Quantum Neural Network Training Algorithm

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

Submitted By:

Mukesh Kumar

(Roll No. 80732014)

Under the supervision of:

Dr. V.P. Singh
Assistant Professor

JUNE 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, **"Quantum Neural Network Training Algorithm"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Computer Science & Engineering submitted in Computer Science and Engineering Department 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 researchers' 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.

mukeshkumar

(Mukesh Kumar)

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)
Assistant Professor
Computer Science and Engineering Department
Thapar University
Patiala

Countersigned by

(Dr. Rajesh Bhatia) 15/07/09
Head
Computer Science & Engineering Department
Thapar University
Patiala.

RK Sharma 14/7

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

Acknowledgment

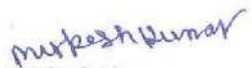
I wish to express my deep gratitude to Dr. V.P. Singh, Assistant Professor, computer Science & engineering department, TU, Patiala for providing his uncanny guidance and support throughout the thesis.

I am thankful to Dr. Rajesh Bhatia, Head, Computer Science & Engineering Department, TU, Patiala, for the motivation and inspiration that triggered me for the thesis work. I would also like to thank all the staff members who were always there at the need of the hour and provided with all the help and facilities, which I required for the completion of the thesis.

Last but not the least, I express my heartfelt thanks to all my friends Sushil Kumar, Subhash Junas, Chetan Jain. Amit Gupta, Vineet Khera, Mayank Kumar, Jasmeet Singh, Vaibhav Bhadade, for encouraging me and providing me useful information during my work.

Most importantly, I would like to give God the glory for all of the efforts I have put into this report.

Finally, my special thanks go to authors whose works I have consulted and quoted in this work.



Mukesh Kumar
80732014
M.E.(Computer Science & Engineering)
Computer Science & Engineering Department
Thapar University
Patiala -147004

Abstract

Quantum neural networks(QNN) refers to the class of neural network models, artificial or biological, which rely on principles inspired in some way from quantum mechanics. Many quantum neural networks have been proposed in the literature, but very few of these proposals have attempted to provide an in-depth method of training them. Most either do not mention how the network will be trained. This assumes that training a quantum neural network will be straightforward and analogous to classical methods. Several different network structures have been proposed. Several of these networks also employ methods, which are speculative or difficult to do in quantum systems. These significant differences between classical networks and quantum neural networks, as well as the problems associated with quantum computation itself, require to more deeply at the issue of training quantum neural networks.

The training of quantum neural network by different algorithm/method in the Matlab enviourment has been proposed and validated the some with data of viechle classification problem.

.

Keywords: Quantum Neural Network, Quantum Theory, Neural Network, Complex, Training algorithm, Complex valued function

Abbreviations

QNN	Quantum Neural Network
QANN	Quantum Artificial Neural Network
ART	Adaptive Resonance Theory
NN	Neural Network
ADAPT	Adaptive Training
LEARDGD	Learning Gradient Descent
LEARDGDM	Gradient Decent With Momentum
TRAINGD	Training Gradient Descent
TRAINGDM	Gradient Descent With Momentum

Table of Contents

List of Figures

List of Tables

Chapter1

Introduction

1.1 Quantum neural network:

In the past years, there has been a growing interest in the artificial neural network based on quantum theoretical concepts, in spite of this ANN has many limitations including the absence of rules for determining optimal architectures, limited memory capacity, catastrophic forgetting due to the pattern interference. Quantum Neural Network (QNN) is a burgeoning new field built upon the combination of classical neural networks and quantum computations.

Many quantum neural networks have been proposed, but very few of these proposals have attempted to provide an in-depth method of training them. Most either do not mention how the network will be trained or simply state that they use a standard gradient descent algorithm. This assumes that training a quantum neural network will be straightforward and analogous to classical methods. While some quantum neural networks seem quite similar to classical networks, others have proposed quantum networks that are vastly different. Several different network structures have been proposed, including lattices and dots. Several of these networks also employ methods which are speculative or difficult to do in quantum systems. These significant differences between classical networks and quantum neural networks, as well as the problems associated with quantum computation itself, requires to look more deeply at the issue of training quantum neural networks. Furthermore, little has been done that is empirical testing on their training methods to show that their methods work with real-world problems [1].

There are many advantages of Quantum neural network (QNN) over a classical network.

- QNNs should have roughly the same computational power as classical networks.
- QNNs may work best with some classical components as well as quantum components.

- Quantum searches can be proven to be faster than comparable classical searches. A new training method has been purposed for a Quantum Neural Network [2].

The development of quantum neural networks holds following two main reasons:

- One has its origin in arguments for the essential role which quantum processes play in the living brain. For example, new physics binding quantum phenomena with general relativity can explain such mental abilities as understanding, awareness and consciousness].However, this approach advocates the study of intracellular structures, such as microtubules rather than that of the networks of neurons themselves.
- The quantum domain by eclectic combination of that field with the promising new field of quantum computing which is the generalized field of classical artificial neural networks.

Both considerations suggest new understanding of mind and brain function as well as new unprecedented abilities in information processing. Here consider quantum neural networks as the next natural step in the evolution of neurocomputing systems, focusing on artificial rather than biological systems. The outline of different approaches to the realization of quantum distribute processing and argue that, as in the case of quantum computing [3].

The main analogies analogies in establishing a connection between quantum mechanics and neural networks shown in the table 1.

Table 1.1: Concepts of quantum mechanics and neural networks

Quantum mechanics	**Neural Networks**
wave function	Neuron
Measurement (decoherence)	evolution to attractor
Entanglement	learning rule
unitary transformations	gain function (transformation)

1.2 Quantum Theory

Quantum theory is set within in the space times of general relativity, a complete adaptation of quantum theory and Einstein's general theory of relativity remains beyond grasp

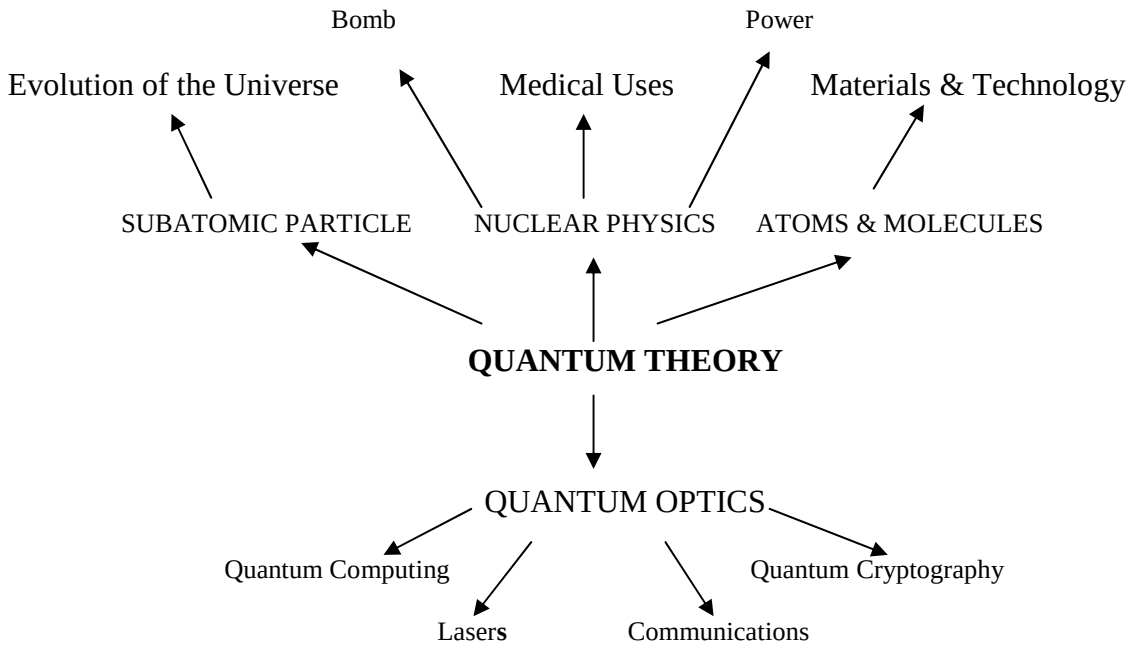


Figure1.1: Quantum Theory

Quantum theory is a theory of matter; or more precisely it is a theory of the small components that comprise familiar matter. The ordinary matter of tables and chairs, omelets and elephants is made up of particles, like electrons, protons and neutrons. Quantum theory provides the best account of these particles. It also provides an account of matter in the form of radiation, such as light. It is commonly known that light somehow consists both of light waves and also particle-like photons. The notion of these photons comes from quantum theory.

Quantum computation and quantum information encompasses processing and transmission of data stored in quantum states. The central novelty of quantum theory lies in the description of the state of these particles. In some ways, the particles of quantum theory are like little tiny points of matter, as the name "particle" suggests. In others, they

are like little bundles of waves. Modern quantum theory has enjoyed enormous empirical success, accounting for a huge array of phenomena and making striking predictions. It is possible to describe the basic posits of quantum theory compactly.

1.3 Neural Network

The study of the human brain is thousands of years old. The history of neural networks that was described can be divided into several periods.

In general, the human nervous system is a very complex neural network. The brain is the central element of the human nervous system, consisting of near 10^{10} biological neurons that are connected to each other through sub-networks. Each neuron in the brain is composed of a body, one axon and multitude of dendrites. The neuron model shown in Figure1.2 serves as the basis for the artificial neuron. The dendrites receive signals from other neurons. The axon can be considered as a long tube, which divides into branches terminating in little end bulbs. The small gap between an end bulb and a dendrite is called a synapse. The axon of a single neuron forms synaptic connections with many other neurons. Depending upon the type of neuron, the number of synapses connections from other neurons may range from a few hundreds to 10^4. The cell body of a neuron sums the incoming signals from dendrites as well as the signals from numerous synapses on its surface. A particular neuron will send an impulse to its axon if sufficient input signals are received to stimulate the neuron to its threshold level. The interest in neural networks comes from the networks' ability to mimic human brain as well as its ability to learn and respond [4].

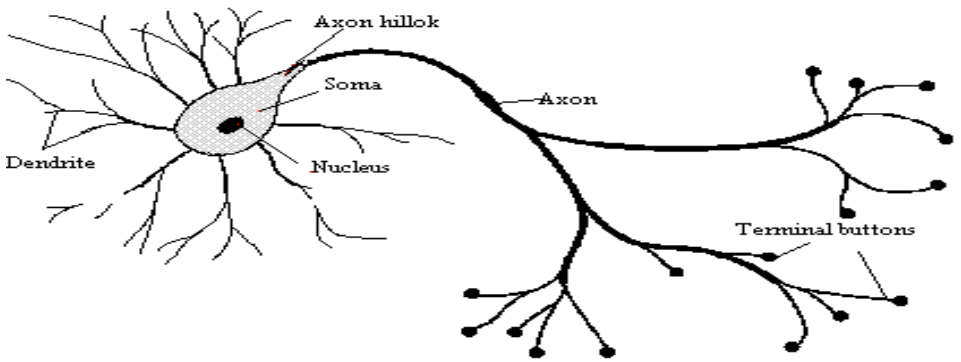


Figure 1.2: A Biological Neuron

As a result, neural networks have been used in a large number of applications and have proven to be effective in performing complex functions in a variety of fields. These include pattern recognition, classification, vision, control systems, and prediction [5,6]. Adaptation or learning is a major focus of neural net research that provides a degree of robustness to the NN model. In predictive modeling, the goal is to map a set of input patterns onto a set of output patterns. NN accomplishes this task by learning from a series of input/output data sets presented to the network. The trained network is then used to apply has been learned to approximate or predict the corresponding output [7].

CHAPTER 2

Literature Review

2.1 Quantum theory

Quantum neural network has been discussed by Alexandr A. Ezhov and Dan Ventura. The power of classical artificial neural networks is due to their massively parallel, distributed processing of information and also due to the nonlinearity of the transformation performed by the network nodes (neurons). On the other hand, quantum mechanics offers the possibility of an even more powerful quantum parallelism which is expressed in the principle of superposition. This principle provides quantum computing an advantage in processing huge data sets. Though quantum computing implies parallel processing of all possible configurations of the state of a register composed of N qubits, only one result can be read after the decoherence of the quantum superposition into one of its basis states. However, entanglement provides the possibility of measuring the states of all qubits in a register whose values are interdependent [3].

The choice of interpretation is important in establishing different analogies between quantum physics and neurocomputing.

The field of neural networks contains several important basic ideas, which include the concept of a processing element (neuron), the transformation performed by this element (in general, input summation and nonlinear mapping of the result into an output value), the interconnection structure between neurons, the network dynamics, and the learning rule which governs the modification of interconnection strengths. A major dichotomization of neural networks can be realized by considering whether they are trained in a supervised or unsupervised manner. An example of the latter is the Hopfield model of content-addressable memory using the concept of attractor states [8]. A Hopfield network as a reference point for the consideration of neural models in general. In fact, the Hopfield model itself was proposed during a previous “invasion” of physics into the theory of artificial neural networks in 1982. Hopfield was discovered an analogy between networks with symmetrical bonds and spin glasses. While quantum mechanics is a linear

theory, neurocomputing is very dependent upon nonlinear approaches to data processing. At first glance, this appears to complicate the establishment of a correspondence between the two fields. However there are different ways to overcome this difficulty.

To reconciling the linearity of quantum mechanics with the nonlinearity inherent in artificial neural networks, quantum mechanics has been considered, which is based on the use of path integrals. Here nonlinearity can be rdue both to the nonlinear form of the potential V(x) and also to the operation of the exponent. This fact has been used in approaches to modeling quantum neural networks by Elizabeth Behrman and coworkers[9,10] and Ben Goertzel[11]. Some analogies used for the development of quantum neural networks are summarized in Table2.

Table2.1: Summarized the Development of QNN

Model	Neuron	Connections	Transformation	Network	Dynamics
Perus	Quantum	Green function	Linear	Temporal	Collapse as convergence
Chirsly	Classical(slit position)	Classical(slit position)	Nonlinear through superposition	Multilayer	Non superpositional
Behrman-et al	Time slice, quantum	Interactions through phonons	Nonlinear through potential energy & exponent function	Temporal & spatial	Feynman path integral
Goertzel	Classical	Quantum	Nonlinear	classical	Feynman path integral

Menneer & Narayanan	Classical	Classical	Nonlinear	Single item network in many universes	Classical
Ventura	Qubit	Entangle-ment	--------	Single item modules in many universes	Unitary & non unitary transfor-mations

Behrman et al. first developed a temporal model of a quantum neural network which utilizes a quantum dot molecule coupled to a substrate lattice through optical phonons. In this model temporal evolution of the system resembles the equations for virtual neurons and the timeline discretization points for the Feynman path integral serve as these virtual neurons. The concept of neurons used here is rather artificial, and in fact the number of neurons depends on the parameters of the temporal discretization scheme, rather than on the number of quantum particles involved. Recently, this group working at Wichita State University proposed a spatial model for a quantum neural network based on the use of a spatial array of quantum dot molecules. It was shown that any logical gate, including a purely quantum one phase shift can be performed using these systems [3].

Using the metatheory of quantum mechanics as a starting point, the field of artificial neural networks can described with that of quantum computation in a natural way. For the purposes it is sufficient to consider the application of neural approaches, in their simplest forms, to pattern recognition. Then see how a concept of quantum neural networks naturally emerges from the theory of neurocomputing.

2.2 Quantum Neural Network

The quantum computing's field is as yet in its formative years with only a few theoretical models and almost no physical hardware to display. Despite this, it is the subject of considerable interest due to the necessity of understanding computational

limits brought on by quantum effects and the ever-decreasing size of electronic componentry.

A theoretical quantum model produced by Feynman that could, in principle, reproduce serial computation. Feynman noticed that the truth tables of all of the basic logical functions could be reproduced by quantum mechanical raising and lowering operators acting on up/down spin sites. In particular, an operator constructed from the required combination of raising and lowering operators (and their adjoints) was found to be Hermitian, permitting its useas the Hamiltonian in the Schrodinger equation[12].

Margolus extended Feynman's model to one that worked as a two-dimensional cellular automaton with nearest neighbor updating, showing in the process that the original was actually a specific case of this more general model. It was provided a formal description of parallel quantum computation to complement that of serial computation[13].

The concept of Quantum Neural Computation firstly presented by Kak. It generated a new paradigm upon the combination of conventional neural computation and quantum computing. In 1998, a first systematic examination of quantum artificial neural network (QANN) was conducted by Menneer in his PhD dissertation. At the same time, many QANN model were developed. Menneer and Narayanan was proposed a quantum inspired neural nets in 1995. In 2000, Ventura and Martinez introduced quantum associative memory and entangled neural nerworks was presented by Li wei-gang. In the last year , Noriaki Kouda et al. introduced qubit neural networks. Unfortunately, these models have not the obvious concept of network weight and their training algorithms were not mentioned in detail upon Grover quantum search algorithm. The QNN is considered to have at least the same computational power as classical networks. Other results have shown that QNNs may work best with some classical components as well as quantum components, owning to the power of quantum.

But Quantum searches can be proven to be faster than comparable classical searches [12].Leverage this idea to propose a new training method for a simple QNN. In this thesis details such a network and how training could be done on it. Results from testing the algorithm on several real-world problems show that it works.

2.3 Neural Network

There were some initial simulations using formal logic. McCulloch and Pitts (1943) developed models of neural networks based on their understanding of neurology. These models made several assumptions about how neurons worked. Their networks were based on simple neurons which were considered to be binary devices with fixed thresholds. The results of their model were simple logic functions such as "a or b" and "a and b". Another attempt was by using computer simulations. Two groups (Farley and Clark, 1954; Rochester, Holland, Haibit and Duda, 1956). The first group (IBM researchers) maintained closed contact with neuroscientists at McGill University. So whenever their models did not work, they consulted the neuroscientists. This interaction established a multidisciplinary trend which continues to the present day.

In 1969 according to Minsky and Papert generalized the limitations of single layer Perceptrons to multilayered systems. In the book they said: "... intuitive judgment that the extension (to multilayer systems) is sterile". The significant result of their book was to eliminate funding for research with neural network simulations. The conclusions supported the disenchantment of researchers in the field. As a result, considerable prejudice against this field was activated.

During the period several paradigms were generated which modern work continues to enhance. Grossberg's (Steve Grossberg and Gail Carpenter in 1988) influence founded a school of thought which explores resonating algorithms. They developed the ART (Adaptive Resonance Theory) networks based on biologically plausible models. Anderson and Kohonen developed associative techniques independent of each other. Klopf (A. Henry Klopf) in 1972 developed a basis for learning in artificial neurons based on a biological principle for neuronal learning called heterostasis. Werbos (Paul Werbos 1974) developed and used the back-propagation learning method, however several years passed before this approach was popularized. Backpropagation nets are probably the most well known and widely applied of the neural networks today. In essence, the back-propagation net. is a Perceptron with multiple layers, a different threshold function in the artificial neuron, and a more robust and capable learning rule.

Amari (A. Shun-Ichi 1967) was involved with theoretical developments and established a mathematical theory for a learning basis (error-correction method) dealing with adaptive pattern classification.While Fukushima (F. Kunihiko) developed a step wise trained multilayered neural network for interpretation of handwritten characters. The original network was published in 1975 and was called the Cognitron[14,15].

During the late 1970s and early 1980s was important to the re-emergence on interest in the neural network field. Several factors influenced this movement. For example, comprehensive books and conferences provided a forum for people in diverse fields with specialized technical languages, and the response to conferences and publications was quite positive. The news media picked up on the increased activity and tutorials helped disseminate the technology.

