconfigurable neurocomputer performance analysis by implementation of recurrent associative memories. PhD thesis, Norwegian institute of technology, 1995.

[15] N.Morgan,” Using a million connections for continues speech recognition,” in proc of int.conferece on Neural network processing pp.1439-1444, October 1994.

[16] K.Asaovic, J.Beck, J.Feldman, N.Morgan, “Designing a connectionist network supercomputer,” international journal of Neural network systems, vol 4. Pp 317-326, December 1993.

[17] R. Foltyniewicz and S.Skoneczny,” An improved high order neural network for invariant recognition of human faces in gray scale,” in proc of world congress on Neural Network, pp 587-592, 1994.

[18] J.N. Weinstein,”Neural network in the biomedical science: A survey of 386 publications since the beginning of 1991,”in proc of world congress on neural network, vol 1, pp.121-126 1994.

[19] L.Tsinas and V.Graefe,”Coupled neural networks for real-time road and obstacle recognition by intelligent road vehicles,” in proc. Of int, joint conference on neural networks, (Japan) 1993.

[20] N. Sonehara, et al,” Image data compression using a neural network model,” in proc of int. joint conference on neural network, vol 2,pp.35-40, 1989.

[21] D.Hammerstrom,” Neural networks at work,” IEEE spectrum, vol.30, no.6. Pp.26-32, 1993.

[22] P. Treleven, Neurocomputers. Research Note 89/8, Department of computer science, University College London, January 1989.

[23] M.E.Azema-Barac, A generic strategy for mapping neural network models o transpter-based machines, pp, 244-249.

[24] U.Muller, B.Baumle, P.Kohler,"Achieving supercomputer performance for neural net simulation with an array of digital signal processors," IEEE micro, pp.55-65, October 1992.

[25] L.E.Atlas and Y.Suzuki,"Digital systems for artificial neural network," IEEE circuits and Devices magazine, pp.20-24, Nov 1989.

[26] Chakraverty S, Bhat R B and Stiharu I, (1999), "Recent research on vibration of structures using boundary characteristic orthogonal polynomials in the Rayleigh-Ritz method", Journal of Shock Vibration Digest, 31(3), pp 187-194.

[27] Singh B and Chakraverty S, (1992 a), "On the use of orthogonal polynomials in Rayleigh-Ritz method for the study of transverse vibration of elliptic plates", Journal of Computers and Structures, 43(3), pp 439.

[28] Singh B and Chakraverty S, (1992 b), "Transverse vibration of simply-supported elliptic and circular plates using boundary characteristic orthogonal polynomials in two dimensions", Journal of Sound and Vibration, 152(1), pp 149.

[29] http://www.mathworks.com/products/parallel-computing

[30] Antonio d' Acierno" Back-propagation learning algorithm and parallel computers: The CLEPSYDRA mapping scheme" I.R.S.I.P.-C.N.R. via P. Castellino, 111-80131 Napoli-Italy.

[31] S. Mahapatra *, R.N. Mahapatra, B.N. Chatterji "A parallel formulation of back-propagation learning on distributed memory multiprocessors" Department of Electronics & Electrical Communication Engineering, Indian Institute of Technology. Kharqpur 721 302.

[32] P. SaratchandranN.Sundararajan Shou King Foo" Parallel Implementation of Back propagation Neural Network on a Heterogeneous RING processor Topology" School of Electrical & Electronic Engineering Nan yang Technological University.

[33] Udo Seiffert "Artificial Neural Networks on Massively Parallel Computer Hardware "University of Magdeburg, Germany Institute of Electronics, Signal Processing and Communications.

Paper Communicated

1. Kiran kumar.k, V.P.Singh," **Parallelization of Back propagation Neural Network Algorithm for Plate Vibration Analysis** "at national conference on "national conference on communication and networking"NCCN-09, Sant Longowal Institute of Engineering & Technology longowal, sangrur. [Communicated]