IMPLEMENTATION OF BACK PROPAGATION ALGORITHM (of neural networks) IN VHDL

Thesis report submitted towards the partial fulfillment of requirements for the award of the degree of

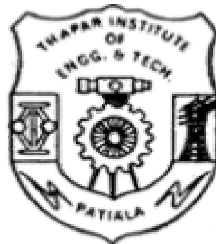
Master of Engineering (Electronics & Communication)

Submitted by

Charu Gupta
Roll No 8044109

Under the Guidance of

Mr. Sanjay Sharma
Astt. Professor, ECED
&
Mrs. Manu Bansal
Lecturer, ECED



Department Of Electronics and Communication Engineering
THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY,
(Deemed University), PATIALA – 147004, INDIA

JUNE, 2006

CERTIFICATE

I, CHARU GUPTA hereby certify that the work which is being presented in this thesis entitled **"VHDL IMPLEMENTATION OF BACK PROPAGATION ALGORITHM FOR NEURAL NETWORKS**" by me in partial fulfillment of requirements for the award of degree of Master of Engineering in Electronics and Communication from THAPAR INSTITUTE OF ENGG & TECH (Deemed University), PATIALA, is an authentic record of my own work carried under the supervision of Mr. SANJAY SHARMA and Mrs. MANU BANSAL at TIET, PATIALA.

The matter presented in this thesis has not been submitted in any other University or Institute for the award of Master of Engineering.

(**CHARU GUPTA**)
Signature of the student

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

(Mr. SANJAY SHARMA)
ASSISTANT PROFESSOR, TIET PATIALA
GUIDE

(Mrs. MANU BANSAL)
Sr. LECTURER, TIET PATIALA
CO-GUIDE

(Dr. R.S. KALER)
Head of Department, ECED

(Dr. T.P. SINGH)
Dean of Academic Affairs,

T.I.E.T, PATIALA. T.I.E.T, PATIALA.

ACKNOWLEGEMENT

Words are often too less to reveal one's deep regards. An understanding of the work like this is never the outcome of the efforts of a single person. I take this opportunity to express my profound sense of gratitude and respect to all those who helped me through the duration of this thesis.

First, I would like to thank the Supreme Power, the GOD, who guided me to work on the right path of life. Without his grace, this would not have been possible. This work would not have been possible without the encouragement and able guidance of **'Mr. SANJAY SHARMA'**, Assistant Professor and **'Mrs. MANU BANSAL',** Sr. Lecturer, TIET, PATIALA. Their enthusiasm and optimism made this experience both rewarding and enjoyable. Most of the novel ideas and solutions found in this thesis are the result of our numerous stimulating discussions. His feedback and editorial comments were also invaluable for the writing of this thesis. I am grateful to **Head of the Department Dr. R. S. KALER** and **Dr. A.K. CHATTERJEE** for providing the facilities for the completion of thesis.

I take pride of my self being daughter of ideal great parents whose everlasting desire, sacrifice, affectionate blessing and help without which it would have not been possible for me to complete my studies.

At last, I would like to thank all the members and employees of Electronics and Communication Department, TIET Patiala whose love and affection made this possible.

(CHARU GUPTA)

LIST OF FIGURES

FIGURE No.	NAME OF FIGURE	PAGE No.
2.1	Components of neuron	12
2.2	The synapse	12
2.3	Neuron model	13
2.4	A simple neuron	14
2.5	MCP neuron	16
2.6	Network layers	17
2.7	Feed forward networks	18
2.8	Feed back networks	19
2.9	The Mcculloh Pitts model	19
2.10	Transfer functions	22
2.11	Back Propagation network	24
2.12	The principle layout for a reinforcement learning agent	26
2.13	Flowchart for implementation of applications	28
3.1	History of VHDL	34
4.1	Back Propagation network	45
5.1	Circuit architecture of prototype chip	51
5.2	Block diagram of neuron	52
5.3	Block diagram of synapse	53
5.4	Block diagram of error generator	54

FIGURE No.	NAME OF FIGURE	PAGE No.
5.5	Block diagram of Back Propagation	55
6.1	Simulation results for synapse	61
6.2	Simulation results for neuron	63
6.3	Simulation results for synapse	65
6.4	Simulation results for neuron	67
6.5	Simulation results for error generator at output	69
6.6	Simulation results for weight update unit	71
6.7	Simulation results for weight transfer unit	73
6.8	Simulation results for error generator at input	75
6.9	Simulation results for weight update unit	77
6.10	Simulation results for weight transfer unit	79
6.11	Simulation results for final entity	82-86

LIST OF TABLES

TABLE No.	NAME OF TABLE	PAGE No.
2.1	Truth Table before applying the firing rules	15
2.2	Truth Table after applying the firing rules	15
5.1	Table of signals for Back Propagation algorithm	56

TABLE OF CONTENTS

CHAPTER TITLE **PAGE No.**

Certificate ……………………………………………………………………….i
Acknowledgement…………………………………………………………….ii
Abstract…………………………………………………………………………...iii
Table of Contents………………………………………………………………iv
List of Figures………………………………………………………………….ix
List of Tables…………………………………………………………………..xi

Chapter 1 Introduction

1.1 Introduction......................................…...1

1.2 Advantages of neural networks ……………………………………………….1

1.3 Learning in neural networks……………………...2

1.4 Overview of back propagation algorithm……………………………………...2

1.5 Use of back propagation neural network solution…………….. ……………...3

1.6 Objective of thesis…...……………………………………………………...5

1.7 Organization of thesis……………………………………………………………5

Chapter 2 Introduction to neural networks

2.1 Introduction ....…....................6
2.2 History of Neural Networks…………………………………………………...7
2.3 Advantages of Neural Networks……..………………………………………..9
2.4 Neural Networks versus Conventional Computers…………………………..10
2.5 Human and Artificial Neurons-Investigating the Similarities………….…....11
2.5.1 Learning process in Human Brain. ……………………………………11
2.5.2 Human Neurons to Artificial Neurons….…………..…………………13
2.6 An engineering approach………..……………………………………………13
2.6.1 A simple neuron ………………………………………………......…...13
2.6.2 Firing rules……………………..…………14

2.6.3 A more complicated neuron…………………………......…………16
2.7 Architecture of Neural Networks……….……………………………………17
2.7.1 Network Layers…………..……………………………………………17
2.7.2 Feed-Forward Networks………………………………………………..18
2.7.3 Feedback Networks……………………………………………………18
2.8 Perceptrons…………………………………………………………………...19
2.8.1The Learning Process……………………………………….…………..20
2.8.2 Transfer Function…………………………………………………....22
2.9 Learning Algorithms of Neural Networks……………………………….....23
2.9.1 Supervised Learning……………………………………………….....24
2.9.1.1 The Back Propagation Learning………………………………..24
2.9.1.2 Reinforcement Learning………………………………………..25
2.9.2 Unsupervised Learning…………………………………………………26
2.9.2.1 Hebbian Learning………………………………………………26

2.9.2.2 Competitive Learning…………………………………………..…27
2.10Applications of Neural Networks…………………………………………...27
2.10.1 Neural Networks in Practice……………………………………….28
2.10.2 Neural Networks in Medicine……………………………………..29
2.10.2.1 Modeling and Diagnosing the Cardiovascular System …….29
2.10.2.2 Electronic Noses………………………………………….…30
2.10.2.3 Instant Physician……………………………………….…30
2.10.3 Neural Networks in Business……………………………………...31

Chapter 3 VHSIC Hardware Description Language (VHDL)

3.1 Introduction…………..32
3.2 History of VHDL………………………………………………….…33
3.3 Advantages of VHDL……………………………………...………34
3.4 Entities and Architecture………………………………………………..35
3.5 Data Types ………………………………………………………...35
3.6 Design Units………………………………………………………….36

3.6.1 Entities…………………………………………....………….37
3.6.2 Architectures………………………………………………...37
3.6.3 Packages and Package Bodies……………………………………….38
3.6.4 Configurations…………………….……………………………39
3.7 Levels of abstraction……………………………………………...39
3.7.1 Behavior……………………………………………..…..40
3.7.2 Dataflow……………………………………………………….41
3.7.3 Structure……………………………………………………….41
3.8 Objects, Data Types and Operators……………………………………..42
3.8.1 Using Signals……………………………………………...…42
3.8.2 Using Variables……………………………………………...42

Chapter 4 Back propagation Algorithm

4.1 Introduction……………………………………………………………………..43

4.2 History of algorithm……………………………………………………………43

4.3 Leaning with the back propagation algorithm………………………………..44

4.4 Implementation of back propagation algorithm……………………………...46

4.5 Drawbacks…………………………………………………………………....48

Chapter 5 Hardware Implementation

5.1 Introduction…………………………………………………………….…….50

5.2 Chip architecture…………………………………………………….…….51

5.3 Detailed analysis of the block diagram and hardware implementation………54

5.4 Brief Description of the Entities………………………………………...…56

5.4.1 Synapse……………………………………………………………………56

5.4.2 Neuron…………………………………………………………………..…57

5.4.3 Error Generator at the output………………………………………...57

5.4.4 Weight update……………………………………………………………..58

5.4.5 Weight transfer unit……………………………………………………….58

CHAPTER	TITLE	PAGE No.