Rumelhart, Hinton and Williams reported back on the developments of the back-propagation algorithm in 1986 and discussed how back-propagation learning had emerged as the most popular learning set for the training of multi-layer perceptrons. With the dawn of the 1990's and the technological era, many advances into the research and development of artificial neural networks are occurring all over the world. Nature itself is living proof that neural networks do in actual fact work. The challenge today lies in finding ways to electronically implement the principals of neural network technology. Electronics companies are working on three types of neuro-chips namely, digital, analog, and optical. With the prospect that these chips may be implemented in neural network design, the future of neural network technology looks very promising.

2.4 Elements of Neural Network

2.4.1 Weighting Factors An artificial neuron is the basic element of a neural network. It consists of three basic components that include weights, thresholds, and a single activation function. The values W_1,W_2,W_3 ,…..,Wn are weight factors associated with each node to determine the strength of input row vector $X = [x_1\ x_2\ x_3\ldots..,x_n]^T$. Each input is multiplied by the associated weight of the neuron connection X^TW. Depending upon

the activation function, if the weight is positive, X^TW commonly excites the node output; whereas, for negative weights, X^TW tends to inhibit the node output.

2.4.2 Threshold

The node's internal threshold is the magnitude offset that affects the activation of the node output y as follows

$$y = \sum_{i=1}^{n} (X_i W_i) - \quad k$$

2.4.3 Linear Function

A linear function satisfies the superposition concept. The function is shown in Figure2.1

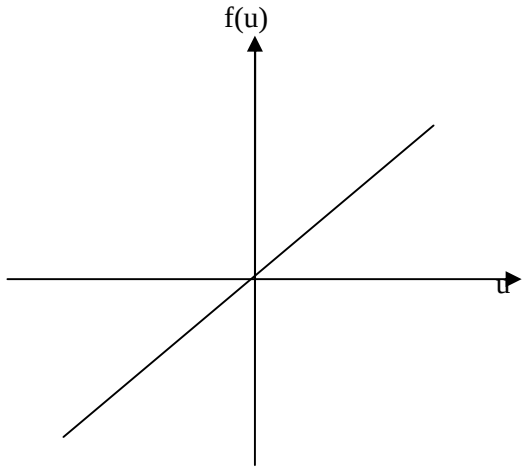


Figure 2.1: Linear Function.

2.4.4 Threshold Function

A threshold (hard-limiter) activation function is either a binary type or a bipolar type as shown in Figures and , respectively. The output of a binary threshold function can be written as:

$$Y = f(u) = \begin{cases} 0 \text{ if } u<0 \\ \\ 1 \text{ if } u \geq 0 \end{cases}$$

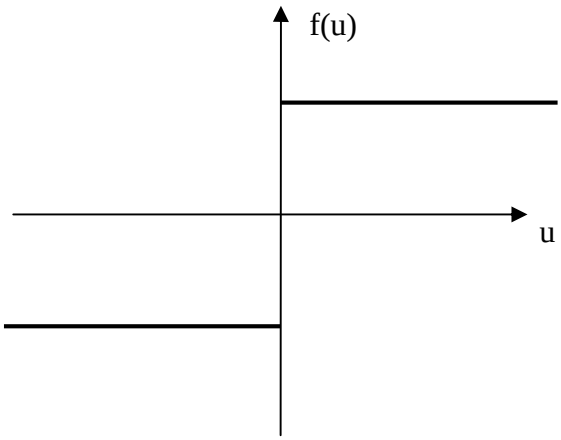


Figure 2.2: Binary Threshold Activation Function

The neuron with the hard limiter activation function is referred to as the McCulloch-Pitts model.

2.4.5 Piecewise Linear Function

This type of activation function is also referred to as saturating linear function and can have either a binary or bipolar range for the saturation limits of the output. The mathematical model for a symmetric saturation function is described as follows.

$$y = f(u) = \begin{cases} -1 & if \quad u < -1 \\ u & if \quad -1 \geq u \geq 1 \\ 1 & if \quad u \geq 1 \end{cases}$$

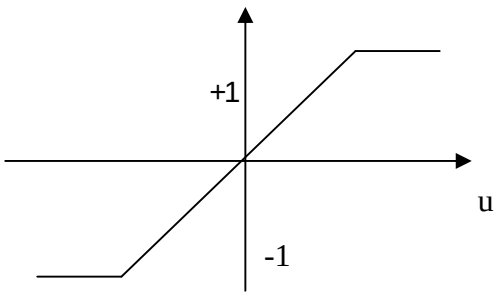


Figure 2.3: Piecewise Linear Activation Function

2.4.6 Sigmoidal (Sshaped) function

This nonlinear function is the most common type of the activation used to construct the neural networks. It is mathematically well behaved, differentiable and strictly increasing function. A sigmoidal transfer function can be written in the following form

where is the shape parameter of the sigmoid function. By varying this parameter, different shapes of the function can be obtained as illustrated in Figure 2.5. This function is continuous and differentiable.

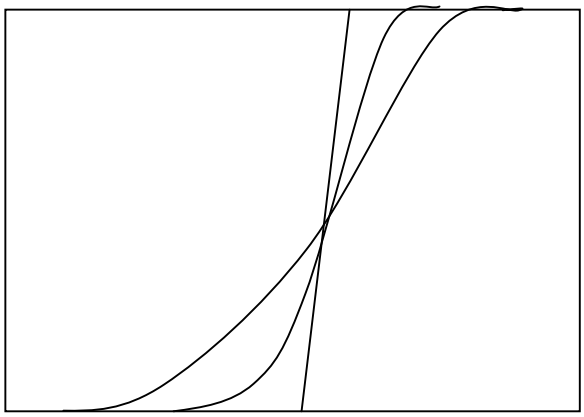


Figure 2.4:A Sigmoid Function

2.4.7 Tangent hyperbolic function:

This transfer function is described by the following mathematical form:

2.5 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 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. 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:

(a) The neural network is stimulated by an environment.

(b) The neural network undergoes changes in its free parameters as a result of the stimulation.

(c) 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 in way 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.

2.5.1 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.

2.5.2 Unsupervised Learning

Unsupervised learning uses no external information, it 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. 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 perform online.

2.6 PERCEPTRON

There are two type of perceptron of the Neural Network

- Single layer perceptron
- Multilayer perceptron

2.6.1 SINGLE LAYER PERCEPTRON

General Architecture

The original idea of the perceptron was developed by Rosenblatt in the late 1950s along with a convergence procedure to adjust the weights. In Rosenblatt's perceptron, the inputs were binary and no bias was included. It was based on the McCulloch-Pitts model of the neuron with the hard limitation activation function. The single layer perceptron as shown in Figure 2.5 is very similar to ADALINE except for the addition of an activation function**.**

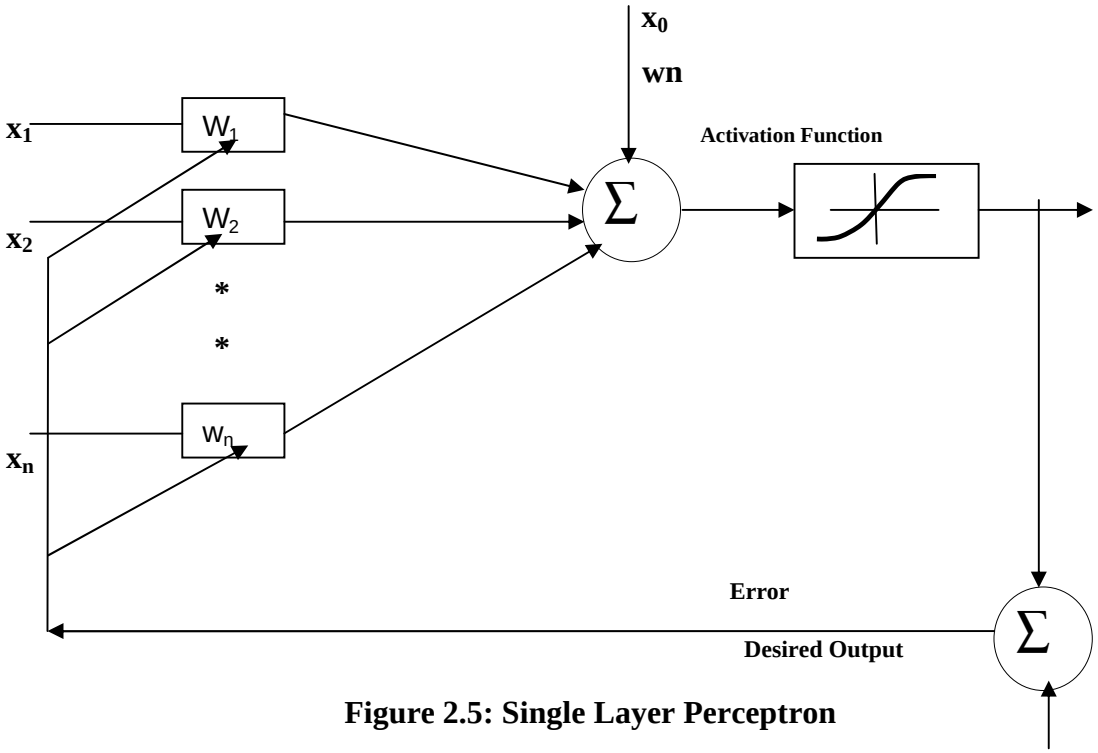


Figure 2.5: Single Layer Perceptron

A Perceptron with a Sigmoid Activation Function. Connection weights and threshold in a perceptron can be fixed or adapted using a number of different algorithms. Here the original perceptron convergence procedure as developed by Minsky and Papert[2] is described. First, connection weights W_1, W_2,…,Wn and the threshold value W_0 are initialized to small non-zero values. Then, a new input set with N values received through sensory units (measurement devices) and the input is computed. Connection weights are only adapted when an error occurs.

2.6.2 MULTI-LAYER PERCEPTRON

Multi-layer perceptrons represent a generalization of the single-layer perceptron as described in the previous section. A single layer perceptron forms a half–plane decision region. On the other hand multi-layer perceptrons can form arbitrarily complex decision regions and can separate various input patterns. The capability of multi-layer perceptron stems from the non-linearities used within the nodes. If the nodes were linear elements, then a single-layer network with appropriate weight could be used instead of two- or three-layer perceptrons. Figure2.6 shows a typical multi-layer perceptron neural network structure. As observed it consists of the following layers.

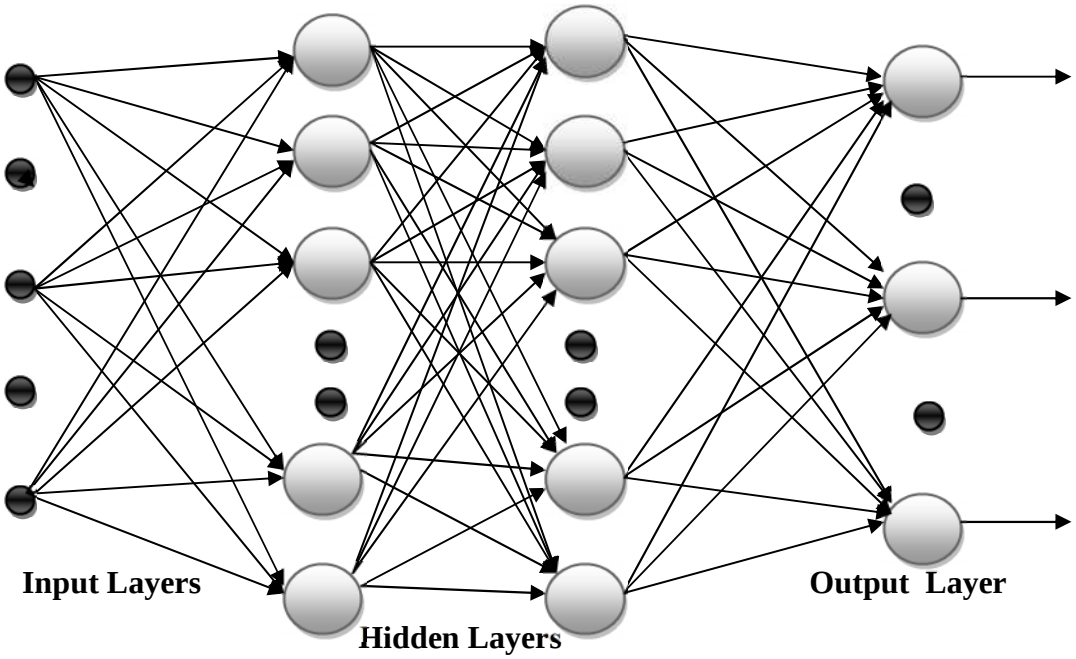


Figure 2.6: Multilayer Perceptron

(i) Input Layer : A layer of neurons that receives information from external sources, and passes this information to the network for processing. These may be either sensory inputs or signals from other systems outside the one being modeled.

(ii) Hidden Layer: A layer of neurons that receives information from the input layer and processes them in a hidden way. It has no direct connections to the outside world (inputs or outputs). All connections from the hidden layer are to other layers within the system.

(iii) Output Layer: A layer of neurons that receives processed information and sends output signals out of the system.

(iv) Bias: Acts on a neuron like an offset. The function of the bias is to provide a threshold for the activation of neurons. The bias input is connected to each of the hidden and output neurons in a network [4].

(vi) Input-Output Mapping The input/output mapping of a network is established according to the weights and the activation functions of their neurons in input, hidden and

output layers. The number of input neurons corresponds to the number of input variables in the neural network, and the number of output neurons is the same as the number of desired output variables. The number of neurons in the hidden layer(s) depends upon the particular NN application. For example, consider the following two-layer feed-forward network with three neurons in the hidden layer and two neurons in the second layer linear relationship for each layer

$$A_1 = W_1 X$$

The linear activity level of the hidden layer (neurons n1 to n3) can be calculated as follows

$$\begin{cases} a_{11} = w_{11} i_1 + w_{21} i_2 \\ a_{21} = w_{12} i_1 + w_{22} i_2 \\ a_{13} = w_{13} i_1 + w_{23} i_2 \end{cases}$$

The output vector for the hidden layer can b e calculated by the following formula:

$$O_1 = F \,.A_1$$

2.7 Backpropagation

2.7.1 Architecture: The backpropagation algorithm architecture which is commonly used for most network i.e. the multilayer feedforward network

2.7.2 Neuron Model (TANSIG, LOGSIG, PURELIN)

An elementary neuron with *R* inputs is shown below. Each input is weighted with an appropriate *w*. The sum of the weighted inputs and the bias forms the input to the transfer function *f*. Neurons may use any differentiable transfer function *f* to generate their output.

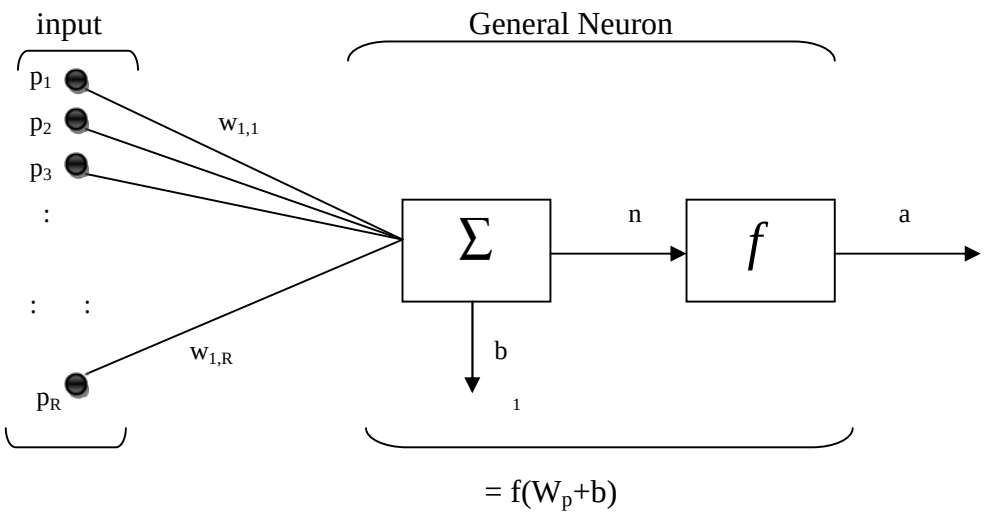


Figure 2.7: Neuron Model

Where R = Element in input vector

2.7.3 Feedforward Network

A single-layer network of *S* logsig neurons having *R* inputs is shown below in full detail on the left and with a layer diagram on the right.

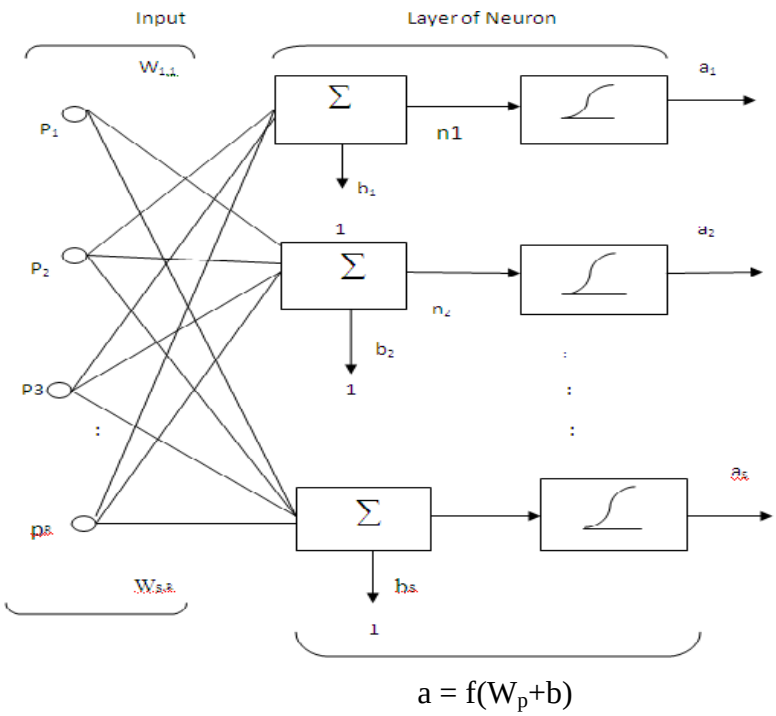


Figure2.8: Feedforward Network

Feed forward networks often have one or more hidden layers of sigmoid neurons followed by an output layer of linear neurons. Multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear and linear relationships between input and output vectors.

The linear output layer lets the network produce values outside the range –1 to +1,if it is desirable to constrain the outputs of a network (such as between 0 and 1) then the output layer should use a sigmoid transfer function (such as logsig)[16].

2.7.4 Backpropagation Algorithm

There are many variations of the backpropagation algorithm, several of which will be discussed. The simplest implementation of backpropagation learning updates the network weights and biases in the direction in which the performance function decreases most rapidly – the negative of the gradient. One iteration of this algorithm can be written

$$X_{K+1} = X_K - {}_k g_k$$

Where X_k is a vector of current weights and biases, g_k is the current gradient, and ${}_k$ is the learning rate.

There are two different ways in which this gradient descent algorithm can be implemented: incremental mode and batch mode. In the incremental mode, the gradient is computed and the weights are updated after each input is applied to the network. In the batch mode all of the inputs are applied to the network before the weights are updated. The next section will describe the incremental training, and the following section will describe batch training [16].

2.7.5 Gradient Descent (LEARDGD).

For the basic steepest (gradient) descent algorithm, the weights and biases are moved in the direction of the negative gradient of the performance function. For this algorithm, the individual learning function parameters for the weights and biases are set to 'learngd'. The following commands illustrate how these parameters are set for the feedforward network created earlier.

The function learngd has one learning parameter associated with it – the learning rate lr. The changes to the weights and biases of the network are obtained by multiplying the learning rate times the negative of the gradient The larger the learning rate, the bigger the step. If the learning rate is made too large the algorithm will become unstable. If the learning rate is set too small, the algorithm will take a long time to converge.

The learning rate parameter is set to the default value for each weight and bias when the learnFcn is set to learngd, as in the code above, although the value can be changed. The following command demonstrates, how can set the learning rate to 0.2 for the layer weights. The learning rate can be set separately for each weight and bias.

2.7.6 Gradient Descent With Momentum (LEARDGDM).

It is another incremental learning algorithm for feedforward networks that often provides faster convergence - learngdm, steepest descent with momentum. Momentum allows a network to respond not only to the local gradient, but also to recent trends in the error surface. Acting like a low pass filter, momentum allows the network to ignore small features in the error surface. Without momentum a network may get stuck in a shallow local minimum. With momentum a network can slide through such a minimum.

Momentum can be added to backpropagation learning by making weight changes equal to the sum of a fraction of the last weight change and the new change suggested by the backpropagation rule. The magnitude of the effect that the last weight change is allowed to have is mediated by a momentum constant, mc, which can be any number between 0 and 1. When the momentum constant is 0 a weight change is based solely on the gradient. When the momentum constant is 1 the new weight change is set to equal the last weight change and the gradient is simply ignored.

The learngdm function is invoked using the same steps shown above for the learngd function, except that both the mc and lr learning parameters can be set. Different parameter values can be used for each weight and bias, since each weight and bias has its own learning parameters.

2.7.7 Batch Training (TRAIN).

The alternative to incremental training is batch training, which is invoked using the function train. In batch mode the weights and biases of the network are updated only

after the entire training set has been applied to the network. The gradients calculated at each training example are added together to determine the change in the weights and biases.

2.7.8 Batch Gradient Descent (TRAINGD)

The batching equivalent of the incremental function learngd is traingd, which implements the batching form of the standard steepest descent training function. The weights and biases are updated in the direction of the negative gradient of the performance function. To train a network using batch steepest descent, the network trainFcn to traingd should be set the function train should be called. Unlike the learning functions in the previous section, which were assigned separately to each weight matrix and bias vector in the network, there is only one training function associated with a given network. There are seven training parameters associated with traingd: epochs, show, goal, time, min_grad, max_fail, and lr. The learning rate lr has the same meaning here as it did for learngd - it is multiplied times the negative of the gradient to determine the changes to the weights and biases. The training status will be displayed every show iterations of the algorithm. The other parameters determine when the training is stopped. The training will stop if the number of iterations exceeds epochs, if the performance function drops below goal, if the magnitude of the gradient is less than mingrad, or if the training time is longer than time seconds. max_fail is associated with the early stopping technique, in the section on improving generalization.Try the Neural Network Design Demonstration nnd12sd1[HDB96] for an illustration of the performance of the batch gradient descent algorithm.

2.7.9 Batch Gradient Descent With Momentum (TRAINGDM).

The batch form of gradient descent with momentum is invoked using the training function traingdm. This algorithm is equivalent to learngdm, with two exceptions. First, the gradient is computed by summing the gradients calculated at each training example, and the weights and biases are only updated after all training examples have been presented. Second, if the new performance function on a given iteration exceeds the performance function on a previous iteration by more than a predefined ratio max_perf_inc (typically 1.04), the new weights and biases are discarded, and the

momentum coefficient mc is set to zero. In the following code previous network is recreated and retrained using gradient descent with momentum. The training parameters for traingdm are the same as those for traingd, with the addition of the momentum factor mc and the maximum performance increase max_perf_inc. (The training parameters are reset to the default values whenever net.trainFcn is set to traingdm.) re-initialized the weights and biases before training, different mean square error is obtained than traingd. If re-initialized and train again using traingdm, Than would get yet a different mean square error. The random choice of initial weights and biases will affect the performance of the algorithm. If compare the performance of different algorithms, than should test each using several different sets of initial weights and biases[16].

2.8.1 Feature of Quantum Neural Network

A QNN with features that make it easy to model, yet powerful enough to leverage quantum physics. QNN consists of :

- known quantum algorithms and gates
- weights which can be measured for each node
- work in classical simulations of reasonable size
- able to transfer knowledge to classical systems

A QNN that operates much like a classical ANN composed of several layers of perceptrons – an input layer, one or more hidden layers and an output layer. Each layer is fully connected to the previous layer. Each hidden layer computes a weighted sum of the outputs of the previous layer. If this is sum above a threshold, the node goes high, otherwise it stays low. The output layer does the same thing as the hidden layer(s), except that it also checks its accuracy against the target output of the network. The network as a whole computes a function, by checking which output bit is high. There are no checks to make sure exactly one output is high. This allows the network to learn data sets which have one output high or binary-encoded outputs.

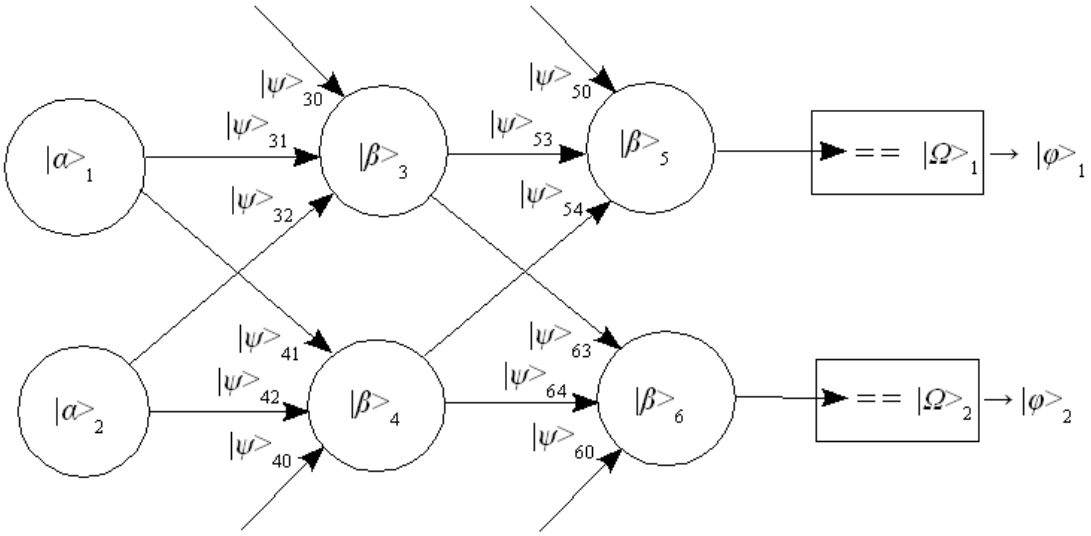


Figure 2.9: QNN

The QNN in Figure is an example of such a network, with sufficient complexity to compute the XOR function. Each input node i is represented by a register, ⟩i. The two hidden nodes compute a weighted sum of the inputs, |⟩i1 and |⟩i2, and compare the sum to a threshold weight, |⟩i0. If the weighted sum is greater than the threshold the node goes high. The ⟩k represent internal calculations that take place at each node. The output layer works similarly, taking a weighted sum of the hidden nodes and checking against a threshold.

The QNN then checks each computed output and compares it to the target output, ⟩j sending ⟩j high when they are equivalent. The performance of the network is denoted by ⟩, which is the number of computed outputs equivalent to their corresponding target output.

At the quantum gate level, the network will require O(blm + m2) gates for each node of the network. Here b is the number of bits used for floating point arithmetic in ⟩, l is the number of bits for each weight and m is the number of inputs to the node[17,18]. The overall network works as follows on a training set. In our example, the network has two input parameters, so all n training examples will have two input registers. These are represented as ⟩11 to ⟩n2. The target answers are kept in registers ⟩11 to ⟩n2. Each hidden or output node has a weight vector, represented by |›i, each vector containing weights for each of its inputs. After classifying a training

example, the registers $\rangle_1$ and $\rangle_2$ reflect the networks ability to classify that the training example. As a simple measure of performance, and increment $\rangle_i$ by the sum of all $\rangle_i$. When all training examples have

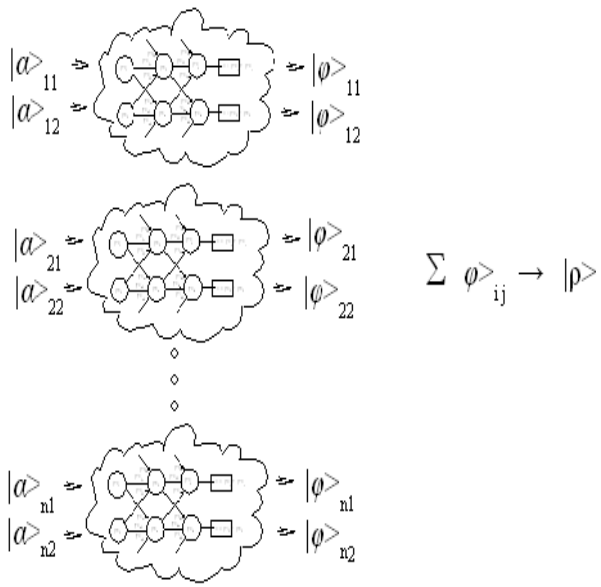


Figure 2.10: QNN Trainingf

2.8.2 Comparesion the classical and quantum manifestations :

Comparesionthe classical and quantum manifestations of the main points of importance in information theory [19]

Table 2.2: Comparesion the classical and quantum manifestations

	Classical	Quantum
basic information unit	bit: {0,1}	qubit: $\alpha\|0\rangle+\beta\|1\rangle$ (superposition principle)
Dynamics	Deterministic (causal)	deterministic (unitary evolution)
measurements	do not influence system	effect the system (uncertainty principle + collapse)

2.9 State Spaces and Bra/Ket Notation: The state space of a quantum system, consisting of the positions, momentums, polarizations, spins, and so on of the various particles, is modeled by a Hilbert space of wave functions. For quantum computing need only deal with finite quantum systems and it suffices to consider finite dimensional complex vector spaces with an inner product that are spanned by abstract wave functions such as $\rangle$.

Quantum state spaces and the transformations acting on them can be described in terms of vectors and matrices or in the more compact bra/ket notation invented by Dirac [1958]. Kets like $|x\rangle$ denote column vectors and are typically used to describe quantum states. The matching bra, $\langle x|$, denotes the conjugate transpose of $|x\rangle$. For example, the orthonormal basis $\{|0\rangle, |1\rangle\}$ can be expressed as $\{(1, 0)^T, (0, 1)^T\}$. Any complex linear combination of $|0\rangle$ and $|1\rangle$, $a|0\rangle + b|1\rangle$, can be written $(a, b)^T$. The choice of the order of the basis vectors is arbitrary. For example, representing $|0\rangle$ as $(0, 1)^T$ and $|1\rangle$ as $(1, 0)^T$ would be fine as long as this is done consistently. Combining $\langle x|$ and $|y\rangle$ as in $\langle x|y\rangle$, also written as $\langle x|y\rangle$, denotes the inner product of the two vectors. For instance, since $|0\rangle$ is a unit vector and the given vector is $\langle 0|0\rangle = 1$ and since $|0\rangle$ and $|1\rangle$ are orthogonal then $\langle 0|1\rangle = 0$

The notation $|x\rangle\langle y|$ is the outer product of $|x\rangle$ and $\langle y|$. For example, $|0\rangle\langle 1|$ is the transformation that maps $|1\rangle$ to $|0\rangle$ and $|0\rangle$ to $(0, 0)^T$, since

$$|0\rangle\langle 1|1\rangle = |0\rangle\langle 1|1\rangle = |0\rangle$$

$$|0\rangle\langle 1|0\rangle = |0\rangle\langle 1|0\rangle = 0|0\rangle$$

Equivalently, $|0\rangle\langle 1|$ can be written in matrix form, where $|0\rangle = (1, 0)^T$, $\langle 0| = (1, 0)$, $|1\rangle = (0, 1)^T$, and $\langle 1| = (0, 1)$. Then

$$|0\rangle\langle 1| = \begin{pmatrix} 1 \\ 0 \end{pmatrix} (0, 1) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

This notation gives a convenient way of specifying transformations on quantum states in terms of what happens to the basis vectors the transformation that exchanges $|0\rangle$ and $|1\rangle$ is given by the matrix .

$$X = |0\rangle\langle 1| + |1\rangle\langle 0|$$

The slightly more intuitive notation which explicitly specifies the result of a transformation on the basis vectors[20].

$$X: |0\rangle \quad |1\rangle,$$

$$|1\rangle \quad |0\rangle,$$

2.10 QUANTUM NEURON

A neuron with N inputs $|x_1\rangle$ …….. $|x_N\rangle$ is a quantum bit (qubit) of the form

$$|x_j\rangle = a_j|0\rangle + b_j|1\rangle = (a_j, b_j)^T$$

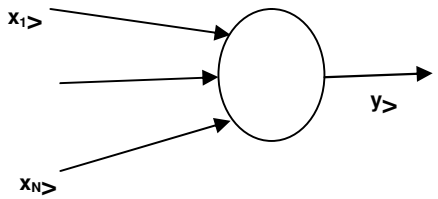


Figure 2.11: The Model of Quantum Neuron

A qubit is a unit vector in a two-dimensional complex vector space for which a particular basis, denoted by$\{|0\rangle, |1\rangle\}$ has been fixed. The orthonormal basis $\{|0\rangle, |1\rangle\}$can be expressed as$\{(1\ 0)^T,(0\ 1)^T\}$. For the purposes of quantum computation, the basis states $|0\rangle$ and $|1\rangle$ are taken to represent the classical bit values 0 and 1 respectively. Unlike the classical bit however, a qubit can he in a superposition of $|0\rangle$ and $|1\rangle$ such as (I) where and *b;* are complex numbers such that $a^2 + b^2 = 1$

Combining $\langle x_j | x_k\rangle$ as in $\langle x_j | x_k\rangle$, denotes the inner product of the two vectors, it's a scalar

$\langle 0|0\rangle = \langle 1|1\rangle = 1, \langle 0|1\rangle = \langle 1|0\rangle = 0$

The notation $|x_j\rangle\langle x_k|$ is the outer product of $\langle x_j|$ and $|x_k\rangle$, it's an operator. For example

The output **y**〉 is also a qubit derived by the rule

$$y\rangle = F^{\wedge}\sum^{N}_{j=1}w^{\wedge}\text{write}$$

where w is 2x2 matrices acting on the basis {〈 0〉, 1〉},is an operator that can be implemented by the network of quantum gates.

Consider the simplistic case with F = I being the identity operator. The output of the QN at the time t will be

$$y(t)\ \rangle = \sum w_j(t)\ \ x_j\rangle$$

In the case of classical , let us provide a quantum leaming rule

$$W_j(t+1) = w_j(t)+\ (\ \ d\rangle -\ \ y(t)\ \rangle\langle\ x_j$$

where d〉 is the desired output[21]

2.11 Quantum Complex-valued Neuron Model

Make the bit is the fundamental concept of classical computation and classical information. Quantum computation and quantum information are built upon an analogous concept, the quantum bit, or qubit for short. A qubit is a unit vector in a two-dimensional complex vector space for which a particular basis, denoted by { 0〉), | 1} has been futed. The orthonormal basis | 0 and 1〉 may correspond to the 〉 and 〉 state of the simplest spin 1/2 system [20].

connection between the neuron state and the quantum state by considering that

- The firing neuron state is the qubit state 1〉
- No firing neuron state is | 0 .
- The arbitrary neuron state is the coherent superposition of the two states

2.12 Node Operation

Each node has a simple operation using two equations below. The net input-of node *J* will be calculated as shown in equation

$$net_j^m = \sum w_{ji}^m y_i^{m-1} = \sum \begin{cases} \{(Re[w_{ji}^m]Re[y_i^{m-1}] - Im[w_{ji}^m]Im[y_i^{m-1}])\} + \\ \\ i\{(Re[w_{ji}^m]Re[y_i^{m-1}] - Im[w_{ji}^m]Im[y_i^{m-1}])\} \end{cases}$$

The signal y_i^{m-1}' is an output signal from node i of the layer m - 1 . There are $N^{m=1}$ signal coming to node j, and w_{ji}^m: is the weight between node j of layer m , and node i of the layer m-1 . The output y_j^m of node j is related to its activation as described in equation.

$$y_j^m = fc.(net_j^m)$$

$$fc = fr(Re[net_j^m]) + ifR(Im[net_j^m])$$

$$\text{Where } net_j^m = Re[net_j^m] + iIm[net_j^m],\ i \text{ denote } \sqrt{-1} \text{ and } fR(u) = 1/(1-e^{-u})$$

This means the real and imaginary parts of the output of a neuron refer to sigmoid functions of the real part, Re[net,"] and the imaginary part, Im[net^j_m]of the net input, netJ" , to a neuron, respectively. The learning rule was obtained using a steepest descent method.

2.13 Mapping a Number into Quantum State

To represent data from real world problem which generally in real number and binary data to quantum state, The mapping function described in equation and for binary number and real number respectively, where b is binary and r is real number.

$$f(\ _b) = \cos(\ _b) + i\sin(\ _b)$$

Where $_b =$ / 2, and b is binary value.

$$f(\ _r) = \cos(\ _r) + i\sin(\ _r)$$

Where r = (/2).sig(r); sig(r) = $1/(1+e^{-r})$ and r is real value.

2.14 Quantum State Output Interpretation

The quantum state neuron output based on the probability amplitude according to the decoherence of the quantum state.The output interpretation as described in equation

$y_{pj} = Re[y_{pj}] + Im[y_{pj}]$

$= \cos(\ y_{pj}) + \sin(\ y_{pj})$ and $\cos^2(\ y_{pj}) + \sin^2(\ y_{pj}) = 1$

From the quantum state mapping scheme using sigmoid function the range of value is 0 to 1, then the real world value of y_{pj} , or output O_{pj}, as described in equ. [22]

$$O_{pj} = \sin^2(\ y_{pj}) = Im[y_{pj}]^2$$

Chapter 3

Problem statement

3.1 Objective

In the second chapter the Neural network, Quantum theory and Quantum neural network have been described, in which backpropegation algorithm of Neural network and features of the Quantum neural network has been explained. Discussed the reason to develop the quantum neural network and all features of the QNN. Quantum neural network based on the complex valued function and many transfer function like as logsig, tansig with different types of training function such as traingda, trainrp, traincgf, traincg etc. of the backpropagation algorithm has studied during the research work.

3.2 Methodology

The results in the neural network are defined in the classical form where as in the quantum neural network the results can be defined by using the complex valued function in the approximation. In other words the result can be defined in the accurate form because the quantum neural network is the generalized form of the classical neural networks.

Using the applications of quantum neural network to classify the vehicle classification in the MATLAB enviournment and apply the complex function at the diverse training function mention in the chapter 4.

The following code will recreate the earlier network, and then train it using batch steepest descent.

3.2.1 Gradient Descent (LEARDGD).

```
net.biases{1,1}.learnFcn = 'learngd';
net.biases{2,1}.learnFcn = 'learngd';
net.layerWeights{2,1}.learnFcn = 'learngd';
```

```
net.inputWeights{1,1}.learnFcn = 'learngd';
```

3.2.2 Gradient Descent With Momentum (LEARDGDM).

```
net.biases{1,1}.learnFcn = 'learngdm';
net.biases{2,1}.learnFcn = 'learngdm';
net.layerWeights{2,1}.learnFcn = 'learngdm';
inputWeights{1,1}.learnFcn = 'learngdm';
[net,a,e]=adapt(net,p,t);
```

Chapter 4
Solution of the problem

4.1 Vehicle classification

Now illustrate the vehicle classification. In vehicle classification, a three layered feed forward quantum neural network is constructed. There are four kinds vehicles (saloon car, microbus, truck and motor coach) need to be recognized. The input vectors are characteristic parameters of each vehicle are given by vehicle's response curve.