5.4.6 Error Generator at the Input……………………………………………58

Chapter 6 Simulation Results

6.1 Synapse………………………………………………………………….…60

6.1.1 Inputs and outputs for the entity Synapse……………………………...60

6.1.2 Simulation Results for the entity Synapse…………………….………..61

6.2 Neuron……………………………………………………………………………..62

6.2.1 Inputs and outputs for the entity Neuron……………………………….62

6.2.2 Simulation Results for the entity Neuron………………………………63

6.3 Synapse……………………………………………………………………….64

6.3.1 Inputs and outputs for the entity Synapse……………………………...64
6.3.2 Simulation Results For the entity Synapse…………………….……….65
6.4 Neuron…………………………………………………………….….…66
6.4.1 Inputs and outputs for the entity Neuron……………………………….66
6.4.2 Simulation Results for the entity Neuron………………………………67
6.5 Error Generator at Output……………………………………………….68
6.5.1 Inputs and outputs for the entity Error Generator at the output………..68
6.5.2 Simulation Results for the entity Error Generator at the output……..…69
6.6 Weight Update……………………………………………………….…70
6.6.1 Inputs and outputs for the entity Weight Update.....................……...70
6.6.2 Simulation Results for the entity Weight Update………………………71
6.7 Weight Transfer…………………………………………………………72
6.7.1 Inputs and outputs for the entity Weight Transfer...................……...72
6.7.2 Simulation Results for the entity Weight Transfer…………………..…73
6.8 Error Generator at Input……………………………………………...74
6.8.1 Inputs and outputs for the entity Error Generator at Input…………….74
6.8.2 Simulation Results for the entity Error Generator at Input……………75
6.9 Weight Update…………………………………………………………..…76
6.9.1 Inputs and outputs for the entity Weight Update.................…………76
6.9.2 Simulation Results for the entity Weight Update ……………………...77
6.10 Weight Transfer……………………………………………………………..78

CHAPTER TITLE PAGE No.

6.10.1 Inputs and outputs for the entity Weight Transfer.......................……..78
6.10.2 Simulation Results for the entity Weight Transfer …………………...79
6.11 Final Entity……………………………………………………………….80
6.11.1 Inputs and Outputs for the Final Entity…………………………….…80
6.11.2 Simulation Results for the Final Entity……………………………….82

Chapter 7 Conclusion and Future Scope

7.1 Conclusion……………………………………………………………..…88
7.2 Future scope…………………………………………………………..……..89

References ….……………………………………………………………………....90

List of Publications……………………………………………………………………….9
3

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Borrowing from biology, researchers are exploring neural networks—a new, non-algorithmic approach to information processing.

A **Neural Network** is a powerful data-modeling tool that is able to capture and represent complex input/output relationships. The motivation for the development of neural network technology stemmed from the desire to develop an artificial system that could perform "intelligent" tasks similar to those performed by the human brain. Neural networks resemble the human brain in the following two ways:

- A neural network acquires knowledge through learning.

- A neural network's knowledge is stored within inter-neuron connection strengths known as synaptic weights. [1]

1.2 ADVANTAGES OF NEURAL NETWORKS

- **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 may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.

- **Fault Tolerance via Redundant Information Coding**: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage. [2]

1.3 LEARNING IN NEURAL NETWORKS

Learning is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place. [3] All learning methods used for neural networks can be classified into two major categories:

SUPERVISED LEARNING which incorporates an external teacher, 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 (back propagation algorithm), reinforcement learning and stochastic 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. Paradigms of unsupervised learning are Hebbian learning and competitive learning.

1.4 OVERVIEW OF BACK PROPAGATION ALGORITHM

Minsky and Papert (1969) showed that there are many simple problems such as the exclusive-or problem which linear neural networks can not solve. Note that term "solve" means learn the desired associative links. Argument is that if such networks can not solve such simple problems how they could solve complex problems in vision, language, and motor control. Solutions to this problem were as follows:

- Select appropriate "recoding" scheme which transforms inputs
- Perceptron Learning Rule -- Requires that you correctly "guess" an acceptable input to hidden unit mapping.
- Back-propagation learning rule -- Learn both sets of weights simultaneously.

Back propagation is a form of supervised learning for multi-layer nets, also known as the generalized delta rule. Error data at the output layer is "back propagated" to earlier ones, allowing incoming weights to these layers to be updated. It is most often used as training algorithm in current neural network applications. The back propagation algorithm was developed by Paul Werbos in 1974 and rediscovered independently by Rumelhart and Parker. Since its rediscovery, the back propagation algorithm has been widely used as a learning algorithm in feed forward multilayer neural networks.

What makes this algorithm different than the others is the process by which the weights are calculated during the learning network. In general, the difficulty with multilayer Perceptrons is calculating the weights of the hidden layers in an efficient way that result in the least (or zero) output error; the more hidden layers there are, the more difficult it

becomes. To update the weights, one must calculate an error. At the output layer this error is easily measured; this is the difference between the actual and desired (target) outputs. At the hidden layers, however, there is no direct observation of the error; hence, some other technique must be used. To calculate an error at the hidden layers that will cause minimization of the output error, as this is the ultimate goal.

The back propagation algorithm is an involved mathematical tool; however, execution of the training equations is based on iterative processes, and thus is easily implementable on a computer.

1.5 USE OF BACK PROPAGATION NEURAL NETWORK SOLUTION

- A large amount of input/output data is available, but you're not sure how to relate it to the output.
- The problem appears to have overwhelming complexity, but there is clearly a solution.
- It is easy to create a number of examples of the correct behavior.
- The solution to the problem may change over time, within the bounds of the given input and output parameters (i.e., today 2+2=4, but in the future we may find that 2+2=3.8).
- Outputs can be "fuzzy", or non-numeric.

One of the most common applications of NNs is in image processing. Some examples would be: identifying hand-written characters; matching a photograph of a person's face with a different photo in a database; performing data compression on an image with minimal loss of content. Other applications could be voice recognition; RADAR signature analysis; stock market prediction. All of these problems involve large amounts of data, and complex relationships between the different parameters.

It is important to remember that with a NN solution, you do not have to understand the solution at all. This is a major advantage of NN approaches. With more traditional techniques, you must understand the inputs, and the algorithms, and the outputs in great

detail, to have any hope of implementing something that works. With a NN, you simply show it: "this is the correct output, given this input". With an adequate amount of training, the network will mimic the function that you are demonstrating. Further, with a NN, it is ok to apply some inputs that turn out to be irrelevant to the solution - during the training process; the network will learn to ignore any inputs that don't contribute to the output. Conversely, if you leave out some critical inputs, then you will find out because the network will fail to converge on a solution

1.6 OBJECTIVE OF THESIS

The objectives of thesis are:

- Exploration of a supervised learning algorithm for artificial neural networks i.e. the Error Back propagation learning algorithm for a layered feed forward network.
- Formulation of individual modules of the Back Propagation algorithm for efficient implementation in hardware.
- Implementation of the Back Propagation learning algorithm in VHDL. VHDL implementation creates a flexible, fast method and high degree of parallelism for implementing the algorithm.
- Analysis of the simulation results of Back Propagation algorithm.

1.7 ORGANIZATION OF THESIS

- This thesis begins with a brief introduction to neural networks and the back propagation learning algorithm.

- Chapter 2 gives a brief introduction to neural networks, history, their architecture, different learning algorithms and applications.
- Chapter 3 gives a brief introduction to VHDL language, its advantages, design units and different levels of abstraction.
- Chapter 4 describes the Back Propagation algorithm, its history, different steps for implementation and drawbacks of the algorithm.
- Chapter 5 gives a detailed analysis of block diagram of Back Propagation algorithm and a discussion of each of the modules the block diagram has been spilt into.
- Chapter 6 gives the simulation results of the implementation of the back propagation algorithm in MODELSIM SE 5.5c.
- Chapter 7 gives the conclusion and future scope of the work proposed in this thesis.

CHAPTER 2

INTRODUCTION TO NEURAL NETWORKS

2.1 INTRODUCTION

Borrowing from biology, researchers are exploring neural networks—a new, non-algorithmic approach to information processing.

A **neural network** is a powerful data-modeling tool that is able to capture and represent complex input/output relationships. The motivation for the development of neural network technology stemmed from the desire to develop an artificial system that could perform "intelligent" tasks similar to those performed by the human brain. Neural networks resemble the human brain in the following two ways:

- A neural network acquires knowledge through learning.

- A neural network's knowledge is stored within inter-neuron connection strengths known as synaptic weights.

Artificial Neural Networks are being counted as the wave of the future in computing. They are indeed self-learning mechanisms which don't require the traditional skills of a programmer. But unfortunately, misconceptions have arisen. Writers have hyped that these neuron-inspired processors can do almost anything. These exaggerations have created disappointments for some potential users who have tried, and failed, to solve their problems with neural networks. These application builders have often come to the conclusion that neural nets are complicated and confusing. Unfortunately, that confusion has come from the industry itself. An avalanche of articles has appeared touting a large assortment of different neural networks, all with unique claims and specific examples. Currently, only a few of these neuron-based structures, paradigms actually, are being used commercially. One particular structure, the feed forward, back-propagation network, is by far and away the most popular. Most of the other neural network structures represent models for "thinking" that are still being evolved in the laboratories. Yet, all of these networks are simply tools and as such the only real demand they make is that they require the network architect to learn how to use them. [4]

The power and usefulness of artificial neural networks have been demonstrated in several applications including speech synthesis, diagnostic problems, medicine, business and finance, robotic control, signal processing, computer vision and many other problems that fall under the category of pattern recognition. For some application areas, neural models show promise in achieving human-like performance over more traditional artificial intelligence techniques.

2.2 HISTORY OF NEURAL NETWORKS

The study of the human brain is thousands of years old. With the advent of modern electronics, it was only natural to try to harness this thinking process. [2], [5]

The history of neural networks that was described above can be divided into several periods:

- **First Attempts**: 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.
- **Promising & Emerging Technology**: Not only was neuroscience influential in the development of neural networks, but psychologists and engineers also contributed to the progress of neural network simulations. Rosenblatt (1958) stirred considerable interest and activity in the field when he designed and developed the *Perceptron*. The Perceptron had three layers with the middle layer known as the association layer. This system could learn to connect or associate a given input to a random output unit. Another system was the ADALINE (*Adaptive Linear Element*) which was developed in 1960 by Widrow and Hoff (of Stanford University). The ADALINE was an analogue electronic device made from simple components. The method used for learning was different to that of the Perceptron; it employed the Least-Mean-Squares (LMS) learning rule.
- **Period of Frustration & Disrepute**: In 1969 Minsky and Papert wrote a book in which they generalized the limitations of single layer Perceptrons to multilayered systems. In the book they said: "...our 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.