4.2 Characteristic parameters of each vehicle

Table 4: parameter of each vehicle

	Peak value	**Width**	**Up–down proportion**	**Left-right proportion**	**Peak value proportion**
T1	**80**	**20**	**31**	**88**	**0**
T2	**20**	**27**	**60**	**63**	**21**
T3	**90**	**90**	**15**	**15**	**57**
T4	**73**	**62**	**75**	**71**	**88**

The object vectors in the QNN are defined as T1=[1 0 0 0] T,T2=[0 1 0 0] T,T3=[0 0 1 0] T and T4=[0 0 0 1] T, which represent saloon car, microbus, truck and motor coach respectively. Apply all the training function at the back propagation training algorithm and find the results using the parameter (peak value, width, up –down proportion, left right proportion, peak value proportion) of the saloon car, microbus, truck and motor coach.

Before training, input data also need to be performed binary conversion. The method is the same as above paragraphs. Compare the present method of QNN with other

traditional neural networks, Back Propagation neural networks (BP)quantum neural network .

4.3 Training Algorithm

The parameters which are shown in Table 4 applied at the back propagation algorithm in Matlab tool and then the transfer function like logsig, tansig etc. and the training function traingda, traingfa, traincgp, etc. have been for training the quantum neural network models.

Algorithm:

Step 1: Initialize the inputs.

Step 2: Define the value of theta.

Step 3: Apply the value of theta in the complex function.

f = cos +isin ;

Step 4: Define the value of the output.

Step 5: Simulate the network (before the training).

Step 6: Declare the all training parameters

(a)network.trainParam.show ;

(b)network.trainParam.lr ;

(c)network.trainParam.lr_inc;

(d)network.trainParam.epochs;

(e)network.trainParam.goal

Step 7: Train the network

Step 8: Simulate the network after the training

Step 9: End

Chapter 5

Results

The results of the parameters before training, performance, after training shown in the figures:

5.1 FOR CAR

5.1.1 Performance by using the TRAINGD

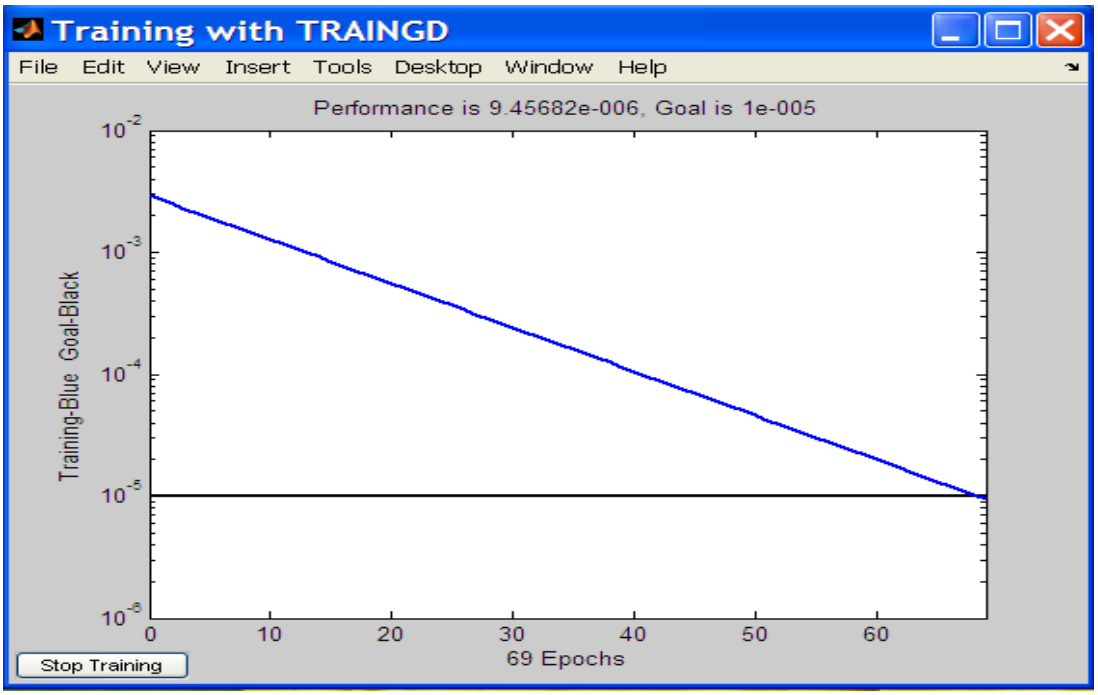


Figure 5.1 Result of training using Traingd

TRAINGD, Epoch 0/500, MSE 0.28395/1e-005, Gradient 0.719336/1e-010

TRAINGD, Epoch 5/500, MSE 0.184281/1e-005, Gradient 0.543411/1e-010

TRAINGD, Epoch 10/500, MSE 0.122525/1e-005, Gradient 0.443735/1e-010

TRAINGD, Epoch 15/500, MSE 0.0832466/1e-005, Gradient 0.37715/1e-010

TRAINGD, Epoch 20/500, MSE 0.0574476/1e-005, Gradient 0.324898/1e-010

TRAINGD, Epoch 25/500, MSE 0.0400416/1e-005, Gradient 0.280452/1e-010

TRAINGD, Epoch 30/500, MSE 0.0280713/1e-005, Gradient 0.241581/1e-010

TRAINGD, Epoch 35/500, MSE 0.0197352/1e-005, Gradient 0.207396/1e-010

TRAINGD, Epoch 40/500, MSE 0.0138857/1e-005, Gradient 0.177408/1e-010

TRAINGD, Epoch 45/500, MSE 0.00976429/1e-005, Gradient 0.151233/1e-010

TRAINGD, Epoch 50/500, MSE 0.00685552/1e-005, Gradient 0.12851/1e-010

TRAINGD, Epoch 55/500, MSE 0.0048026/1e-005, Gradient 0.108885/1e-010

TRAINGD, Epoch 60/500, MSE 0.00335538/1e-005, Gradient 0.092017/1e-010

TRAINGD, Epoch 65/500, MSE 0.00233713/1e-005, Gradient 0.0775782/1e-010

TRAINGD, Epoch 70/500, MSE 0.0016225/1e-005, Gradient 0.0652645/1e-010

TRAINGD, Epoch 75/500, MSE 0.00112243/1e-005, Gradient 0.0547973/1e-010

TRAINGD, Epoch 80/500, MSE 0.00077361/1e-005, Gradient 0.0459256/1e-010

TRAINGD, Epoch 85/500, MSE 0.000531138/1e-005, Gradient 0.0384254/1e-010

TRAINGD, Epoch 90/500, MSE 0.0003632/1e-005, Gradient 0.0320995/1e-010

TRAINGD, Epoch 95/500, MSE 0.000247324/1e-005, Gradient 0.0267752/1e-010

TRAINGD, Epoch 100/500, MSE 0.000167683/1e-005, Gradient 0.0223026/1e-010

TRAINGD, Epoch 105/500, MSE 0.000113169/1e-005, Gradient 0.0185519/1e-010

TRAINGD, Epoch 110/500, MSE 7.60114e-005/1e-005, Gradient 0.015412/1e-010

TRAINGD, Epoch 115/500, MSE 5.07945e-005/1e-005, Gradient 0.0127873/1e-010

TRAINGD, Epoch 120/500, MSE 3.37592e-005/1e-005, Gradient 0.0105965/1e-010

TRAINGD, Epoch 125/500, MSE 2.23062e-005/1e-005, Gradient 0.00877036/1e-010

TRAINGD, Epoch 130/500, MSE 1.4645e-005/1e-005, Gradient 0.00725027/1e-010

TRAINGD, Epoch 135/500, MSE 9.54777e-006/1e-005, Gradient 0.00598653/1e-010

TRAINGD, Performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using traingd trainfunction. In this performance goal of the network has been achived.

5.1.2Performance by using the TRAINOSS

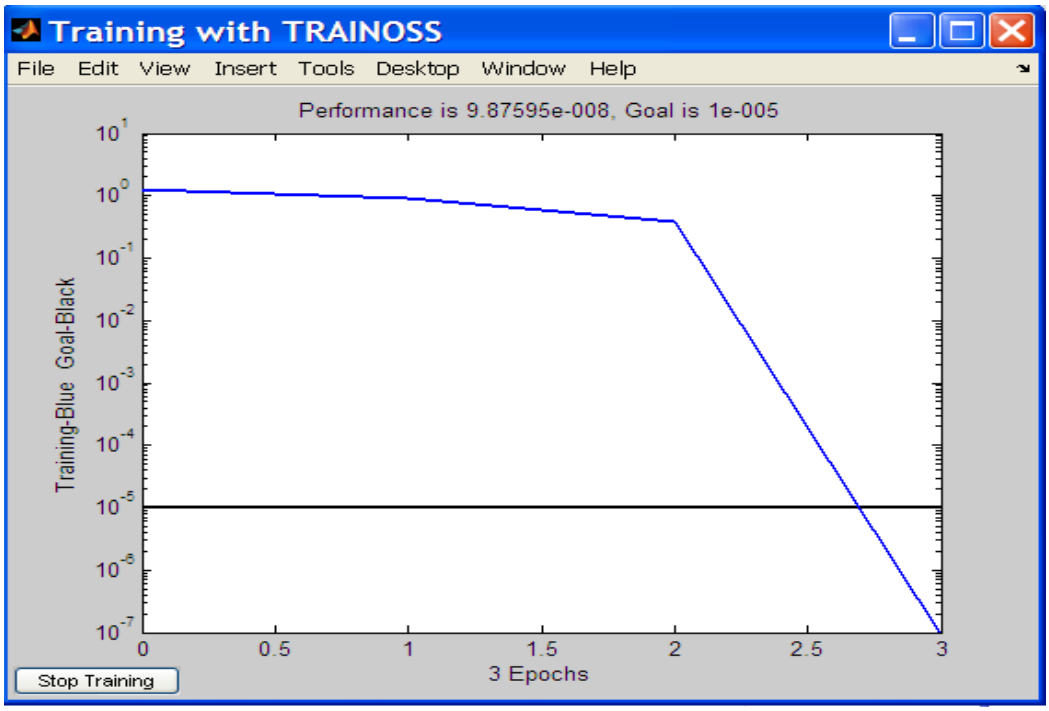


Figure 5.2 Result of training using Trainnoss

TRAINOSS-srchbac, Epoch 0/300, MSE 1.92022e-005/1e-005, Gradient 0.0589612/1e-006

TRAINOSS-srchbac, Epoch 2/300, MSE 6.98553e-010/1e-005, Gradient 5.5309e-005/1e-006

TRAINOSS, Performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using trainoss trainfunction. In this performance goal of the network has been achived.

5.1.3 Performance by using the TRAINGDM

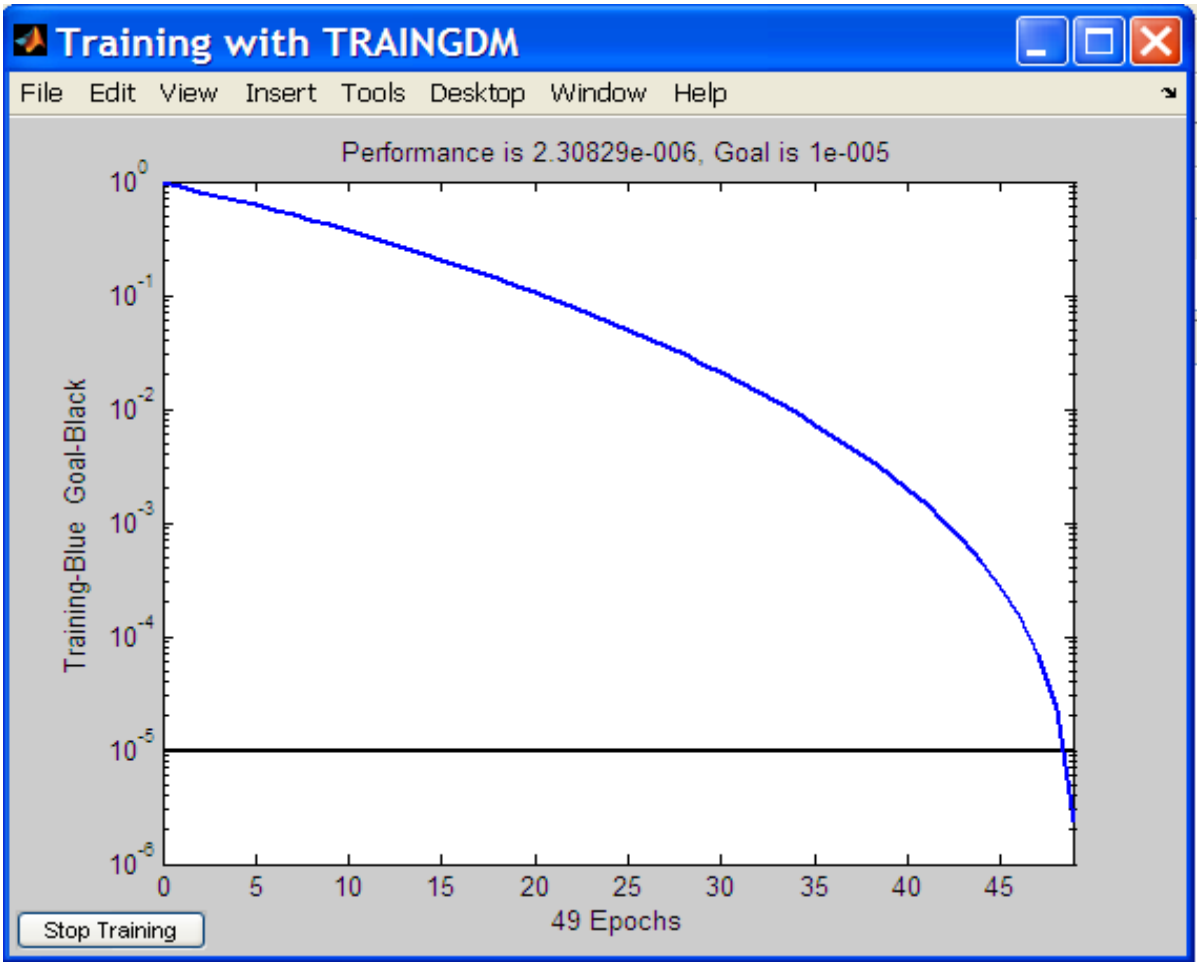


Figure 5.3 Result of training using Traingdm

TRAINGDM, Epoch 0/300, MSE 2.16861/1e-005, Gradient 4.20248/1e-010

TRAINGDM, Epoch 48/300, MSE 5.21448e-009/1e-005, Gradient 0.000485771/1e-010

TRAINGDM, Performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using traingdm trainfunction. In this performance goal of the network has been achived

5.1.4 Performance by using the TRAINGDA

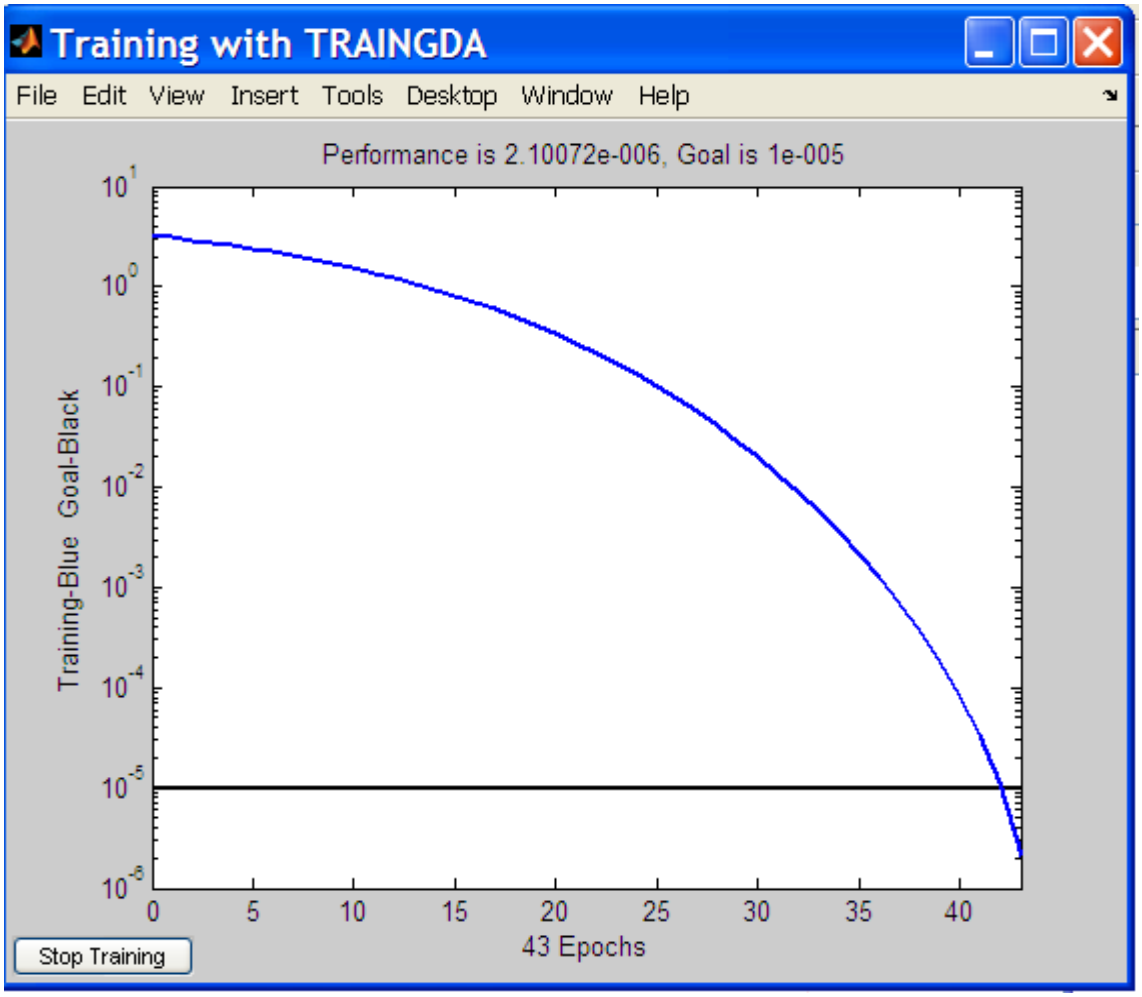


Figure 5.4 Result of training using Traingda

TRAINGDA, Epoch 0/300, MSE 3.48013/1e-005, Gradient 5.32155/1e-006

TRAINGDA, Epoch 43/300, MSE 8.92218e-006/1e-005, Gradient 0.00851981/1e-006

TRAINGDA, Performance goal met

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using traingda trainfunction. In this performance goal of the network has been achived.

5.1.5 Performance by using the TRAINSCG

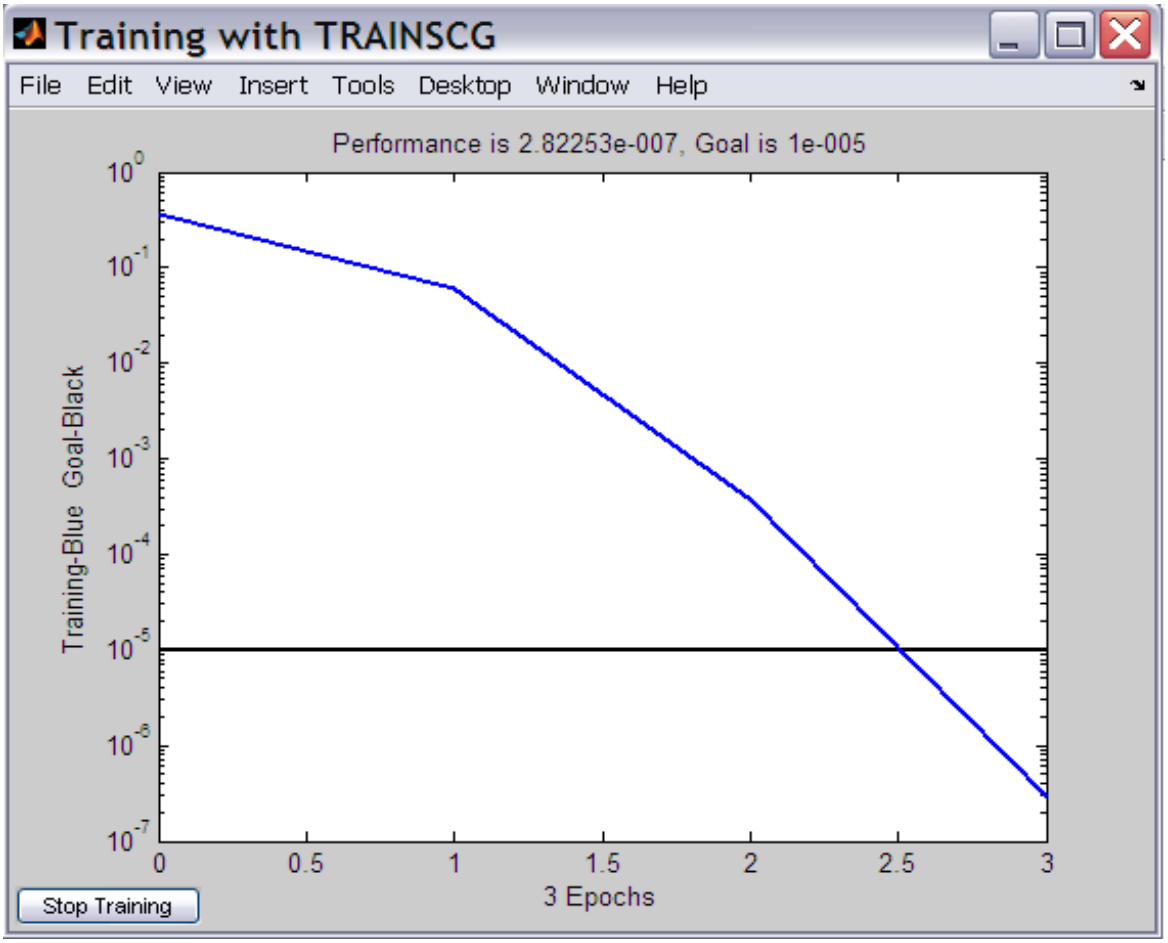


Figure 5.5 Result of training using Trainscg

TRAINSCG, Epoch 0/300, MSE 0.371405/1e-005, Gradient 2.72375/1e-006

TRAINSCG, Epoch 3/300, MSE 2.82253e-007/1e-005, Gradient 0.00193582/1e-006

TRAINSCG, Performance goal met

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using trainscg trainfunction. In this performance goal of the network has been achived.

5.1.6Performance by using the TRAINCGB

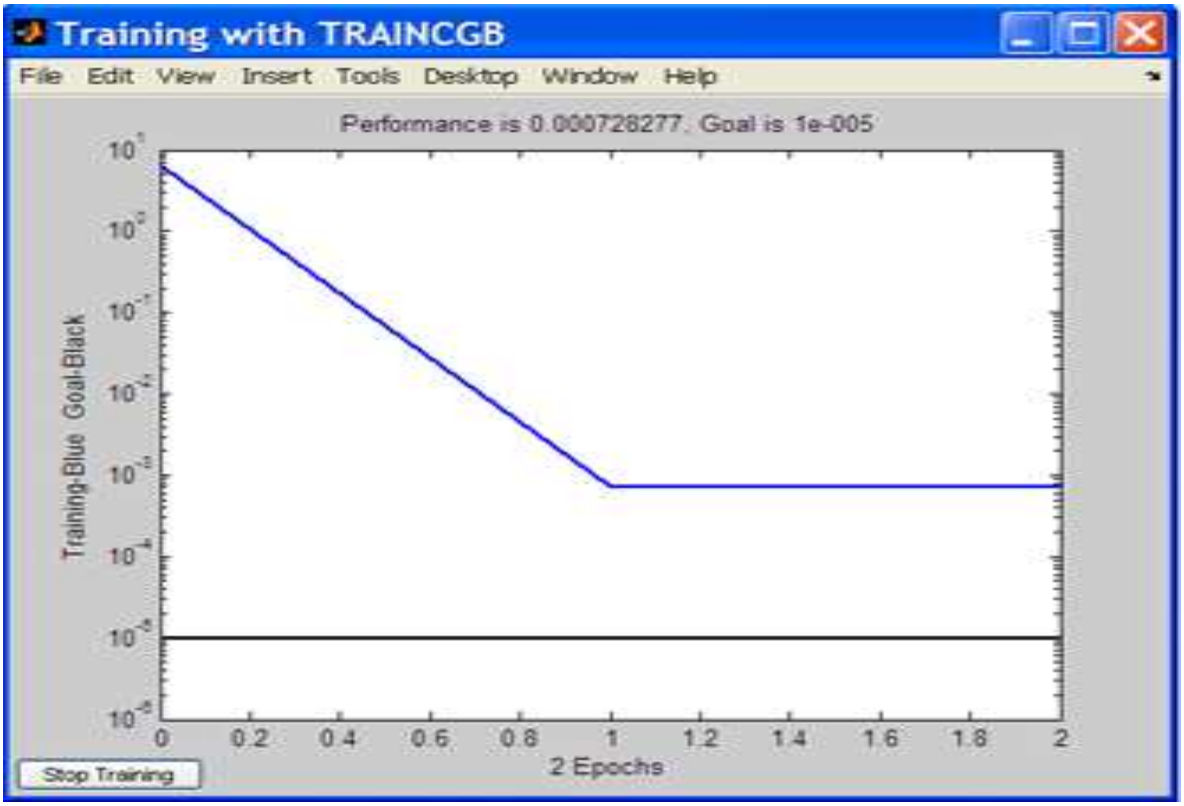


Figure 5.6 Result of training using Traincgb

TRAINCGB-srchcha, Epoch 2/200, MSE 0.000728277/1e-005, Gradient 0.423724/1e-006

TRAINCGB, Minimum step size reached, performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using traincgb trainfunction. In this performance goal of the network has been achived.

5.1.7Performance by using the TRAINLM

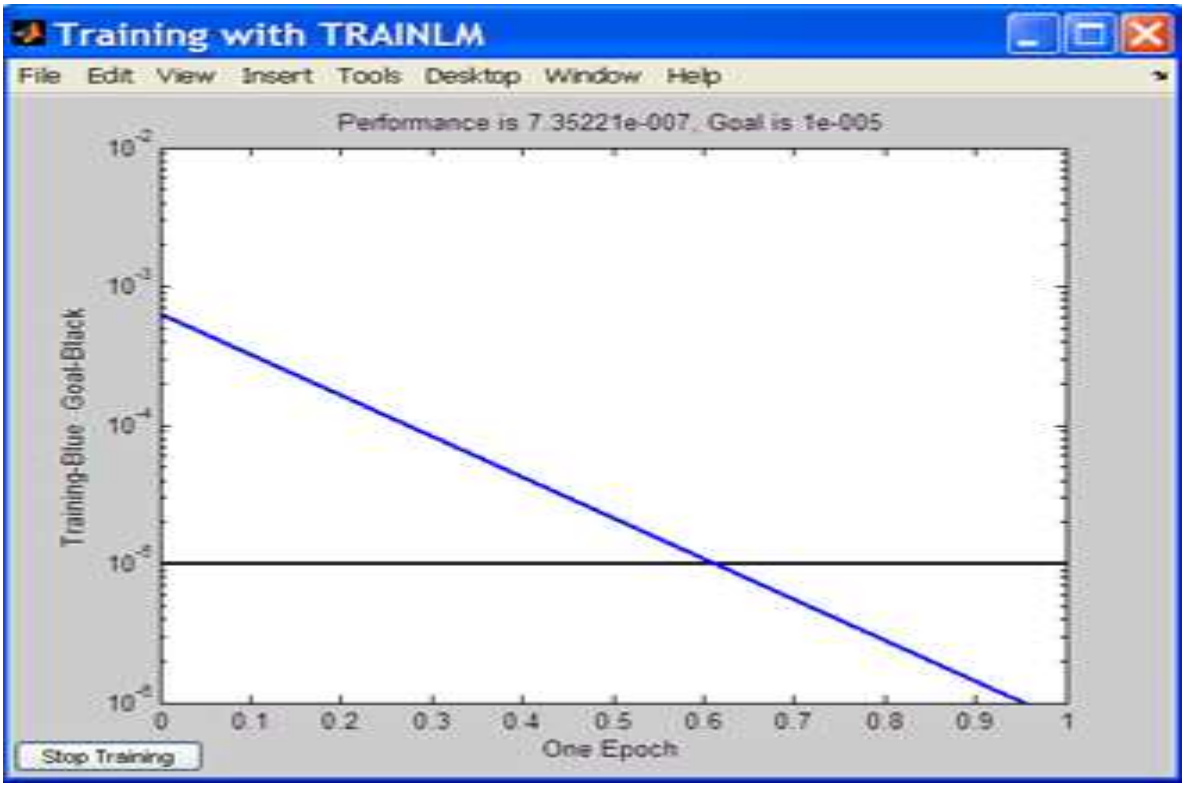


Figure 5.7 Result of training using Trainlm

TRAINLM, Epoch 0/300, MSE 0.000640329/1e-005, Gradient 0.0554796/1e-010

TRAINLM, Epoch 1/300, MSE 7.35221e/1e-005, Gradient 0.00256651/1e-010

TRAINLM, Performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using trainlm trainfunction. In this performance goal of the network has been achived.

5.1.8 Performance by using the TRAIBFG

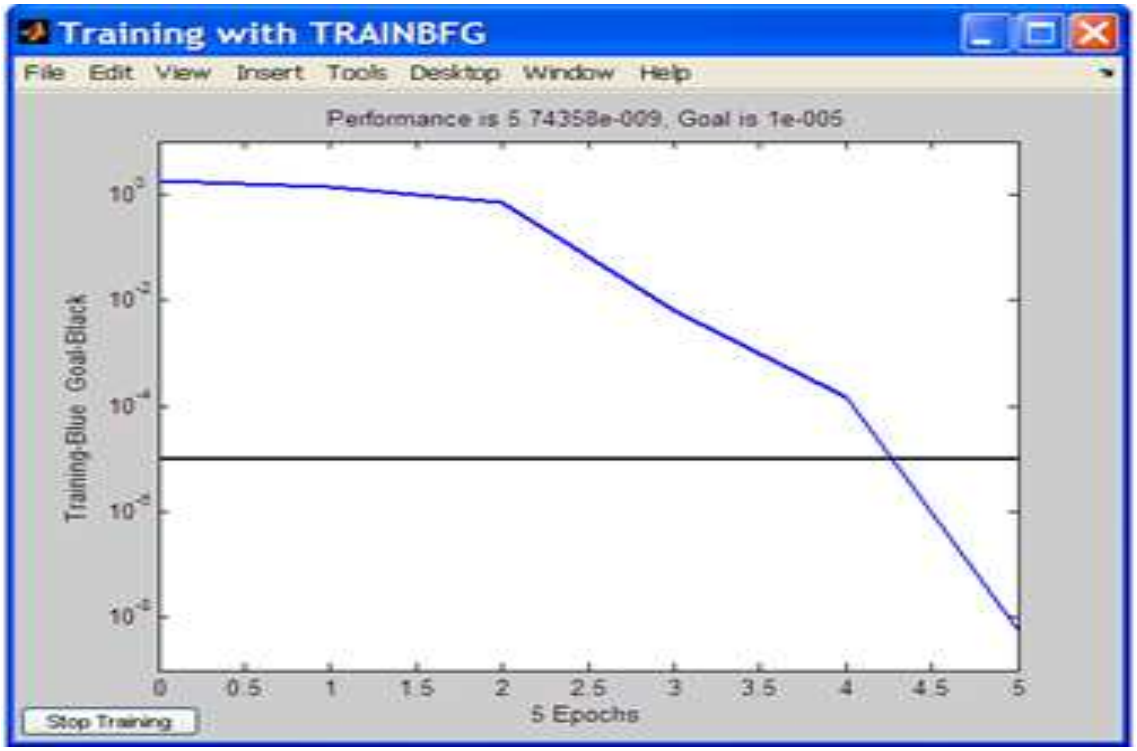


Figure 5.8 Result of training using Trainbfg

TRAINBFG-srchbac, Epoch 0/200, MSE 1.78646/1e¯005, Gradient 4.53691/1e-006

TRAINBFG, Epoch 5/200, MSE 5.74358e¯009/1e-005, Gradient 0.0000349266/1e-006

TRAINBFG, Performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using trainbfg train function. In this performance goal of the network has been achived.

5.2 For Truck

5.2.1 Performance by using the TRAINGD

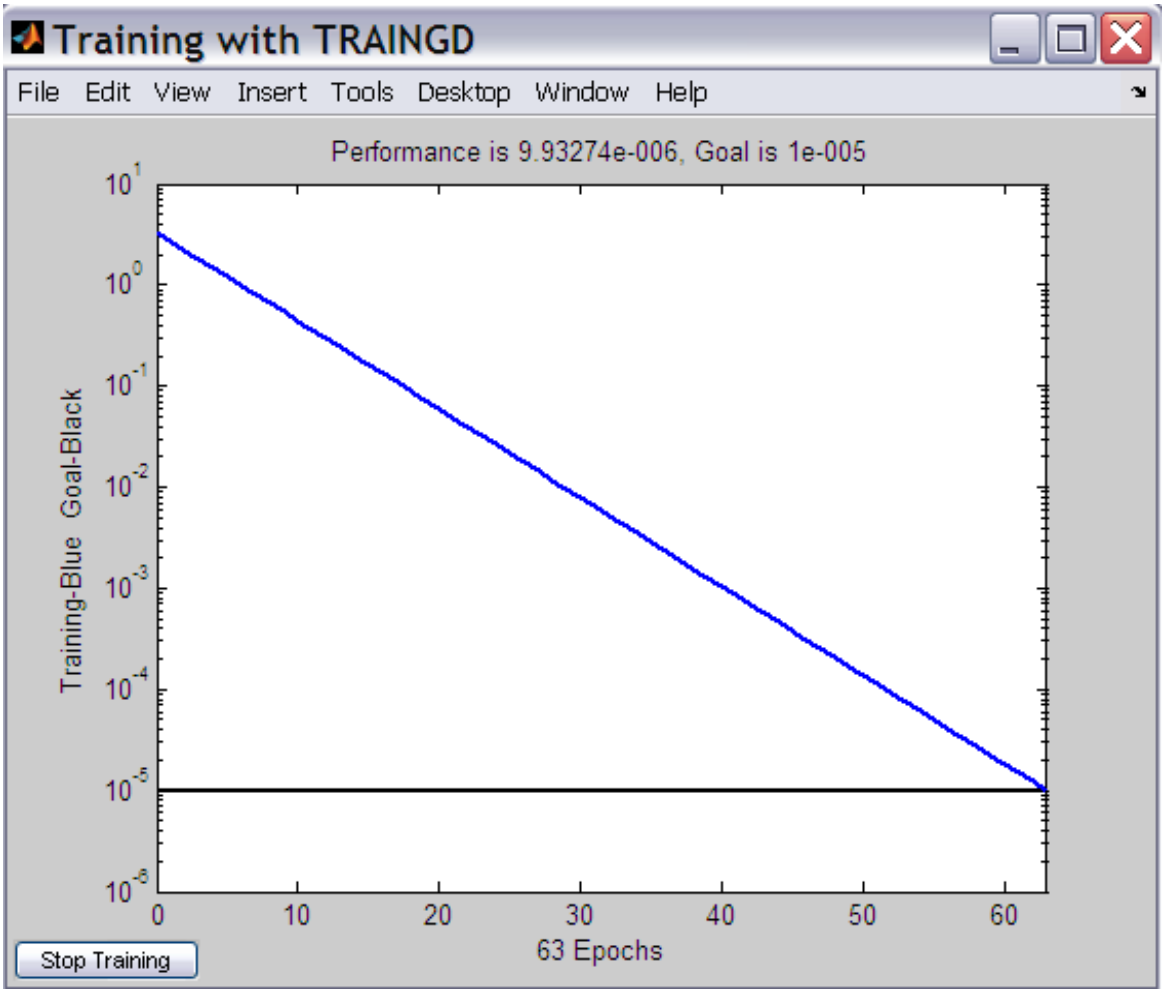


Figure 5.9 Result of training using Traingd

TRAINGD, Epoch 0/500, MSE 3.28368/1e-005, Gradient 7.89485/1e-010

TRAINGD, Epoch 50/500, MSE 0.000137723/1e-005, Gradient 0.0513003/1e-010

TRAINGD, Epoch 63/500, MSE 9.93274e-006/1e-005, Gradient 0.0137802/1e-010

TRAINGD, Performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using traingd trainfunction. In this performance goal of the network has been achived

5.2.2 Performance by using the TRAINOSS

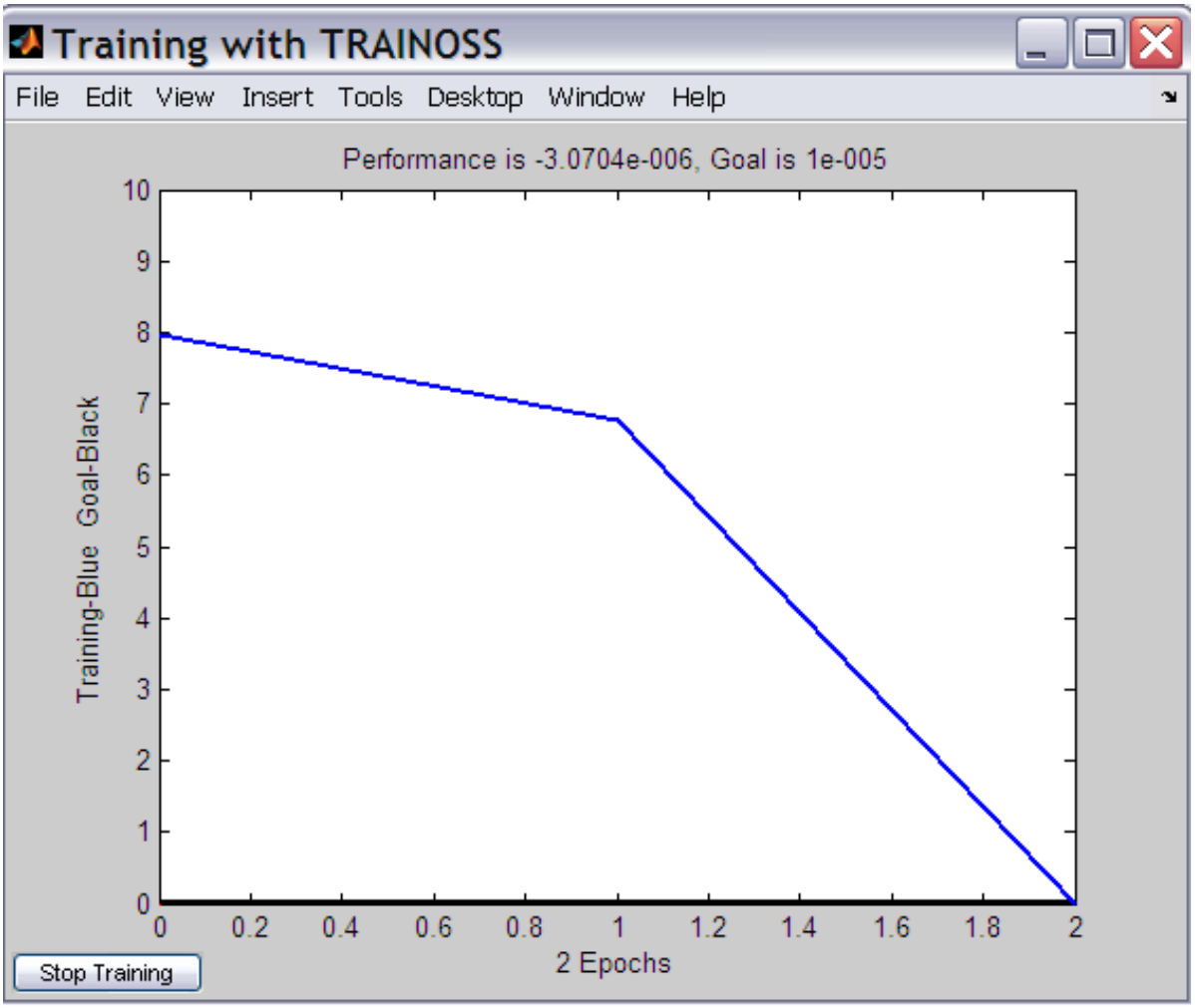


Figure 5.10 Result of training using Trainnoss

TRAINOSS-srchbac, Epoch 0/300, MSE 7.96276/1e-005, Gradient 17.6154/1e-006
TRAINOSS-srchbac, Epoch 2/300, MSE -3.0704e-006/1e-005, Gradient 0.00515972/1e-006
TRAINOSS, Performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using trainoss trainfunction. In this performance goal of the network has been achived