- **Innovation**: Although public interest and available funding were minimal, several researchers continued working to develop neuromorphically based computational methods for problems such as pattern recognition. During this 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. Back-propagation 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: he published a paper which 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*.
- **Re-Emergence**: Progress 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. Academic programs appeared and courses were introduced at most major Universities (in US and Europe). Attention is now focused on funding levels throughout Europe, Japan and the US and as this funding becomes available, several new commercial with applications in industry and financial institutions are emerging.

- **Today**: Significant progress has been made in the field of neural networks-enough to attract a great deal of attention and fund further research. Advancement beyond current commercial applications appears to be possible, and research is advancing the field on many fronts. Neurally based chips are emerging and applications to complex problems developing. Clearly, today is a period of transition for neural network technology.

2.3 ADVANTAGES OF NEURAL NETWORKS

Either humans or other computer techniques can use neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, to extract patterns and detect trends that are too complex to be noticed. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. 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 may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
- **Fault Tolerance via Redundant Information Coding**: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

2.4 NEURAL NETWORKS VERSUS CONVENTIONAL COMPUTERS

Neural networks take a different approach to problem solving than that of conventional computers.

- **Conventional computers** use an algorithmic approach i.e. the computer follows a set of instructions in order to solve a problem. Unless the specific steps that the computer needs to follow are known the computer cannot solve the problem. That restricts the problem solving capability of conventional computers to problems that we already understand and know how to solve. But computers would be so much more useful if they could do things that we don't exactly know how to do.

  Neural networks on the other hand, process information in a similar way the human brain does. The network is composed of a large number of highly interconnected processing elements (neurons) working in parallel to solve a specific problem. Neural networks learn by example. They cannot be programmed to perform a specific task.

- The disadvantage of **neural networks** is that because the network finds out how to solve the problem by itself, its operation can be unpredictable.

  On the other hand, **conventional computers** use a cognitive approach to problem solving; the way the problem is to solve must be known and stated in small unambiguous instructions. These instructions are then converted to a high-level language program and then into machine code that the computer can understand. These machines are totally predictable; if anything goes wrong is due to a software or hardware fault.

Neural networks and conventional algorithmic computers are not in competition but complement each other. There are tasks are more suited to an algorithmic approach like arithmetic operations and tasks that are more suited to neural networks. Even more, a large number of tasks require systems that use a combination of the two approaches (normally a conventional computer is used to supervise the neural network) in order to perform at maximum efficiency.

2.5 HUMAN AND ARTIFICIAL NEURONS-INVESTIGATING THE SIMILARITIES

2.5.1 LEARNING PROCESS IN HUMAN BRAIN

Much is still unknown about how the brain trains itself to process information, so theories abound. In the human brain, a typical neuron collects signals from others through a host of fine structures called *dendrites*.

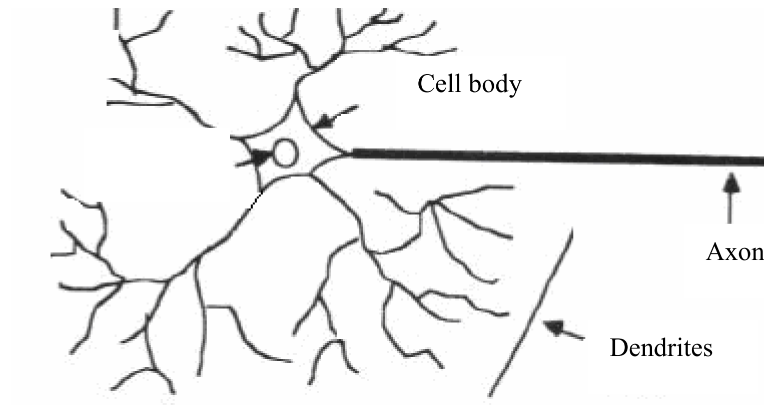


Fig- 2.1: Components of a Neuron

The neuron sends out spikes of electrical activity through a long, thin stand known as an *axon*, which splits into thousands of branches. At the end of each branch, a structure called a *synapse* converts the activity from the axon into electrical effects that inhibit or excite the activity in the connected neurons. When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity down its axon. Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes. [6]

Synapse

Axon

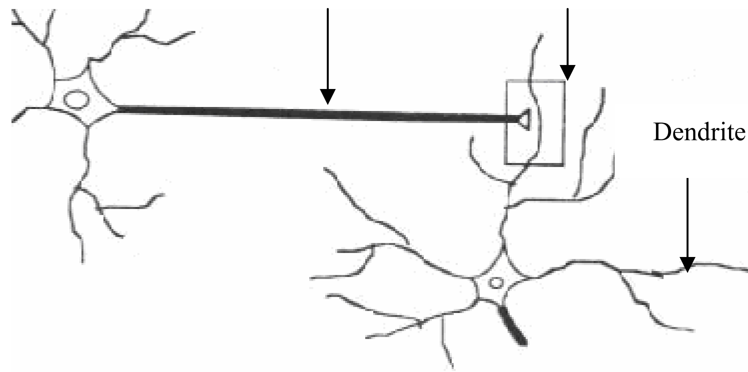


Fig- 2.2: The Synapse

2.5.2 HUMAN NEURONS TO ARTIFICIAL NEURONS

We conduct these neural networks by first trying to deduce the essential features of neurons and their interconnections. We then typically program a computer to simulate these features. However because our knowledge of neurons is incomplete and our computing power is limited, our models are necessarily gross idealizations of real networks of neurons.

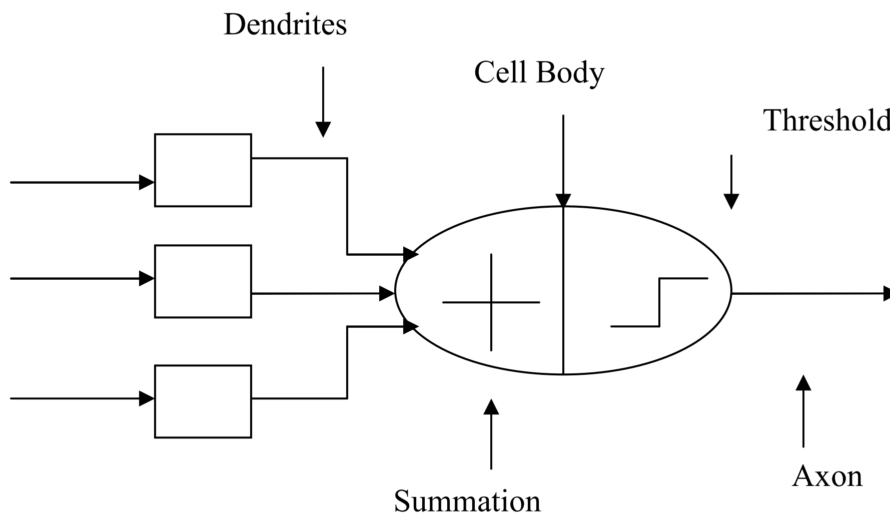


Fig-2.3: Neuron Model

2.6 AN ENGINEERING APPROACH

2.6.1 A SIMPLE NEURON

An artificial neuron is a device with many inputs and one output. The neuron has two **modes of operation**, the **training mode** and the **using mode**.

- In the training mode, the neuron can be trained to fire (or not), for particular input patterns.
- In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not.

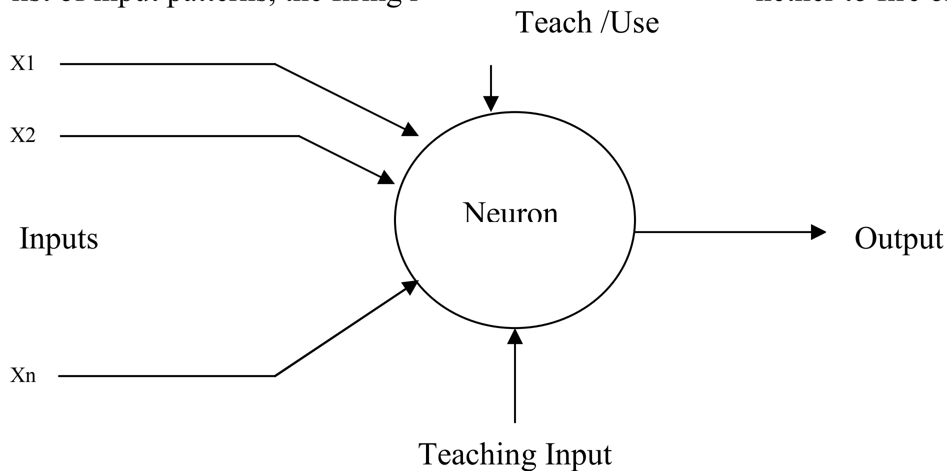


Fig-2.4: A Simple Neuron

2.6.2 FIRING RULES

The firing rule is an important concept in neural networks and accounts for their high flexibility. **A firing rule determines how one calculates whether a neuron should fire for any input pattern**. It relates to all the input patterns, not only the ones on which the node was trained. A simple firing rule can be implemented by using Hamming distance technique. The rule goes as follows:

Take a collection of training patterns for a node, some of which cause it to fire (the 1-taught set of patterns) and others, which prevent it from doing so (the 0-taught set). Then the patterns not in the collection cause the node to fire if, on comparison, they have more

input elements in common with the 'nearest' pattern in the 1-taught set than with the 'nearest' pattern in the 0-taught set. If there is a tie, then the pattern remains in the undefined state.

For example, a 3-input neuron is taught to output 1 when the input (X1, X2 and X3) is 111 or 101 and to output 0 when the input is 000 or 001. Then, before applying the firing rule, the truth table is:

Table 2.1: Truth Table before applying the firing rule

X1:	0	0	0	0	1	1	1	1
X2:	0	0	1	1	0	0	1	1
X3:	0	1	0	1	0	1	0	1
OUT:	0	0	0/1	0/1	0/1	1	0/1	1

As an example of the way the firing rule is applied, take the pattern 010. It differs from 000 in 1 element, from 001 in 2 elements, from 101 in 3 elements and from 111 in 2 elements. Therefore, the 'nearest' pattern is 000, which belongs, in the 0-taught set. Thus the firing rule requires that the neuron should not fire when the input is 001. On the other hand, 011 is equally distant from two taught patterns that have different outputs and thus the output stays undefined (0/1).