5.2.3 Performance by using the TRAINGDM

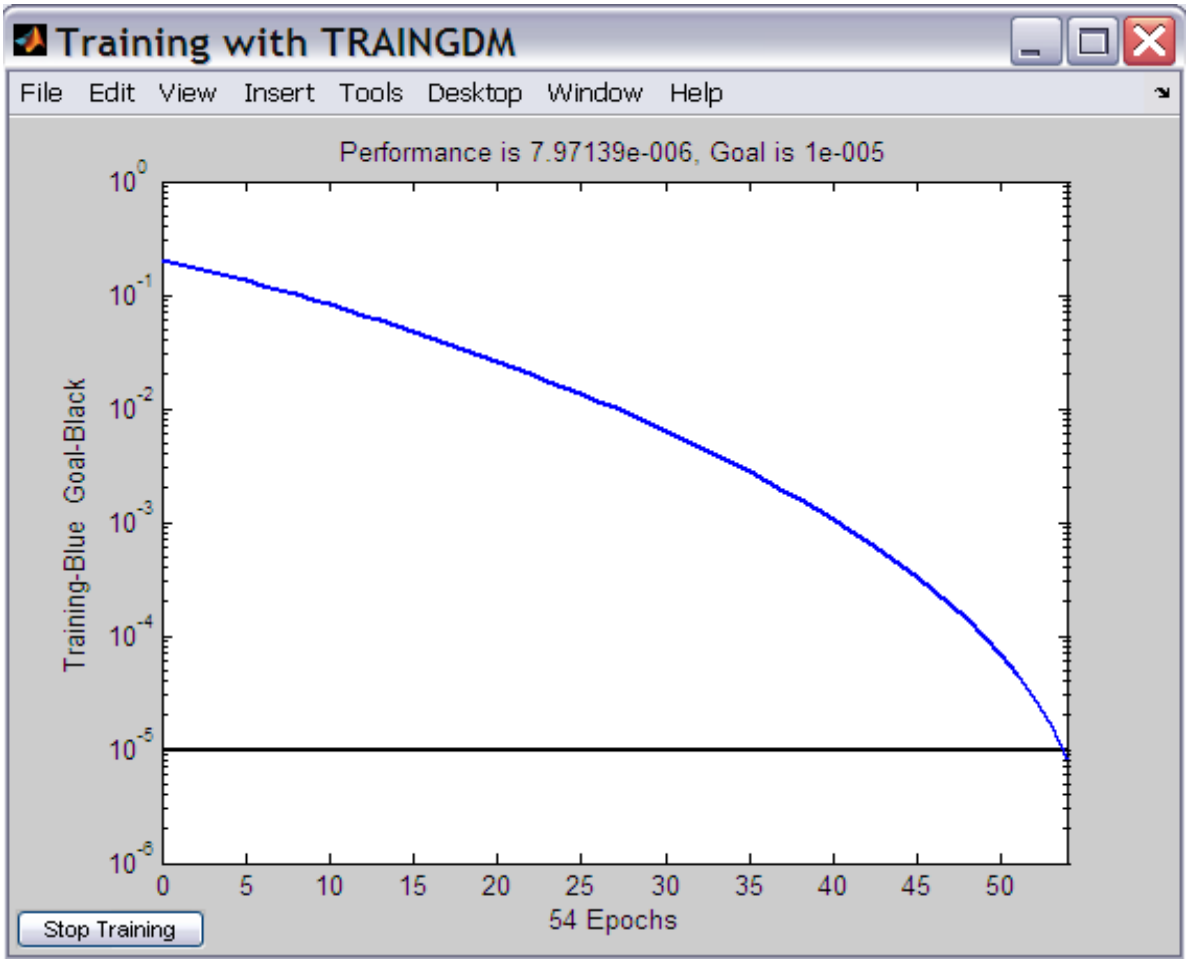


Figure 5.11 Result of training using Traingdm

TRAINGDM, Epoch 0/300, MSE 0.201638/1e-005, Gradient 1.20898/1e-010

TRAINGDM, Epoch 50/300, MSE 6.88355e-005/1e-005, Gradient 0.0224806/1e-010

TRAINGDM, Epoch 54/300, MSE 7.97139e-006/1e-005, Gradient 0.00816406/1e-010

TRAINGDM, Performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using traingdm trainfunction. In this performance goal of the network has been achived

5.2.4 Performance by using the TRAINCGB

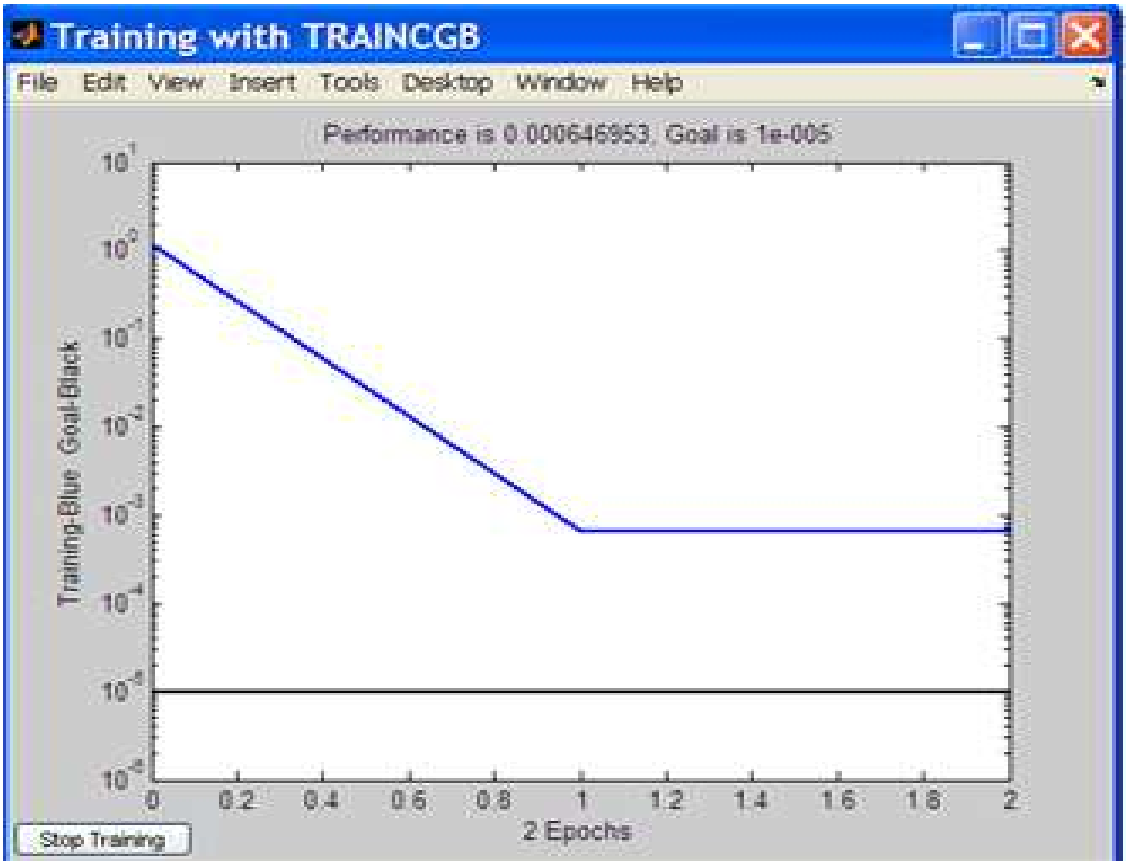


Figure 5.12 Result of training using Traincgb

TRAINCGB-srchcha, Epoch 0/300, MSE 0.000646953/1e-005, Gradient 0.0667302/1e-006

TRAINCGB, Minimum step size reached,performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using traincgb train function. In this performance goal of the network has been achived

5.2.5Performance by using the TRAINLM

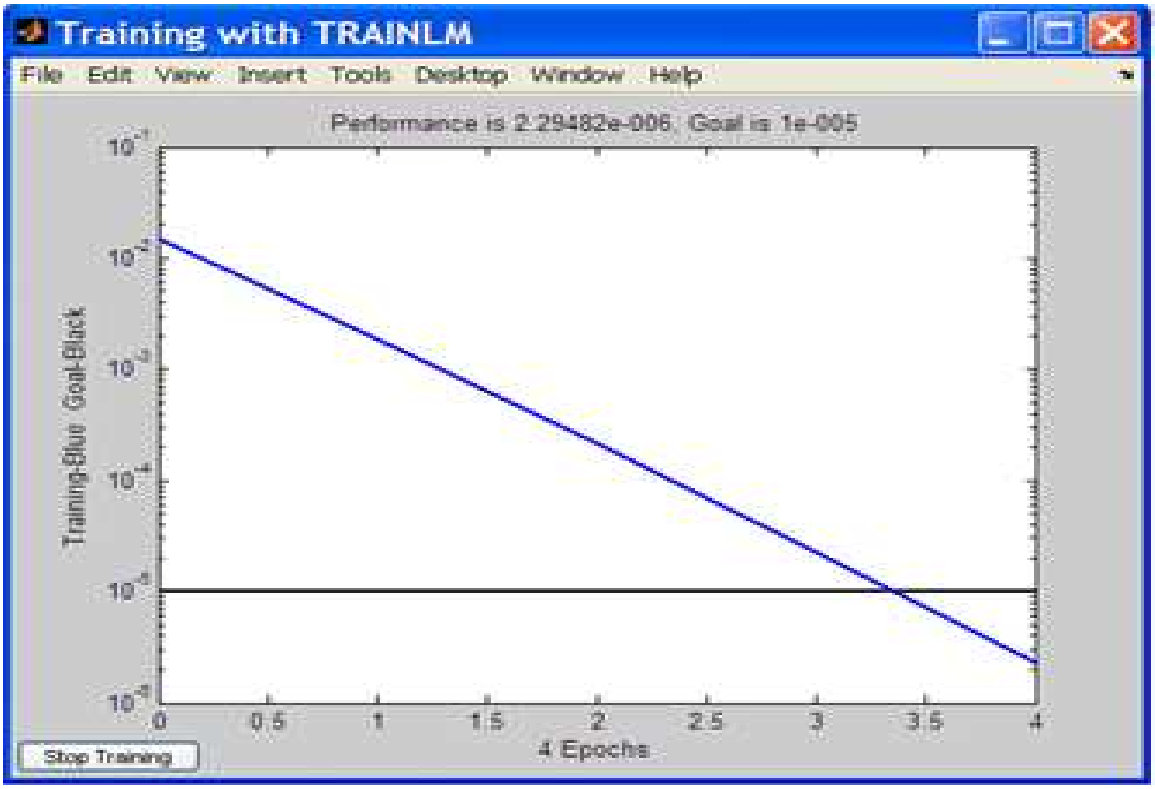


Figure 5.13 Result of training using Trainlm

TRAINLM, Epoch 0/300, MSE 0.0151555/1e-005, Gradient 0.139293/1e-010

TRAINLM, Epoch 4/300, MSE 2.29482e-006/1e-005, Gradient 0.00170844/1e-010

TRAINLM, Performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using trainlm train function. In this performance goal of the network has been achived.

5.3 For Microbus

5.3.1 Performance by using the TRAINGD

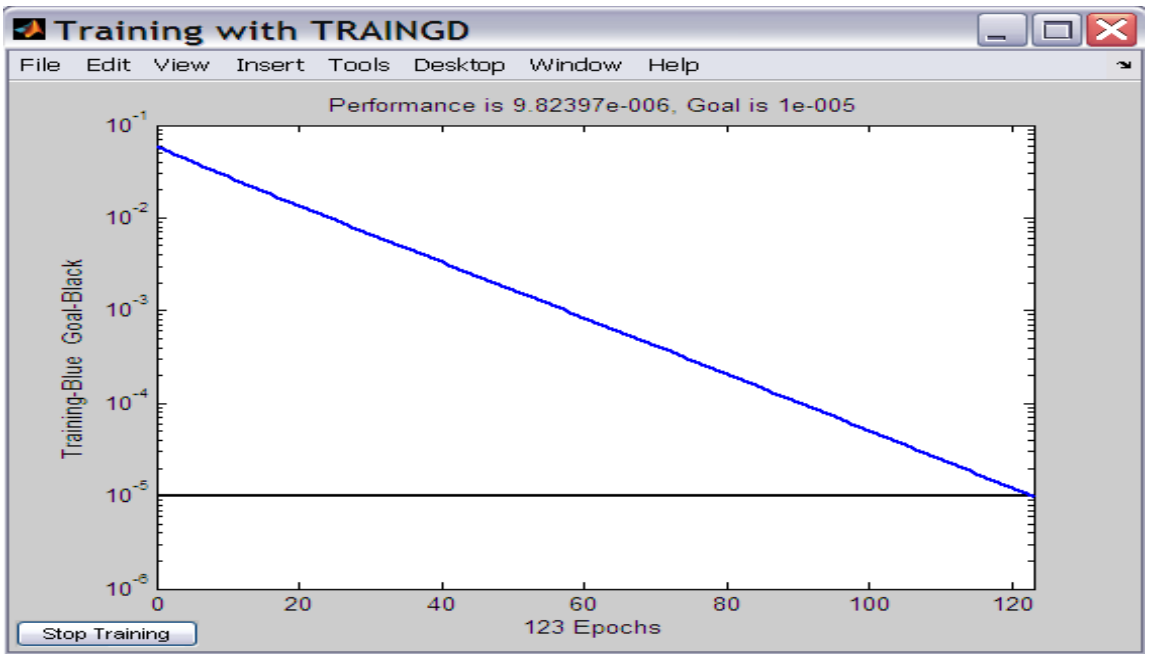


Figure 5.14 Result of training using Traingd

TRAINGD, Epoch 0/500, MSE 0.0590626/1e-005, Gradient 0.718561/1e-010

TRAINGD, Epoch 5/500, MSE 0.0403745/1e-005, Gradient 0.592465/1e-010

TRAINGD, Epoch 10/500, MSE 0.0278572/1e-005, Gradient 0.490343/1e-010

TRAINGD, Epoch 15/500, MSE 0.019365/1e-005, Gradient 0.407103/1e-010

TRAINGD, Epoch 20/500, MSE 0.0135396/1e-005, Gradient 0.338858/1e-010

TRAINGD, Epoch 25/500, MSE 0.00950731/1e-005, Gradient 0.282625/1e-010

TRAINGD, Epoch 30/500, MSE 0.00669628/1e-005, Gradient 0.236096/1e-010

TRAINGD, Epoch 35/500, MSE 0.00472616/1e-005, Gradient 0.197464/1e-010

TRAINGD, Epoch 40/500, MSE 0.00333999/1e-005, Gradient 0.165303/1e-010

TRAINGD, Epoch 45/500, MSE 0.00236204/1e-005, Gradient 0.138471/1e-010

TRAINGD, Epoch 50/500, MSE 0.00167084/1e-005, Gradient 0.116048/1e-010

TRAINGD, Epoch 55/500, MSE 0.00118178/1e-005, Gradient 0.0972865/1e-010

TRAINGD, Epoch 60/500, MSE 0.000835575/1e-005, Gradient 0.081573/1e-010

TRAINGD, Epoch 65/500, MSE 0.00059046/1e-005, Gradient 0.0684036/1e-010

TRAINGD, Epoch 70/500, MSE 0.000416955/1e-005, Gradient 0.0573608/1e-010

TRAINGD, Epoch 75/500, MSE 0.000294195/1e-005, Gradient 0.0480983/1e-010

TRAINGD, Epoch 80/500, MSE 0.000207391/1e-005, Gradient 0.0403274/1e-010

TRAINGD, Epoch 85/500, MSE 0.00014606/1e-005, Gradient 0.0338074/1e-010

TRAINGD, Epoch 90/500, MSE 0.000102762/1e-005, Gradient 0.0283367/1e-010

TRAINGD, Epoch 95/500, MSE 7.22247e-005/1e-005, Gradient 0.0237467/1e-010

TRAINGD, Epoch 100/500, MSE 5.07078e-005/1e-005, Gradient 0.0198959/1e-010

TRAINGD, Epoch 105/500, MSE 3.55626e-005/1e-005, Gradient 0.0166658/1e-010

TRAINGD, Epoch 110/500, MSE 2.49133e-005/1e-005, Gradient 0.0139567/1e-010

TRAINGD, Epoch 115/500, MSE 1.74335e-005/1e-005, Gradient 0.0116851/1e-010

TRAINGD, Epoch 120/500, MSE 1.21856e-005/1e-005, Gradient 0.00978068/1e-010

TRAINGD, Epoch 123/500, MSE 9.82397e-006/1e-005, Gradient 0.00878933/1e-010

TRAINGD, Performance goal met

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using traingd trainfunction. In this performance goal of the network has been achived

5.3.2 Performance by using the TRAINOSS

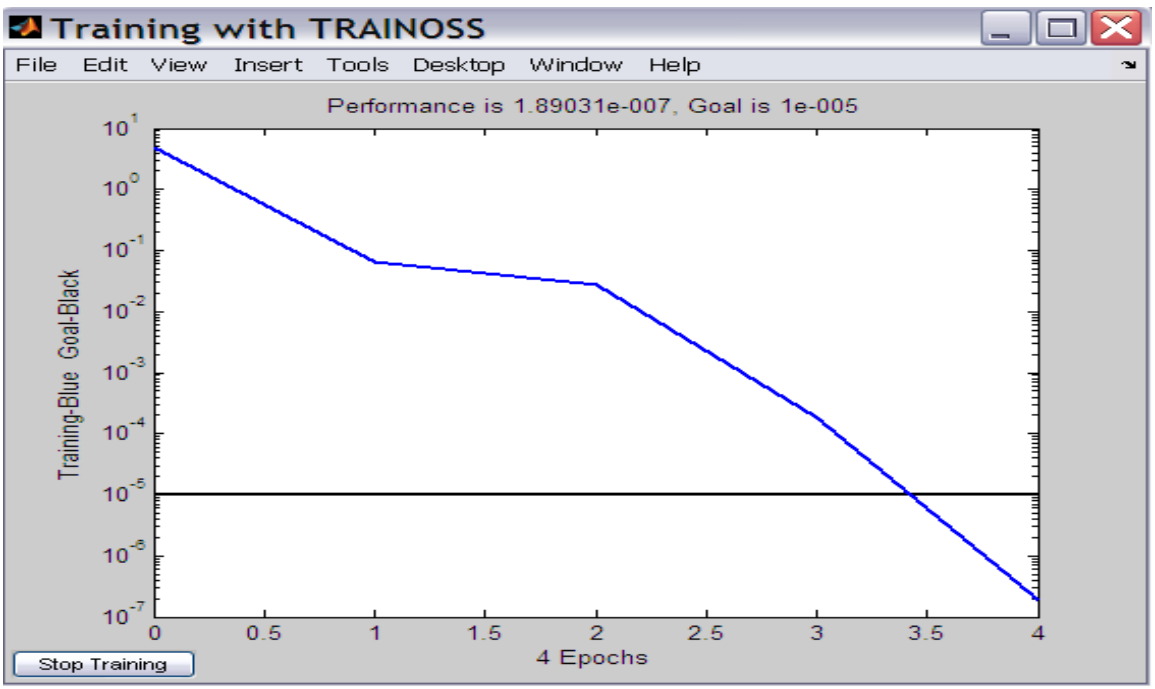


Figure 5.15 Result of training using Trainoss

TRAINOSS-srchbac, Epoch 0/500, MSE 4.88377/1e-005, Gradient 10.0651/1e-006

TRAINOSS-srchbac, Epoch 4/500, MSE 1.89031e-007/1e-005, Gradient 0.00196507/1e-006

TRAINOSS, Performance goal met

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using trainoss trainfunction. In this performance goal of the network has been achived.