By applying the firing in every column the following truth table is obtained:

Table 2.2: Truth Table after applying the firing rule

X1:		0	0	0	0	1	1	1	1
X2:		0	0	1	1	0	0	1	1

X3:		0	1	0	1	0	1	0	1
OUT:		0	0	0	0/1	0/1	1	1	1

The difference between the two truth tables is called the *generalization of the neuron.* Therefore the firing rule gives the neuron a sense of similarity and enables it to respond 'sensibly' to patterns not seen during training.

2.6.3 A MORE COMPLICATED NEURON

The previous neuron doesn't do anything that conventional computers don't do already. A more sophisticated neuron is the McCulloch and Pitts model (MCP). The difference from the previous model is that the **inputs are 'weighted'**; the effect that each input has at decision-making is dependent on the weight of the particular input. The weight of an input is a number which when multiplied with the input gives the weighted input. These weighted inputs are then added together and if they exceed a pre-set threshold value, the neuron fires. In any other case the neuron does not fire.

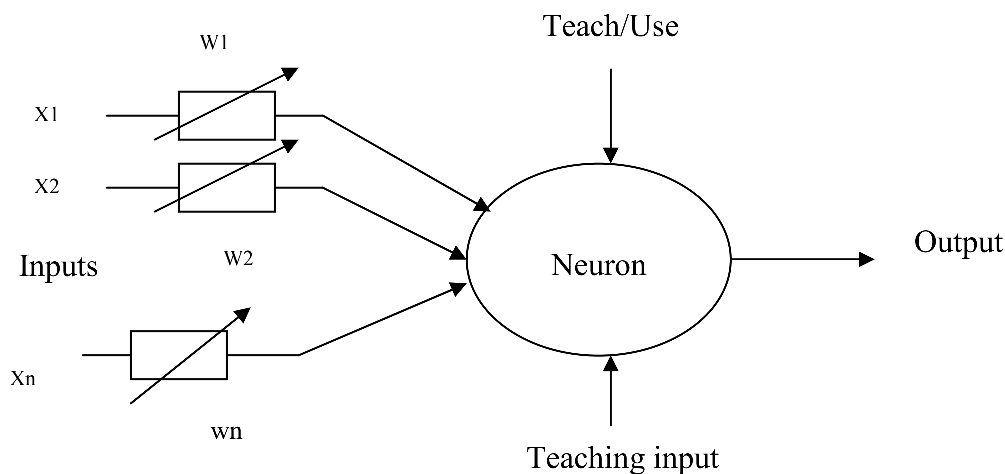


Fig-2.5: MCP Neuron

In mathematical terms, the neuron fires if and only if;

$X_1W_1 + X_2W_2 + X_3W_3 + ... > T$

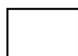
The addition of input weights and of the threshold makes this neuron a very flexible and powerful one. The MCP neuron has the ability to adapt to a particular situation by changing its weights and/or threshold.

2.7 ARCHITECTURE OF NEURAL NETWORKS

2.7.1 NETWORK LAYERS

The commonest type of artificial neural network consists of three groups, or layers, of units: a layer of "**input**" units is connected to a layer of "**hidden**" units, which is connected to a layer of **"output**" units. [6], [7]

- The activity of the input units represents the raw information that is fed into the network.
- The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units.
- The behavior of the output units depends on the activity of the hidden units and the weights between the hidden and output units.



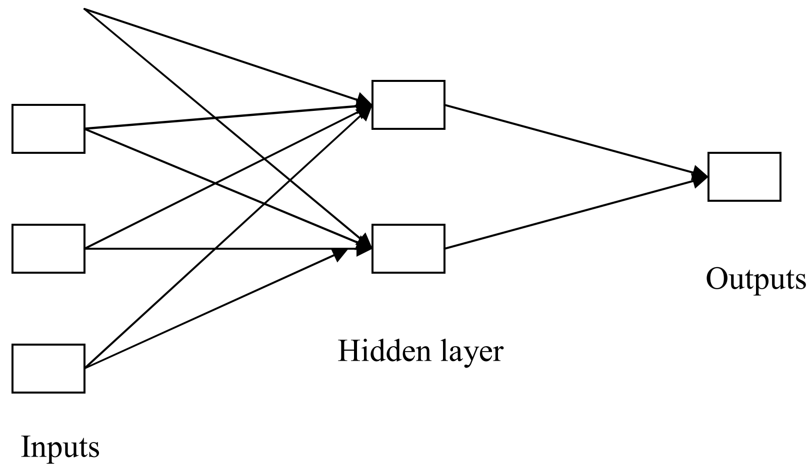


Fig-2.6: Network Layers

This simple type of network is interesting because the hidden units are free to construct their own representations of the input. The weights between the input and hidden units determine when each hidden unit is active, and so by modifying these weights, a hidden unit can choose what it represents.

We also distinguish **single-layer** and **multi-layer** architectures. The single-layer organization, in which all units are connected to one another, constitutes the most general case and is of more potential computational power than hierarchically structured multi-layer organizations. In multi-layer networks, layer, instead of following a global numbering, often numbers units.

2.7.2 FEED-FORWARD NETWORKS

Feed-forward Ann's allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer. Feed-forward Ann's tend to be straightforward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organization is also referred to as bottom-up or top-down.

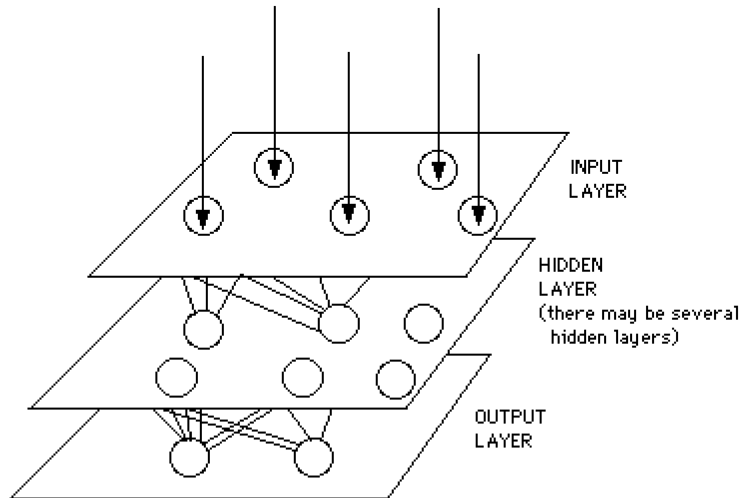


Fig -2.7: Feed Forward Neural Network

2.7.3 FEEDBACK NETWORKS

Feedback networks can have signals traveling in both directions by introducing loops in the network. Feedback networks are very powerful and can get extremely complicated. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent, although the latter term is often used to denote feedback connections in single-layer organizations.

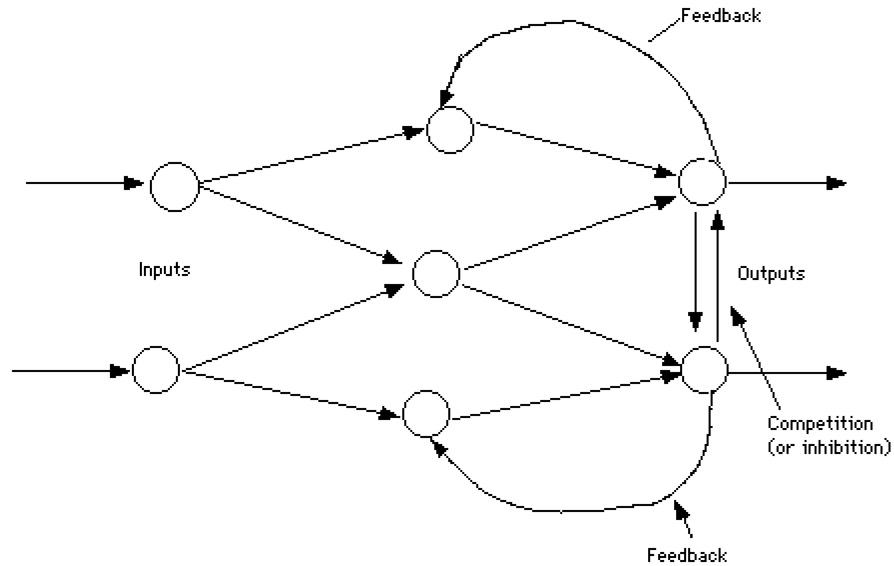


Fig- 2.8: Feedback Neural Network

2.8 PERCEPTRONS

The perceptron calculates a weighted sum of inputs and compares it to a threshold. If the sum is higher than the threshold, the output is set to 1, otherwise to -1, depending upon activation function.

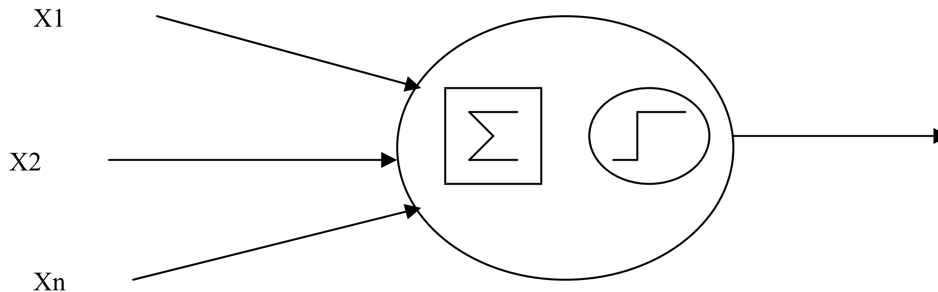


Fig- 2.9: The Mcculloh-Pitts Model

In 1969 Minsky and Papert wrote a book in which they described the limitations of single layer Perceptrons. The impact that the book had was tremendous and caused a lot of neural network researchers to loose their interest. The book was very well written and showed mathematically that single layer perceptrons could not do some basic pattern recognition operations like determining the parity of a shape or determining whether a

shape is connected or not. What they did not realize, until the 80's, is that given the appropriate training, multilevel perceptrons can do these operations.

2.8.1 THE LEARNING PROCESS

The memorization of patterns and the subsequent response of the network can be categorized into two general paradigms:

- **ASSOCIATIVE MAPPING** in which the network learns to produce a particular pattern on the set of output units whenever another particular pattern is applied on the set of input units. The associative mapping can generally be broken down into two mechanisms:

  - **Auto-Association**: an input pattern is associated with itself and the states of input and output units coincide. This is used to provide pattern completion, i.e. to produce a pattern whenever a portion of it or a distorted pattern is presented. In the second case, the network actually stores pairs of patterns building an association between two sets of patterns. [11]
  - **Hetero-Association**: is related to two recall mechanisms
    - **Nearest-neighbor recall**, where the output pattern produced corresponds to the input pattern stored, which is closest to the pattern presented, and
    - **Interpolative recall**, where the output pattern is a similarity dependent interpolation of the patterns stored corresponding to the pattern presented.

- **REGULARITY DETECTION** in which units learn to respond to particular properties of the input patterns. Whereas in associative mapping the network stores the relationships among patterns, in regularity detection the response of each unit has a particular 'meaning'. This type of learning mechanism is essential for feature discovery and knowledge representation.

Every neural network possesses knowledge, which is contained in the values of the connections weights. Modifying the knowledge stored in the network as a function of experience implies a learning rule for changing the values of the weights. Information is

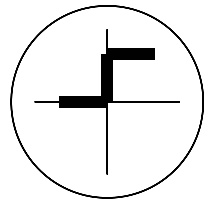
stored in the weight matrix of a neural network. **Learning is the determination of the weights**. Following the way learning is performed; we can distinguish two major categories of neural networks:

- **FIXED NETWORKS** in which the weights cannot be changed, i.e. dW/dt=0. In such networks, the weights are fixed a priori, according to the problem to solve.
- **ADAPTIVE NETWORKS** which are able to change their weights, i.e. dW/dt not= 0. All learning methods used for adaptive neural networks can be classified into two major categories:
  - **Supervised learning** which incorporates an external teacher, so that each output unit is told what its desired response to input signals ought to be. Paradigms of supervised learning include error-correction learning, reinforcement learning and stochastic learning. An 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.
  - **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. Paradigms of unsupervised learning are Hebbian learning and competitive learning.

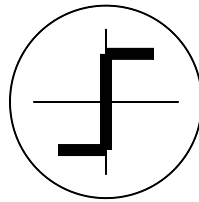
2.8.2 TRANSFER FUNCTION

The behavior of an ANN (Artificial Neural Network) depends on both the weights and the input-output function (transfer function) that is specified for the units. This function typically falls into one of three categories:

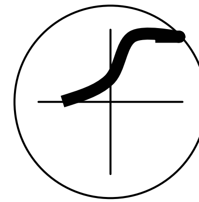
- **Linear (or ramp)**: the output activity is proportional to the total weighted output.
- **Threshold:** the output is set at one of two levels, depending on whether the total input is greater than or less than some threshold value.
- **Sigmoid**: the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurons than do linear or threshold units, but all three must be considered rough approximations.



Step function
Step(x) =1, if
x>=threshold
x<threshold



Sign function
Sign(x) =+1,
if x>=0
Sign (x) = -1,
if x<0



Sigmoid function
Sigmoid(x) =
$1/ (1+e^{x})$

Fig -2.10: Transfer Functions

To make a neural network that performs some specific task, we must choose how the units are connected to one another, and we must set the weights on the connections appropriately. The connections determine whether it is possible for one unit to influence another. The weights specify the strength of the influence. A three-layer network can be taught to perform a particular task by using the following procedure:

- The network is presented with training examples, which consist of a pattern of activities for the input units together with the desired pattern of activities for the output units.
- It is determined how closely the actual output of the network matches the desired output.
- The weight of each connection can be changed so that the network produces a better approximation of the desired output.

2.9 LEARNING ALGORITHMS OF NEURAL NETWORKS

Learning is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes takes place. All learning methods used for neural networks can be classified into two major categories:

- **SUPERVISED LEARNING** which incorporates an external teacher, 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.

  An 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.
- **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. Paradigms of unsupervised learning are Hebbian learning and competitive learning.

2.9.1 SUPERVISED LEARNING

2.9.1.1 THE BACK-PROPAGATION LEARNING

A back propagation neural network uses a feed-forward topology, supervised learning, and back propagation learning algorithm. This algorithm was responsible in large part for the re-emergence of neural networks in the mid 1980s. [7], [8]

Back propagation is a general purpose learning algorithm. It is powerful but also expensive in terms of computational requirements for training. A back propagation network with a single hidden layer of processing elements can model any continuous function to any degree of accuracy (given enough processing elements in the hidden layer).

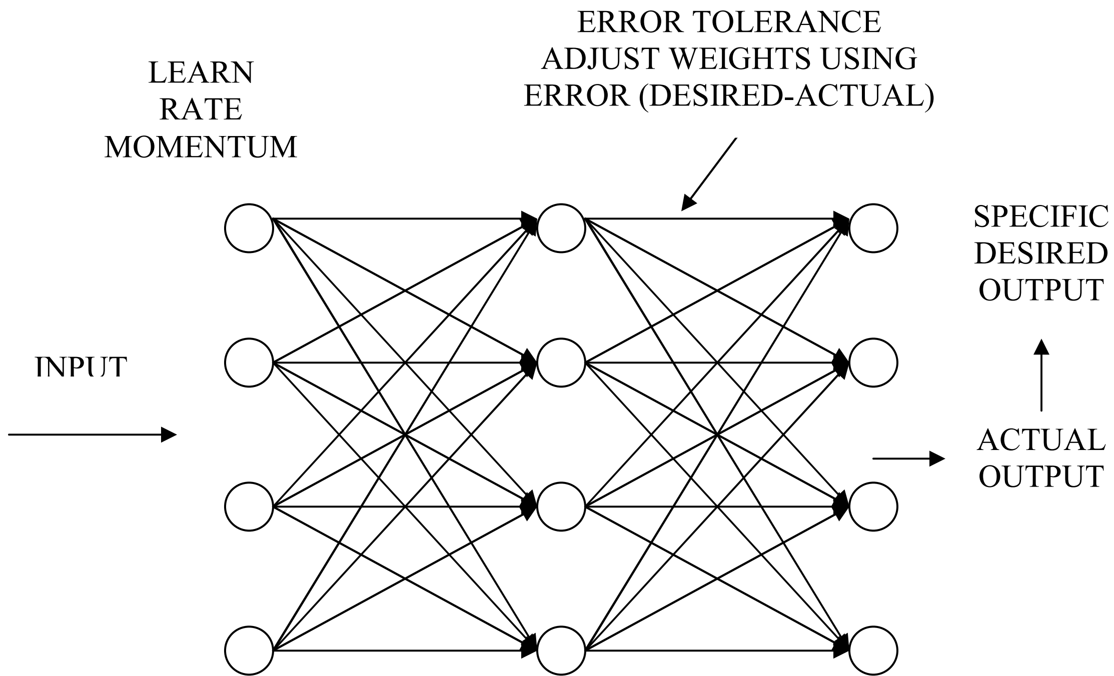


Fig- 2.11: Back Propagation Network

There are literally hundreds of variations of back propagation in the neural network literature, and all claim to be superior to “basic” back propagation in one way or the other. Indeed, since back propagation is based on a relatively simple form of optimization known as gradient descent, mathematically astute observers soon proposed modifications using more powerful techniques such as conjugate gradient and Newton’s methods. However, “basic” back propagation is still the most widely used variant. Its two primary virtues are that it is simple and easy to understand, and it works for a wide range of problems. The basic back propagation algorithm consists of three steps.

- The input pattern is presented to the input layer of the network. These inputs are propagated through the network until they reach the output units. This forward pass produces the actual or predicted output pattern.
- Because back propagation is a supervised learning algorithm, the desired outputs are given as part of the training vector. The actual network outputs are subtracted from the desired outputs and an error signal is produced.
- This error signal is then the basis for the back propagation step, whereby the errors are passed back through the neural network by computing the contribution of each hidden processing unit and deriving the corresponding adjustment needed to produce the correct output. The connection weights are then adjusted and the neural network has just "learned" from an experience.

Two major learning parameters are used to control the training process of a back propagation network. The *learn rate* is used to specify whether the neural network is going to make major adjustments after each learning trial or if it is only going to make minor adjustments. *Momentum* is used to control possible oscillations in the weights, which could be caused by alternately signed error signals. While most commercial back propagation tools provide anywhere from 1 to 10 or more parameters for you to set, these two will usually produce the most impact on the neural network training time and performance.

2.9.1.2 REINFORCEMENT LEARNING

Reinforcement learning is a form of supervised learning because the network gets some feedback from its environment. The feedback signal (yes/no reinforcement signal) is only evaluative, not instructive, i.e. if the reinforcement signal says that a particular output is wrong and it gives no hint as to what the right answer should be. It is therefore important in a reinforcement learning network to implement some source of randomness in the network, so that the space of possible outputs can be explored until a correct value is found. Reinforcement learning is sometimes called "learning with a critic" as opposed to "learning with a teacher" which refers to more traditional learning schemes were an error signal is generated which also contain information in which direction the synaptic

weights of the networks should be changed in order to improve the performance . In reinforcement learning problems it is common to think explicitly of a network functioning in an environment. [4], [6]
The environment supplies the inputs to the network, receives its output and then provides the reinforcement signal.