5.3.3 Performance by using the TRAINGDM

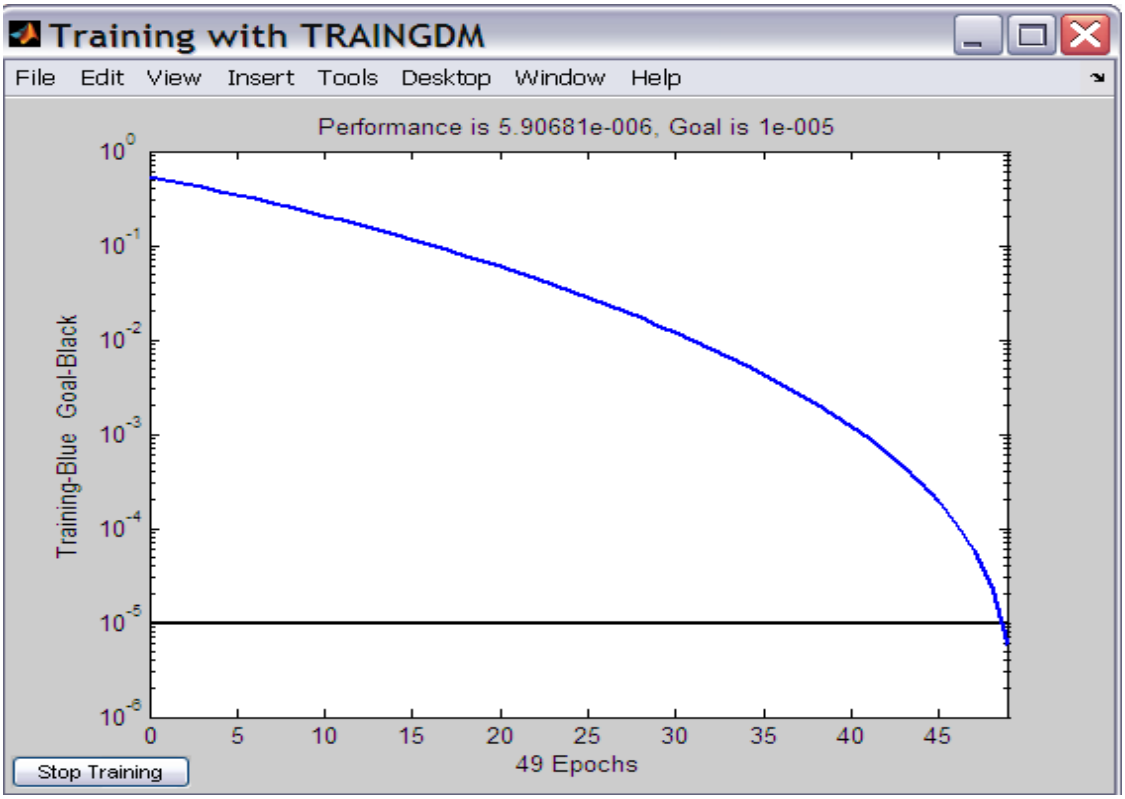


Figure 5.16 Result of training using Traingdm

TRAINGDM, Epoch 0/300, MSE 0.53416/1e-005, Gradient 2.06512/1e-010

TRAINGDM, Epoch 49/300, MSE 5.90681e-006/1e-005, Gradient 0.00686758/1e-010

TRAINGDM, Performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using traingdm trainfunction. In this performance goal of the network has been achived

5.3.4Performance by using the TRAINGDA

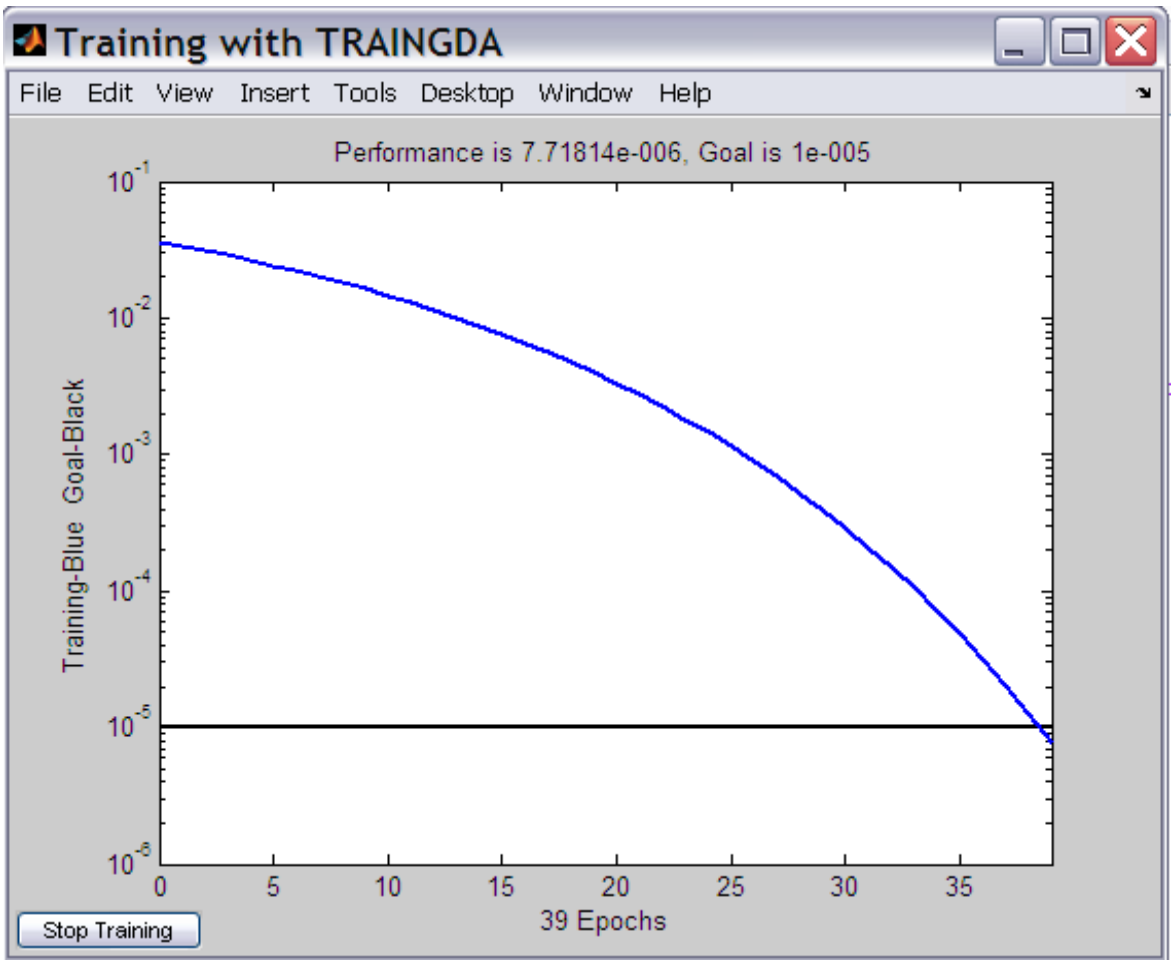


Figure 5.17 Result of training using Traingda

TRAINGDA, Epoch 0/300, MSE 0.0362893/1e-005, Gradient 0.274845/1e-006

TRAINGDA, Epoch 39/300, MSE 7.71814e-006/1e-005, Gradient 0.00415103/1e-006

TRAINGDA, Performance goal met**.**

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using traingda trainfunction. In this performance goal of the network has been achived

5.3.5 Performance by using the TRAINCGB

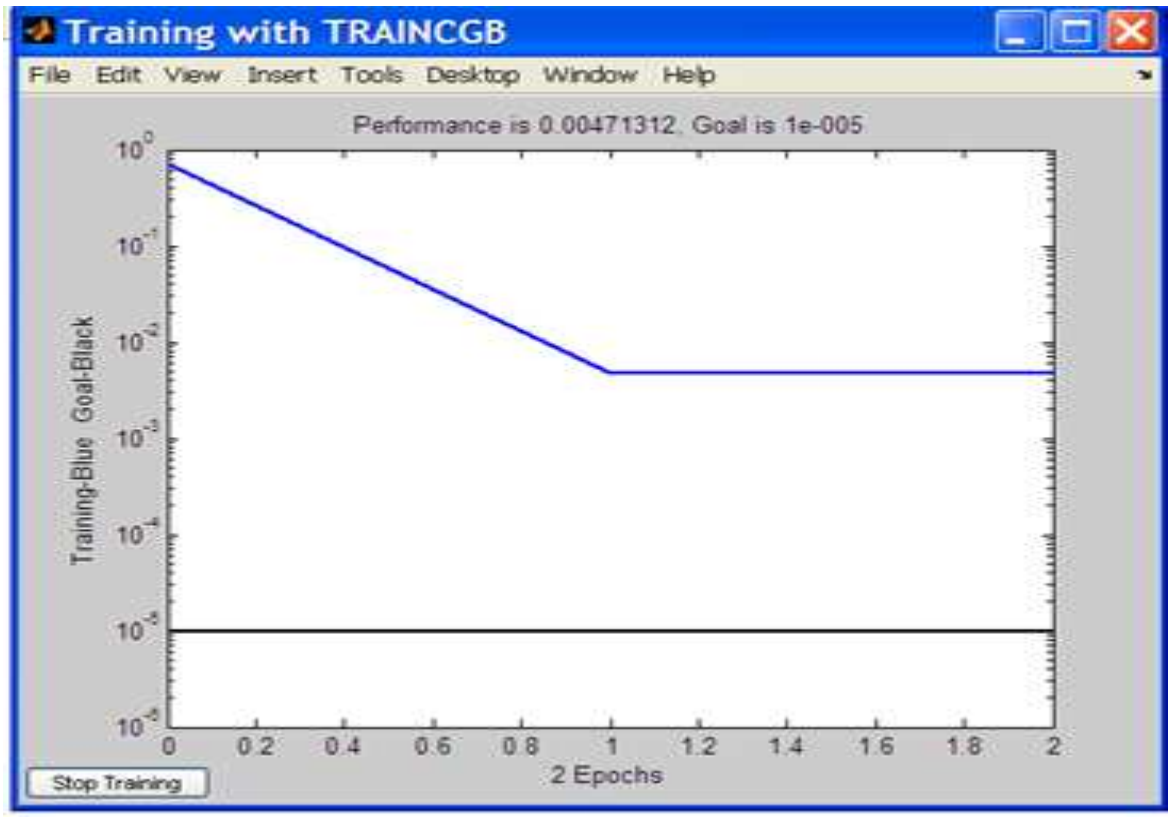


Figure 5.18 Result of training using Traincgb

TRAINCGB-srchcha, Epoch 2/300, MSE 0.00471312/1e-005, Gradient 0.213275/1e-006 TRAINCGB, Performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using traincgb trainfunction. In this performance goal of the network has been achived

5.3.6 Performance by using the TRAINLM

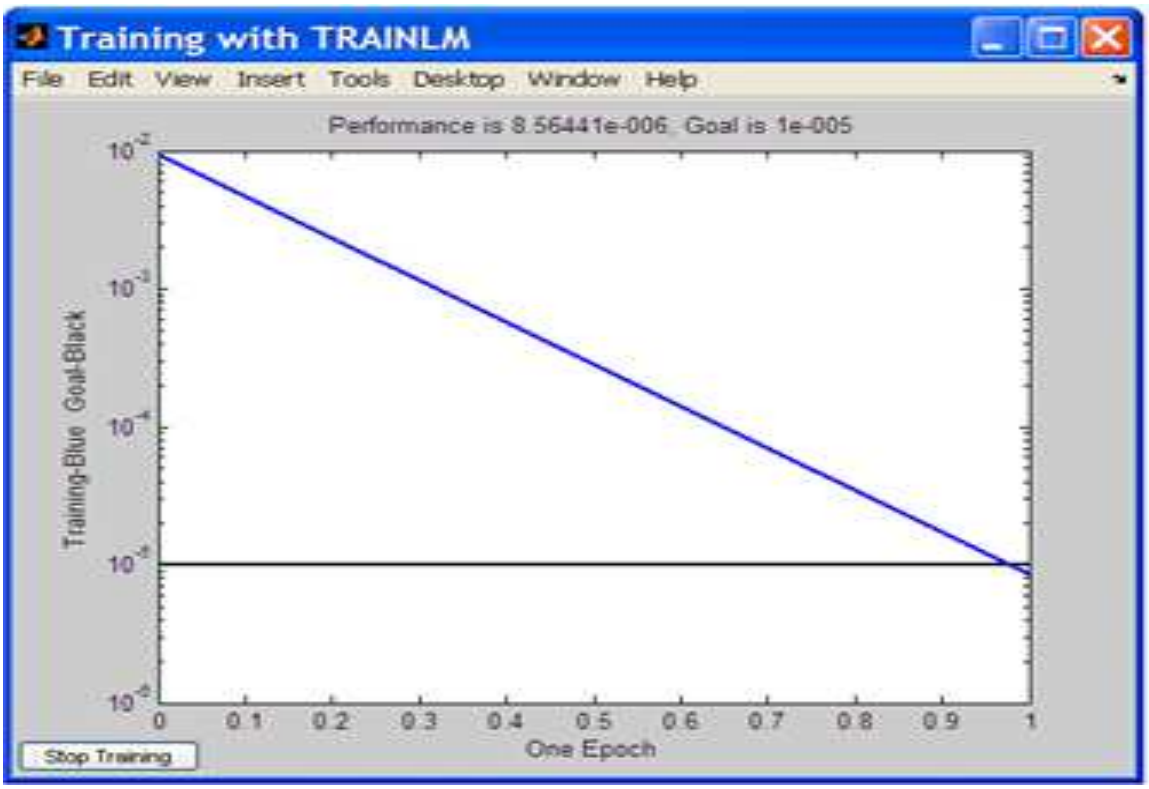


Figure 5.19 Result of training using Trainlm

TRAINLM, Epoch 0/300, MSE 0.00928855/1e-005, Gradient 0.132085/1e-010

TRAINLM, Epoch 1/300, MSE 8.56441e-006/1e-005, Gradient 0.00486758/1e-010

TRAINLM, Performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using trainlm train function. In this performance goal of the network has been achived

5.4 For motor coach

5.4.1 Performance by using the TRAINGD

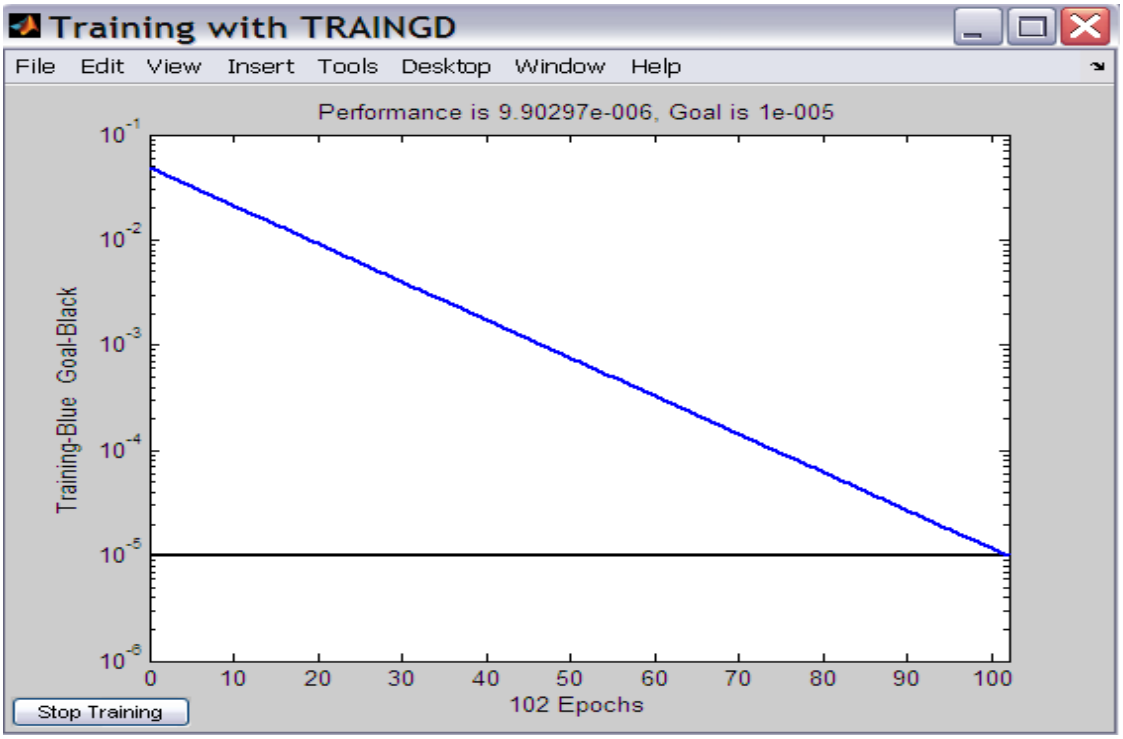


Figure 5.20 Result of training using Traingd

TRAINGD, Epoch 0/500, MSE 0.0488412/1e-005, Gradient 0.631498/1e-010

TRAINGD, Epoch 5/500, MSE 0.0321925/1e-005, Gradient 0.51269/1e-010

TRAINGD, Epoch 10/500, MSE 0.021219/1e-005, Gradient 0.416236/1e-010

TRAINGD, Epoch 15/500, MSE 0.0139861/1e-005, Gradient 0.337929/1e-010

TRAINGD, Epoch 20/500, MSE 0.00921863/1e-005, Gradient 0.274354/1e-010

TRAINGD, Epoch 25/500, MSE 0.00607628/1e-005, Gradient 0.222739/1e-010

TRAINGD, Epoch 30/500, MSE 0.00400506/1e-005, Gradient 0.180835/1e-010

TRAINGD, Epoch 35/500, MSE 0.00263985/1e-005, Gradient 0.146814/1e-010

TRAINGD, Epoch 40/500, MSE 0.00174/1e-005, Gradient 0.119194/1e-010

TRAINGD, Epoch 45/500, MSE 0.00114689/1e-005, Gradient 0.0967694/1e-010

TRAINGD, Epoch 50/500, MSE 0.000755945/1e-005, Gradient 0.0785639/1e-010

TRAINGD, Epoch 55/500, MSE 0.000498264/1e-005, Gradient 0.0637834/1e-010

TRAINGD, Epoch 60/500, MSE 0.00032842/1e-005, Gradient 0.0517836/1e-010

TRAINGD, Epoch 65/500, MSE 0.00021647/1e-005, Gradient 0.0420414/1e-010

TRAINGD, Epoch 70/500, MSE 0.000142681/1e-005, Gradient 0.034132/1e-010

TRAINGD, Epoch 75/500, MSE 9.40451e-005/1e-005, Gradient 0.0277106/1e-010

TRAINGD, Epoch 80/500, MSE 6.19876e-005/1e-005, Gradient 0.0224973/1e-010

TRAINGD, Epoch 85/500, MSE 4.08576e-005/1e-005, Gradient 0.0182648/1e-010

TRAINGD, Epoch 90/500, MSE 2.69303e-005/1e-005, Gradient 0.0148286/1e-010

TRAINGD, Epoch 95/500, MSE 1.77505e-005/1e-005, Gradient 0.0120388/1e-010

TRAINGD, Epoch 100/500, MSE 1.16998e-005/1e-005, Gradient 0.00977388/1e-010

TRAINGD, Epoch 102/500, MSE 9.90297e-006/1e-005, Gradient 0.00899209/1e-010

TRAINGD, Performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using traingd trainfunction. In this performance goal of the network has been achived.