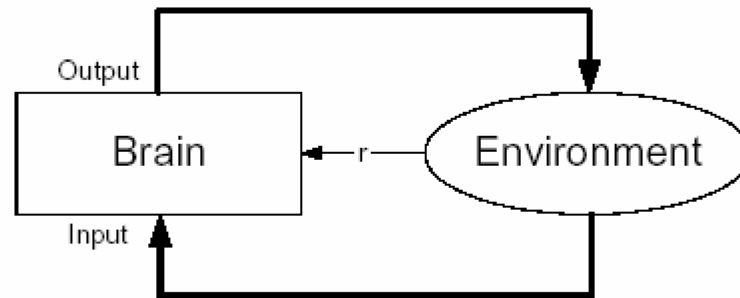


Fig- 2.12: The principle layout for a reinforcement-learning agent

2.9.2 UNSUPERVISED LEARNING

2.9.2.1 HEBBIAN LEARNING

In the year 1949 Donald Hebb postulated a learning rule that states that the connection between two neurons is strengthened if the neurons fire simultaneously (or within a time interval). The Hebbian learning rule specifies how much the weight between two neurons should be increased or decreased in proportion to their activation. [4], [6] If *xj* and *xj* are the activations of neurons *i* and *j*, *wij* is the connection weight between them, and̰̃ the learning rate, then Hebb's rule in its basic form can be written as:

$$\Delta wij = \gamma * x_i * x_j$$

where Δwij is the change of the connection *wij*. Hebbian learning is a type of unsupervised learning that does not consider the result of the output, i.e. it is a correlation-based form of learning. We may expand and rephrase it as a two part rule:

- If two neurons on either side of a synapse are activated simultaneously, then the strength of that synapse is selectively increased.
- If two neurons on either side of synapse are activated asynchronously, then that synapse is selectively weakened or eliminated.

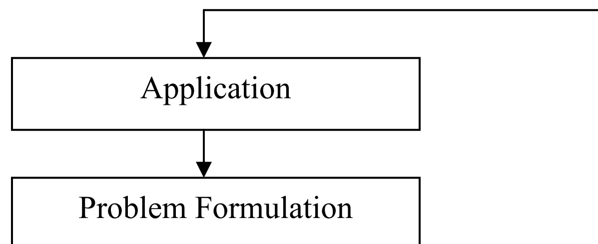
2.9.2.2 COMPETITIVE LEARNING

In competitive learning, the output neurons of a neural network compete among themselves to become active (fired) whereas in a neural network based on hebbian learning several output neurons may be active simultaneously, in competitive learning only a single output neuron is active at only one time. [4], [6] There are three basic elements to a competitive learning rule:

- A set of neurons that are all same except for some randomly distributed synaptic weights, and which therefore respond differently to a given set of input patterns.
- A limit imposed on the strength of each neuron.
- A mechanism that permits the neurons to compete for the right to respond to a given subset of inputs, such that only one output neuron, or only one neuron per group is active at a time. The neuron that wins the competition is called a winner takes all neuron.

2.10 APPLICATIONS OF NEURAL NETWORKS

In order to have an integrated understanding on neural networks, we adopt the next perspective, called top-down, from application, algorithm to architecture:



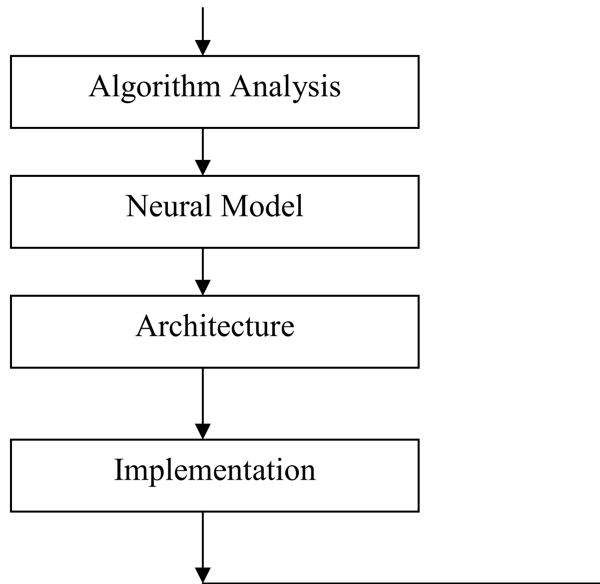


Fig-2.13: Flowchart for Implementation of Applications

The approach is application-motivated, theoretically based, and implementation oriented. The main applications are for signal processing and pattern recognition. The algorithmic treatment represents a combination of mathematical theory and heuristic justification for neural models. The ultimate objective is the implementation of digital neurocomputers, embracing technologies of VLSI, adaptive, digital and parallel processing.

2.10.1 NEURAL NETWORKS IN PRACTICE

Neural networks have broad applicability to real world business problems. In fact, they have already been successfully applied in many industries. Since neural networks are best at identifying patterns or trends in data, they are well suited for prediction or forecasting needs including:

- Sales Forecasting
- Industrial Process Control
- Customer Research
- Data Validation
- Risk Management
- Target Marketing

But to give you some more specific examples; ANN are also used in the following specific paradigms: recognition of speakers in communications; diagnosis of hepatitis; recovery of telecommunications from faulty software; interpretation of multimeaning Chinese words; undersea mine detection; texture analysis; three-dimensional object recognition; hand-written word recognition; and facial recognition.

2.10.2 NEURAL NETWORKS IN MEDICINE

Artificial Neural Networks (ANN) is currently a 'hot' research area in medicine and it is believed that they will receive extensive application to biomedical systems in the next few years. At the moment, the research is mostly on modeling parts of the human body and recognizing diseases from various scans (e.g. cardiograms, CAT scans, ultrasonic scans, etc.). [9], [10]

Neural networks are ideal in recognizing diseases using scans since there is no need to provide a specific algorithm on how to identify the disease. Neural networks learn by example so the details of how to recognize the disease are not needed. What is needed is a set of examples that are representative of all the variations of the disease. The quantity of examples is not as important as the 'quantity'. The examples need to be selected very carefully if the system is to perform reliably and efficiently.

2.10.2.1 MODELING AND DIAGNOSING THE CARDIOVASCULAR SYSTEM

Neural Networks are used experimentally to model the human cardiovascular system. Diagnosis can be achieved by building a model of the cardiovascular system of an individual and comparing it with the real time physiological measurements taken from the patient. If this routine is carried out regularly, potential harmful medical conditions can be detected at an early stage and thus make the process of combating the disease much easier.

A model of an individual's cardiovascular system must mimic the relationship among physiological variables (i.e., heart rate, systolic and diastolic blood pressures, and breathing rate) at different physical activity levels. If a model is adapted to an individual,

then it becomes a model of the physical condition of that individual. The simulator will have to be able to adapt to the features of any individual without the supervision of an expert. This calls for a neural network.

Another reason that justifies the use of ANN technology is the ability of Ann's to provide sensor fusion which is the combining of values from several different sensors. Sensor fusion enables the Ann's to learn complex relationships among the individual sensor values, which would otherwise be lost if the values were individually analyzed. In medical modeling and diagnosis, this implies that even though each sensor in a set may be sensitive only to a specific physiological variable, Ann's are capable of detecting complex medical conditions by fusing the data from the individual biomedical sensors.

2.10.2.2 ELECTRONIC NOSES

Ann's are used experimentally to implement electronic noses. Electronic noses have several potential applications in telemedicine. Telemedicine is the practice of medicine over long distances via a communication link. The electronic nose would identify odors in the remote surgical environment. These identified odors would then be electronically transmitted to another site where a door generation system would recreate them. Because the sense of smell can be an important sense to the surgeon, telesmell would enhance telepresent surgery.

2.10.2.3 INSTANT PHYSICIAN

An application developed in the mid-1980s called the "instant physician" trained an auto associative memory neural network to store a large number of medical records, each of which includes information on symptoms, diagnosis, and treatment for a particular case. After training, the net can be presented with input consisting of a set of symptoms; it will then find the full stored pattern that represents the "best" diagnosis and treatment.

2.10.3 NEURAL NETWORKS IN BUSINESS

Business is a diverted field with several general areas of specializations such as accounting or financial analysis. Almost any neural network application would fit into one business area or financial analysis. There is some potential for using neural networks for business purposes, including resource allocation and scheduling. There is also a strong potential for using neural networks for database mining that is, searching for patterns implicit within the explicitly stored information in databases. Most of the funded work in this area is classified as proprietary. Thus, it is not possible to report on the full extent of the work going on. Most work is applying neural networks, such as the Hopfield-Tank network for optimization and scheduling.

CHAPTER 3

VHSIC HARDWARE DESCRIPTION LANGUAGE

3.1 INTRODUCTION

Hardware description languages are especially useful to gain more control of parallel processes as well as to circumvent some of the idiosyncrasies of the higher level programming languages. For example, the compiler will often add latency to loops during compilation for implementation. This can be difficult to fix in the higher-level languages, though the solution may be quite obvious at the hardware description level. One particularly frustrating peculiarity is the implementation of multipliers. For all multiply commands, the complier requires three multipliers to be used, though typically one is sufficient. The compiler's multipliers also are intended for integers. For a fixed-point design, the decimal point must be moved after every multiply. This is much easier to implement at the hardware description level. [11]

VHDL is a programming language that has been designed and optimized for describing the behavior of digital systems. VHDL has many features appropriate for describing the behavior of electronic components ranging from simple logic gates to complete microprocessors and custom chips. Features of VHDL allow electrical aspects of circuit behavior (such as rise and fall times of signals, delays through gates, and functional operation) to be precisely described. The resulting VHDL simulation models can then be used as building blocks in larger circuits (using schematics, block diagrams or system-level VHDL descriptions) for the purpose of simulation.

VHDL is also a general-purpose programming language: just as high-level programming languages allow complex design concepts to be expressed as computer programs, VHDL allows the behavior of complex electronic circuits to be captured into a design system for automatic circuit synthesis or for system simulation. Like Pascal, C and C++, VHDL includes features useful for structured design techniques, and offers a rich set of control and data representation features. Unlike these other programming languages, VHDL provides features allowing concurrent events to be described. This is important because the hardware described using VHDL is inherently concurrent in its operation.

One of the most important applications of VHDL is to capture the performance specification for a circuit, in the form of what is commonly referred to as a test bench. Test benches are VHDL descriptions of circuit stimuli and corresponding

expected outputs that verify the behavior of a circuit over time. Test benches should be an integral part of any VHDL project and should be created in tandem with other descriptions of the circuit.

One of the most compelling reasons for you to become experienced with and knowledgeable in VHDL is its adoption as a standard in the electronic design community. Using a standard language such as VHDL virtually guarantees that you will not have to throw away and recapture design concepts simply because the design entry method you have chosen is not supported in a newer generation of design tools. Using a standard language also means that you are more likely to be able to take advantage of the most up-to-date design tools and that you will have access to a knowledge base of thousands of other engineers, many of whom are solving problems similar to your own.

3.2 HISTORY OF VHDL

1981 - Initiated by US DoD to address hardware life-cycle crisis

1983-85 - Development of baseline language by Intermetrics, IBM and TI

1986 - All rights transferred to IEEE

1987 - Publication of IEEE Standard

1987 - Mil Std 454 requires comprehensive VHDL descriptions to be delivered with Asics

1994 - Revised standard (named VHDL 1076-1993)

Fig-3.1: History of VHDL

3.3 ADVANTAGES OF VHDL

Why choose to use VHDL for your design efforts? There are many likely reasons. If you ask most VHDL tool vendors this question, the first answer you will get is, "It will improve your productivity." But just what does this mean? Can you really expect to get your projects done faster using VHDL than by using your existing design methods?

The answer is yes, but probably not the first time you use it, and only if you apply VHDL in a structured manner. VHDL (like a structured software design language) is most beneficial when you use a structured, top-down approach to design. Real increases in productivity will come later, when you have climbed higher on the VHDL learning curve and have accumulated a library of reusable VHDL components.

Productivity increases will also occur when you begin to use VHDL to enhance communication between team members and when you take advantage of the more powerful tools for simulation and design verification that are available. In addition, VHDL allows you to design at a more abstract level. Instead of focusing on a gate-level implementation, you can address the behavioral function of the design.

How will VHDL increase your productivity? By making it easy to build and use libraries of commonly used VHDL modules. VHDL makes design reuse feel natural. As you discover the benefits of reusable code, you will soon find yourself thinking of ways to write your VHDL statements in ways that make them general purpose. Writing portable code will become an automatic reflex.

Another important reason to use VHDL is the rapid pace of development in electronic design automation (EDA) tools and in target technologies. Using a standard language such as VHDL can greatly improve your chances of moving into more advanced tools (for example, from a basic low-cost simulator to a more advanced one) without having to re-enter your circuit descriptions. Your ability to retarget circuits to new types of device targets (for example, ASICs, FPGAs, and complex PLDs) will also be improved by using a standard design entry method. [12]

3.4 ENTITIES AND ARCHITECTURES

Every VHDL design description consists of at least one entity/architecture pair. An entity declaration describes the circuit as it appears from the "outside" - from the perspective of its input and output interfaces. If you are familiar with schematics, you might think of the entity declaration as being analogous to a block symbol on a schematic.

The second part of a minimal VHDL design description is the architecture declaration. Before simulation or synthesis can proceed, every referenced entity in a VHDL design description must be bound with a corresponding architecture. The architecture describes the actual function—or contents—of the entity to which it is bound. Using the schematic as a metaphor, you can think of the architecture as being roughly analogous to a lower-level schematic referenced by the higher-level functional block symbol.

3.5 DATA TYPES

Like a high-level software programming language, VHDL allows data to be represented in terms of high-level data types. A data type is an abstract representation of stored data, such as you might encounter in software languages. These data types might represent individual wires in a circuit, or they might represent collections of wires.

The **bit** data type has only two possible values: ‘1’ or ‘0’. (A **bit vector** is simply an array of **bits.**) Every data type in VHDL has a defined set of values, and a defined set of valid operations. Type checking is strict, so it is not possible, for example, to directly assign the value of an **integer** data type to a **bit vector** data type. There are ways to get around this restriction, using what are called type conversion functions.

3.6 DESIGN UNITS

One concept unique to VHDL (when compared to software programming languages and to its main rival, Verilog) is the concept of a design unit. Design units in VHDL (which may also be referred to as library units) are segments of VHDL code that can be compiled separately and stored in a library. [13]

You have been introduced to two design units already: the entity and the architecture. There are actually five types of design units in VHDL; entities, architectures, packages, package bodies, and configurations. Entities and architectures are the only two design units that you must have in any VHDL design description. Packages and configurations are optional.

3.6.1 ENTITIES

A VHDL entity is a statement (indicated by the **entity** keyword) that defines the external specification of a circuit or sub-circuit. The minimum VHDL design description must include at least one entity and one corresponding architecture.

When you write an entity declaration, you must provide a unique name for that entity and a port list defining the input and output ports of the circuit. Each port in the port list must be given a name, direction (or mode, in VHDL jargon) and a type. Optionally, you may also include a special type of parameter list (called a generic list) that allows you to pass additional information into an entity.

3.6.2 ARCHITECTURES

A VHDL architecture declaration is a statement (beginning with the **architecture** keyword) that describes the underlying function and/or structure of a circuit. Each architecture in your design must be associated (or bound) by name with one entity in the design.

VHDL allows you to create more than one alternate architecture for each entity. This feature is particularly useful for simulation and for project team environments in which the design of the system interfaces (expressed as entities) is performed by a different engineer than the lower-level architectural description of each component circuit, or when you simply want to experiment with different methods of description.

An architecture declaration consists of zero or more declarations (of items such as intermediate signals, components that will be referenced in the architecture, local functions and procedures, and constants) followed by a **begin** statement, a series of concurrent statements, and an **end** statement.

3.6.3 PACKAGES AND PACKAGE BODIES

A VHDL package declaration is identified by the package keyword, and is used to collect commonly used declarations for use globally among different design units. You can think of a package as a common storage area, one used to store such things as type declarations, constants, and global subprograms. Items defined within a package can be

made visible to any other design unit in the complete VHDL design, and they can be compiled into libraries for later re-use.

A package can consist of two basic parts: a package declaration and an optional package body. Package declarations can contain the following types of statements:

- Type and subtype declarations
- Constant declarations
- Global signal declarations
- Function and procedure declarations
- Attribute specifications
- File declarations
- Component declarations
- Alias declarations
- Disconnect specifications
- Use clauses

Items appearing within a package declaration can be made visible to other design units through the use of a **use** statement.

If the package contains declarations of subprograms (functions or procedures) or defines one or more deferred constants (constants whose value is not immediately given), then a package body is required in addition to the package declaration. A package body (which is specified using the **package body** keyword combination) must have the same name as its corresponding package declaration, but it can be located anywhere in the design, in the same or a different source file.

3.6.4 CONFIGURATIONS

The final type of design unit available in VHDL is called a configuration declaration. You can think of a configuration declaration as being roughly analogous to a parts list for your design. A configuration declaration (identified with the **configuration** keyword) specifies which architectures are to be bound to which entities, and it allows

you to change how components are connected in your design description at the time of simulation.

Configuration declarations are always optional, no matter how complex a design description you create. In the absence of a configuration declaration, the VHDL standard specifies a set of rules that provide you with a default configuration. For example, in the case where you have provided more than one architecture for an entity, the last architecture compiled will take precedence and will be bound to the entity.

3.7 LEVELS OF ABSTRACTION

VHDL supports many possible styles of design description. These styles differ primarily in how closely they relate to the underlying hardware. When we speak of the different styles of VHDL, we are really talking about the differing levels of abstraction possible using the language—behavior, dataflow, and structure. [14]

Suppose the performance specifications for a given project are: "the compressed data coming out of the DSP chip needs to be analyzed and stored within 70 nanoseconds of the strobe signal being asserted..." This human language specification must be refined into a description that can actually be simulated. A test bench written in combination with a sequential description is one such expression of the design. These are all points in the **Behavior** level of abstraction.

After this initial simulation, the design must be further refined until the description is something a VHDL synthesis tool can digest. Synthesis is a process of translating an abstract concept into a less-abstract form. The highest level of abstraction accepted by today's synthesis tools is the **Dataflow** level.

The **Structure** level of abstraction comes into play when little chunks of circuitry are to be connected together to form bigger circuits. (If the little chunks being connected are actually quite large chunks, then the result is what we commonly call a block

diagram.) Physical information is the most basic level of all and is outside the scope of VHDL. This level involves actually specifying the interconnects of transistors on a chip, placing and routing macro cells within a gate array or FPGA, etc.

3.7.1 BEHAVIOR

The highest level of abstraction supported in VHDL is called the behavioral level of abstraction. When creating a behavioral description of a circuit, you will describe your circuit in terms of its operation over time. The concept of time is the critical distinction between behavioral descriptions of circuits and lower-level descriptions (specifically descriptions created at the dataflow level of abstraction).

In a behavioral description, the concept of time may be expressed precisely, with actual delays between related events (such as the propagation delays within gates and on wires), or it may simply be an ordering of operations that are expressed sequentially (such as in a functional description of a flip-flop). When you are writing VHDL for input to synthesis tools, you may use behavioral statements in VHDL to imply that there are registers in your circuit. It is unlikely, however, that your synthesis tool will be capable of creating precisely the same behavior in actual circuitry as you have defined in the language. (Synthesis tools today ignore detailed timing specifications, leaving the actual timing results at the mercy of the target device technology.) It is also unlikely that your synthesis tool will be capable of accepting and processing a very wide range of behavioral description styles.

3.7.2 DATAFLOW

In the dataflow level of abstraction, you describe your circuit in terms of how data moves through the system. At the heart of most digital systems today are registers, so in the dataflow level of abstraction you describe how information is passed between registers in the circuit. You will probably describe the combinational logic portion of your circuit at a

relatively high level (and let a synthesis tool figure out the detailed implementation in logic gates), but you will likely be quite specific about the placement and operation of registers in the complete circuit. The dataflow level of abstraction is often called register transfer logic, or RTL.