5.4.2 Performance by using the TRAINOSS

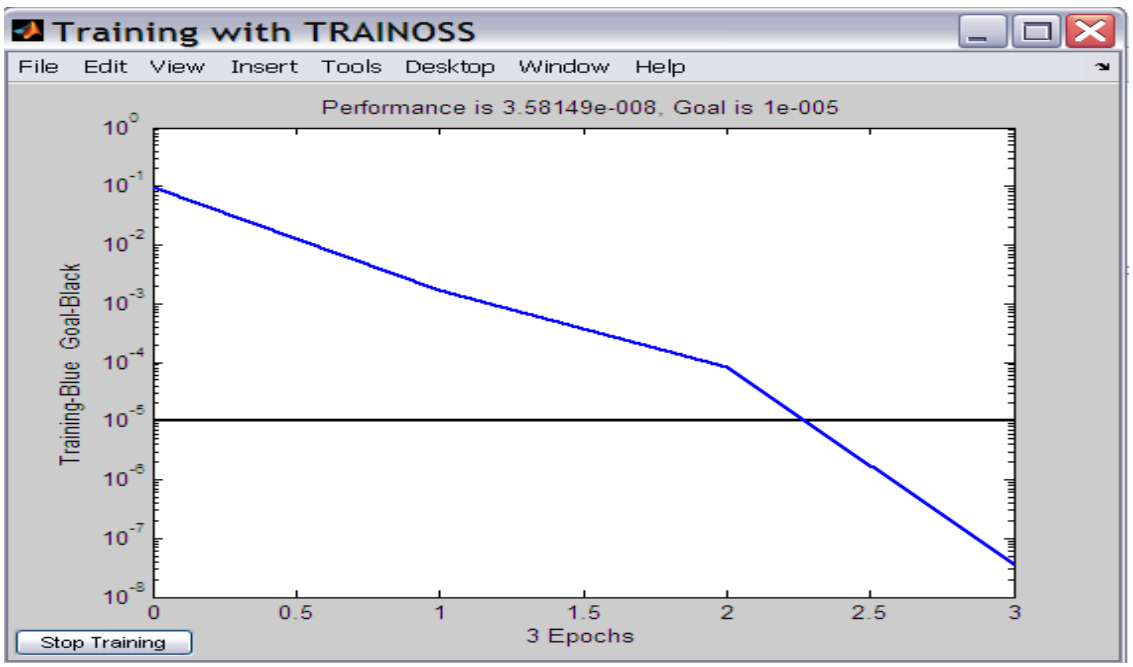


Figure 5.21 Result of training using Trainoss

TRAINOSS-srchbac, Epoch 0/500, MSE 0.0964272/1e-005, Gradient 1.73111/1e-006

TRAINOSS-srchbac, Epoch 3/500, MSE 3.58149e-008/1e-005, Gradient 0.000906779/1e-006

TRAINOSS, Performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using trainoss trainfunction. In this performance goal of the network has been achived

5.4.3 Performance by using the TRAINGDM

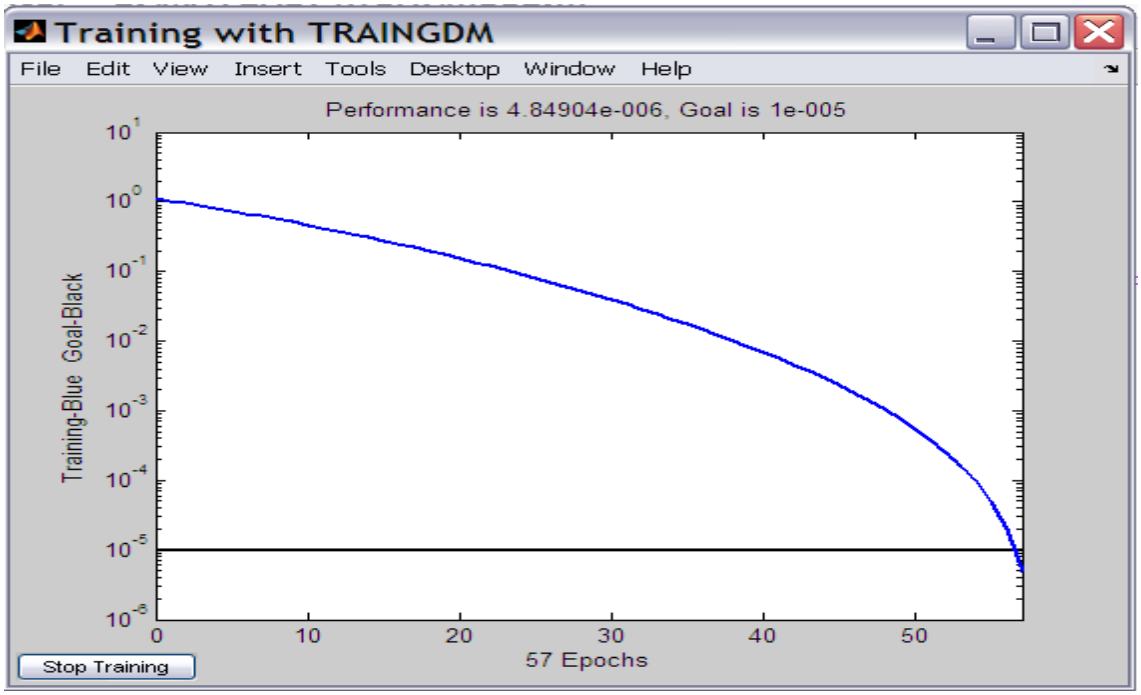


Figure 5.22 Result of training using Traingdm

TRAINGDM, Epoch 0/300, MSE 1.11264/1e-005, Gradient 2.8732/1e-010

TRAINGDM, Epoch 50/300, MSE 0.000543476/1e-005, Gradient 0.0634952/1e-010

TRAINGDM, Epoch 57/300, MSE 4.84904e-006/1e-005, Gradient 0.00620739/1e-010

TRAINGDM, Performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using traingdm trainfunction. In this performance goal of the network has been achived

5.4.4 Performance by using the TRAINLM

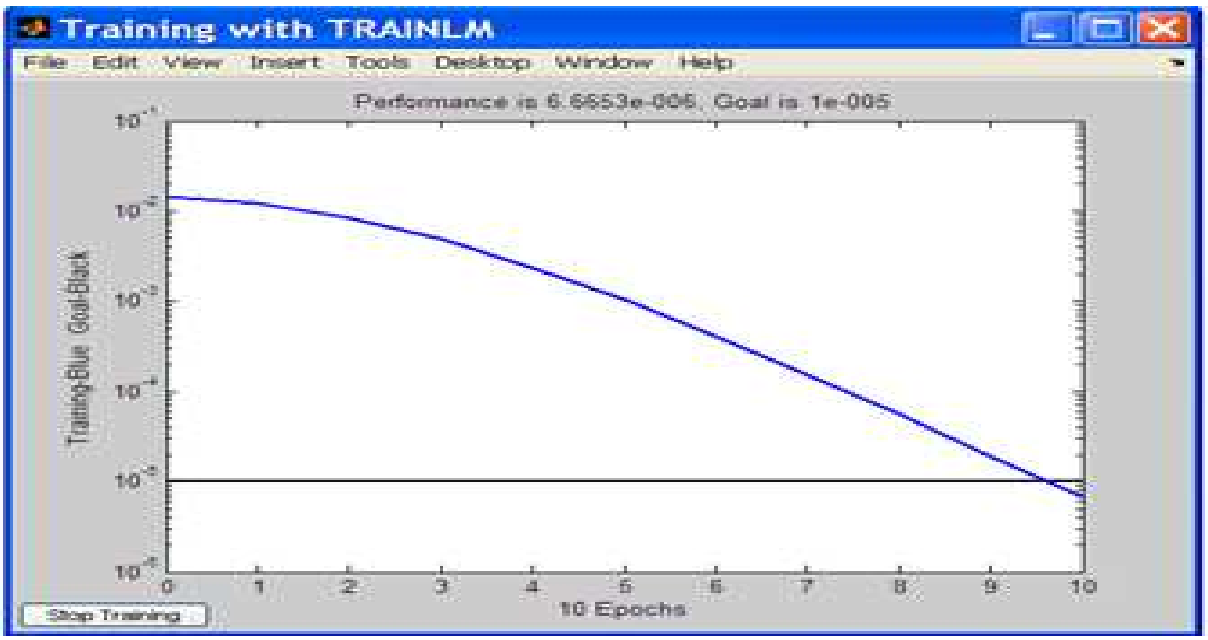


Figure 5.23 Result of training using Trainlm

TRAINLM, Epoch 0/300, MSE 0.014446/1e-005, Gradient 0.339681/1e-010

TRAINLM, Epoch 5/300, MSE 0.00102746/1e-005, Gradient 0.0671757/1e-010

TRAINLM, Epoch 10/300, MSE 6.6653e-006/1e-005, Gradient 0.00571757/1e-010

TRAINLM, Performance goal met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using trainlm train function. In this performance goal of the network has been achived.

5.4.5 Performance by using the TRACGB

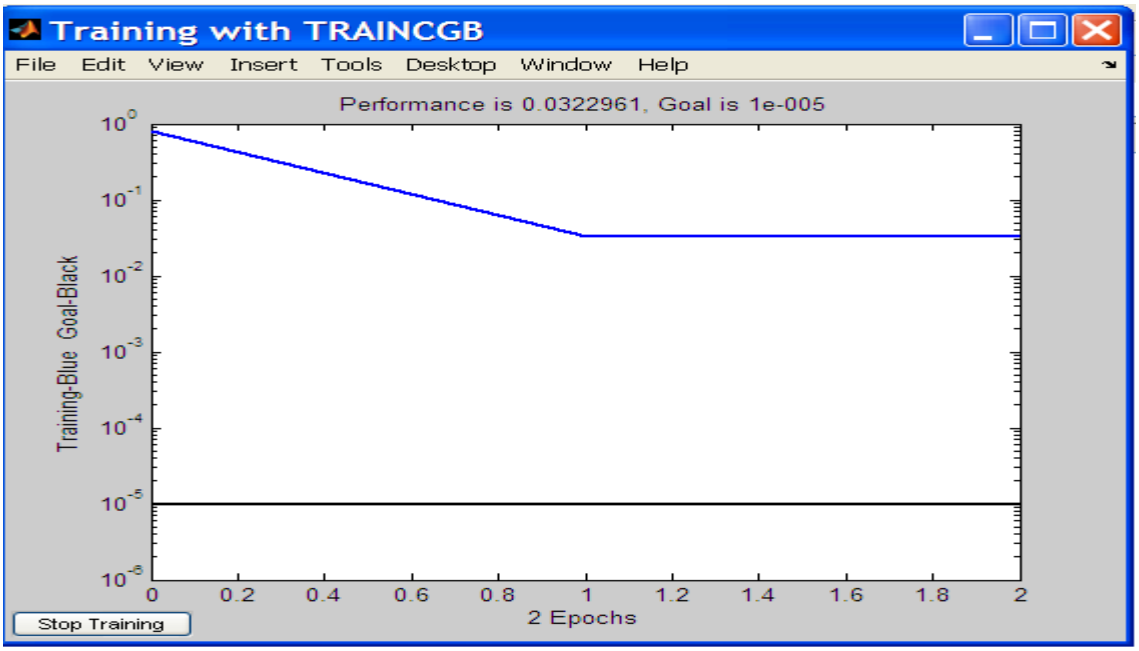


Figure 5.24 Result of training using Traincgb

TRAINCGB-srchcha, Epoch 2/200, MSE 0.0322961/1e-005, Gradient 1.46881/1e-006
TRAINCGB, Minimum step size reached, performance goal was met.

The graph has shown in the Figure represents the output of training the network & Epochs have been taken to get trained the network using traincgb train function. In this performance goal of the network has been achived.

5.5 Performance of QNN Model by using Training functions:

Training function	Car (No. of Epochs)	Truck (No. of Epochs)	Microbus (No. of Epochs)	Motor coach (No. of Epochs)
Traingd	**69**	**63**	**123**	**102**
Traingda	**43**	**31**	**39**	**42**
Traingdm	**49**	**54**	**49**	**57**
Traincgb	**2**	**2**	**2**	**3**
Trainscg	**3**	**2**	**1**	**3**
Trainnoss	**3**	**2**	**4**	**3**
Trainbfg	**5**	**3**	**3**	**4**
Trainlm	**1**	**4**	**3**	**10**

5.6 Results for validation of QNN Model

Function	QNN output values [for Saloon Car]	QNN output values [for Microbus]	QNN output values [for Truck]	QNN output values [for Motor coach]
Traingd	0.7990	0.3909	0.1999	0.0999
Traingda	0.7909	0.3998	0.1997	0.1001
Traingdm	0.7999	0.3997	0.1998	0.0998
Traincgb	0.8000	0.3999	0.1996	0.0997
Trainscg	0.7989	0.4000	0.1998	0.1000
Trainnoss	0.8001	0.4001	0.2001	0.1002
Trainbfg	0.8000	0.3996	0.1995	0.0999
Trainlm	0.8002	0.4000	0.1999	0.0998

Chapter 6

Conclusions and Future Scope

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.

The training functions using back propagation quantum neural network algorithm has been implemented in the Matlab environment. The back propagation of quantum neural network has been trained and tested for the viechle classification data. It has been observed that the convergence time for the training function traincgb of back propagation quantum neural network. Quantum neural network model have been achieved the performance goal with desired accuracy of the results by using the complex valued function.

The training of quantum neural network algorithm has performed by using training function like as traingda, trainrp, traincgf, traincg, trainlm of the Mat Lab environment. It has also been observed that the Result obtain in table 5.6 are in good agreement with the desired values. The proposed research work gives a faster estimation for the analysis of classification data. It is well documented that the back propagation of quantum neural network model with training function trainbgf gives the faster training convergence time.

Future Scope

In the future work this system can be trained with different training function with the MATLAB environment. As showed in this thesis work, back propagation quantum neural networks can be successfully to implement the complex valued function used in the training function. The same experiments can also be conducted with other types of quantum neural network modules with that may improve the performance of the system of QNN with the back propagation algorithm. This training algorithm can also be used with the special training functions.

Referances

1. Bob Ricks, Dan Ventura "Training a Quantum Neural Network" Advance in neural information processing system, NIPS press,2003.
2. Minsky, Marvin L, and Papert, and Seymour S., "Perceptrons: An Introduction to Computational Geometry", MIT Press, Cambridge, MA 1969.
3. Alexandr Ezhov and Dan Ventura. "Quantum neural networks". Future Directions for Intelligent Systems and Information Science. Physica- Verlang, 2000.
4. Ali Zilouchian "Fundamentals of neuralnetworks" by CRC Press LLC, 2001
5. Widrow, B and Hoff, M.E, "Adaptive Switching Circuits, IREWESCON Convention Record, Part 4, NY, IRE, 96–104, 1960.
6. Fausett, L, "Fundamentals of Neural Networks", Prentice-Hall,Englewood Cliffs, NJ, 1994.
7. Haykin, S, "Neural Networks: A Comprehensive Foundation", Prentice Hall, Upper Saddle River, NJ, 1999
8. Hopfield, J.J. "Neural networks and physical systems with emergent collective computational abilities", Proceedings of the National Academy of Sciences USA, vol.79, pp.2554-2558, 1982
9. Behrman, E.C, Niemel, J., Steck, J.E., and Skinner, S.R. "A quantum dot neural network". Proceedings of the 4th Workshop on Physics of Computation, Boston, pp.22-24, November, 1996.
10. Behrman, E.C, Steck, J.E., and Skinner, S.R. "A spatial quantum neural computer", Proceedings of the International Joint Conference on Neural Networks, to appear, 1999.
11. Goertzel, B " Quantum Neural Networks". http://goertzel/org/ben/quantnet.html.
12. G. Bonnell I and G. Papini "Quantum Neural Network" International Journal of Theoretical Physics, Vol. 36, No. 12, 1997.
13. Christos Stergiou and Dimitrios Siganos, "Neural Networks", Computer Science Deptt. University of U.K., Journal, Vol. 4, 1996.
14. Robert J Schalkoff, "Artificial Neural Networks", McGraw-Hill International Editions, 1997.

15. Margolus,N, W. Zurek,ed “In complexity,Entropy, and physics of information”, Addition- Wesley, Redwood city, California,1990.
16. Howard Demuth Mark Beale “Neural Network Toolbox” For Use with MATLAB”.
17. Peter Shor, “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer”. In SIAM Journal of Computing, volume 26 no. 5, pages 1484–1509, 1997.
18. Vlatko Vedral, Adriano Barenco, and Artur Ekert. “Quantum networks for elementary arithmetic operations”. In Physical Review A, volume 54 no. 1, pages 147–153, 1996.
19. Dr. Benni Reznik “Quantum Information” Spring , Tel-Aviv University written by Amir Seginer last modified: March 3rd, 2003-05.
20. Eleanor Rieffel and WOLFGANG POLAK “An Introduction to Quantum Computing for Non-Physicists” FX Palo Alto Laboratory.
21. LI Fei Zheng Baoyu “A study of quantum neural networks”, IEEE int.Conf. Neural Networks & Signal Processing Nanjing, China, December 14- 17, 2003
22. Jarernsri. L.Mitrpanont, and Ananta Srisuphab “The realization of quantum complex-valued backpropagation neural network in pattern recognition problem”, Procd. of the 9th Int Conf on Neural Information Processing (ICONIP’OZ) , Vol. 1.pages 462-466, Nov,2002.

Paper Published/Communicated

- Mukesh Kumar, V.P. Singh **"Quantum Neural Network Training using Matlab"** at International Conference on Advances in Computing Control and Telecommunication Technology, Trivandrum, Kerala, India Dec 24-25, 2009.[Communicated]