There are some drawbacks to using a dataflow method of design in VHDL. First, there are no built-in registers in VHDL; the language was designed to be general-purpose, and VHDL's designers on its behavioral aspects placed the emphasis. If you are going to write VHDL at the dataflow level of abstraction, you must first create (or obtain) behavioral descriptions of the register elements you will be using in your design. These elements must be provided in the form of components (using VHDL's hierarchy features) or in the form of subprograms (functions or procedures).

3.7.3 STRUCTURE

The third level of abstraction, structure, is used to describe a circuit in terms of its components. Structure can be used to create a very low-level description of a circuit (such as a transistor-level description) or a very high-level description (such as a block diagram).

In a gate-level description of a circuit, for example, components such as basic logic gates and flip-flops might be connected in some logical structure to create the circuit. This is what is often called a net list. For a higher-level circuit—one in which the components being connected are larger functional blocks—structure might simply be used to segment the design description into manageable parts.

Structure-level VHDL features, such as components and configurations, are very useful for managing complexity. The use of components can dramatically improve your ability to re-use elements of your designs, and they can make it possible to work using a top-down design approach.

3.8 OBJECTS, DATA TYPES AND OPERATORS

VHDL includes a number of language elements, collectively called objects that can be used to represent and store data in the system being described. The three basic types of objects that you will use when entering a design description for synthesis or creating functional tests (in the form of a test bench) are signals, variables and constants. Each object that you declare has a specific data type (such as **bit** or **integer**) and a unique set of possible values. [15]

3.8.1 USING SIGNALS

Signals are objects that are used to connect concurrent elements (such as components, processes and concurrent assignments), similar to the way that wires are used to connect components on a circuit board or in a schematic. Signals can be declared globally in an external package or locally within architecture, block or other declarative region.

3.8.2 USING VARIABLES

Variables are objects used to store intermediate values between sequential VHDL statements. Variables are only allowed in processes, procedures and functions, and they are always local to those functions. The 1076-1993 language standard adds a new type of global variable that has visibility between different processes and subprograms. Variables in VHDL are much like variables in a conventional software programming language. They immediately take on and store the value assigned to them and they can be used to simplify a complex calculation or sequence of logical operations.

Chapter 4

BACK PROPAGATION ALGORITHM

4.1 INTRODUCTION

Back propagation is a form of supervised learning for multi-layer nets, also known as the generalized delta rule. Error data at the output layer is back propagated to earlier ones, allowing incoming weights to these layers to be updated. It is most often used as training algorithm in current neural network applications. The back propagation algorithm was developed by Paul Werbos in 1974 and rediscovered independently by Rumelhart and Parker. Since its rediscovery, the back propagation algorithm has been widely used as a learning algorithm in feed forward multilayer neural networks. [7]

4.2 HISTORY OF ALGORITHM

Minsky and Papert (1969) showed that there are many simple problems such as the exclusive-or problem, which linear neural networks cannot solve. Note that term "solve" means learn the desired associative links. Argument is that if such networks cannot solve such simple problems how could they solve complex problems in vision, language, and motor control. Solutions to this problem were:

- Select appropriate "recoding" scheme which transforms inputs
- Perceptron Learning Rule -- Requires that you correctly "guess" an acceptable input to hidden unit mapping.
- Back-propagation learning rule -- Learn both sets of weights simultaneously. [5]

4.3 LEARNING WITH THE BACK PROPAGATION ALGORITHM

The back propagation algorithm is an involved mathematical tool; however, execution of the training equations is based on iterative processes, and thus is easily implement able on a computer. [8]

- Weight changes for hidden to output weights just like Widrow-Hoff learning rule.
- Weight changes for input to hidden weights just like Widrow-Hoff learning rule but error signal is obtained by "back-propagating" error from the output units.

During the training session of the network, a pair of patterns is presented ($X_{k,}$ T_k), where X_k in the input pattern and T_k is the target or desired pattern. The X_k pattern causes output responses at teach neurons in each layer and, hence, an output O_k at the output layer. At the output layer, the difference between the actual and target outputs yields an error signal. This error signal depends on the values of the weights of the neurons I each layer. This error is minimized, and during this process new values for the weights are obtained. The speed and accuracy of the learning process-that is, the process of updating the weights-also depends on a factor, known as the learning rate.

Before starting the back propagation learning process, we need the following:

- The set of training patterns, input, and target
- A value for the learning rate
- A criterion that terminates the algorithm
- A methodology for updating weights
- The nonlinearity function (usually the sigmoid)
- Initial weight values (typically small random values)

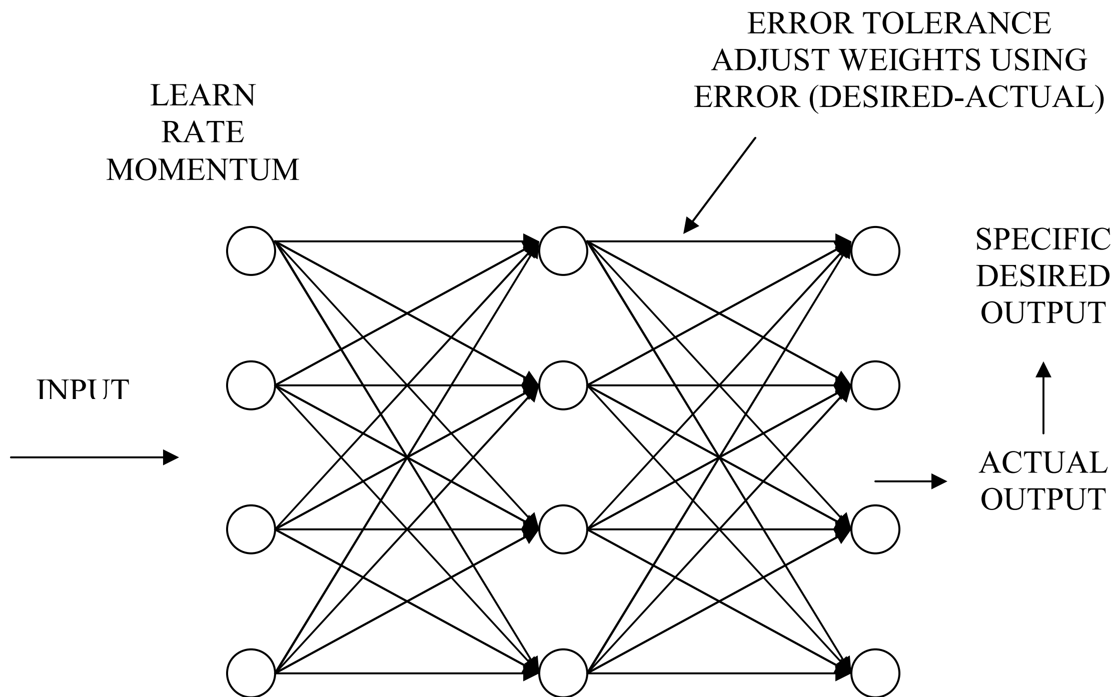


Fig-4.1: Back Propagation Network

The process then starts by applying the first input pattern X_k and the corresponding target output T_k. The input causes a response to the neurons of the first layer, which in turn cause a response to the neurons of the next layer, and so on, until a response is obtained at the output layer. That response is then compared with the target response; and the difference (the error signal) is calculated. From the error difference at the output neurons, the algorithm computes the rate at which the error changes as the activity level of the neuron changes. So far, the calculations were computed forward (i.e., from the input layer to the output layer). Now, the algorithm steps back one layer before that output layer and recalculate the weights of the output layer (the weights between the last hidden layer and the neurons of the output layer) so that the output error is minimized. The algorithm next calculates the error output at the last hidden layer and computes new values for its weights (the weights between the last and next-to-last hidden layers). The algorithm continues calculating the error and computing new weight values, moving layer by layer backward, toward the input.

When the input is reached and the weights do not change, (i.e., when they have reached a steady state), then the algorithm selects the next pair of input-target patterns and repeats the process. Although responses move in a forward direction, weights are calculated by moving backward, hence the name back propagation.

4.4 IMPLEMENTATION OF BACK PROPAGATION ALGORITHM

The back-propagation algorithm consists of the following steps: [16], [17]

- Each Input is then multiplied by a weight that would either inhibit the input or excite the input. The weighted sum of then inputs in then calculated

First, it computes the total weighted input *Xj,* using the formula:

$$X_j = \sum_i y_i W_{ij} \tag{4.1}$$

Where *yi* is the activity level of the jth unit in the previous layer and *Wij* is the weight of the connection between the ith and the jth unit.

Then the weighed *Xj* is passed through a sigmoid function that would scale the output in between a 0 and 1.

- Next, the unit calculates the activity *yj* using some function of the total weighted input. Typically we use the sigmoid function:

$$y_j = \frac{1}{1 + e^{-x_j}} \tag{4.2}$$

Once the output is calculated, it is compared with the required output and the total Error E is computed.

- Once the activities of all output units have been determined, the network computes the error E, which is defined by the expression:

$$E = \frac{1}{2}\sum_{j} (y_j - d_j)^2 \tag{4.3}$$

where y_j is the activity level of the *ith* unit in the top layer and dj is the desired output of the *ith* unit.

Now the error is propagated backwards. [18], [19]

1. Compute how fast the error changes as the activity of an output unit is changed. This error derivative (*EA*) is the difference between the actual and the desired activity.

$$EA_j = \frac{\partial E}{\partial y_j} = y_j - d_j \tag{4.4}$$

2. Compute how fast the error changes as the total input received by an output unit is changed. This quantity (*EI*) is the answer from step 1 multiplied by the rate at which the output of a unit changes as its total input is changed.

$$EI_j = \frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \times \frac{dy_j}{dx_j} = EA_j y_j (1 - y_j) \tag{4.5}$$

3. Compute how fast the error changes as a weight on the connection into an output unit is changed. This quantity (*EW*) is the answer from step 2 multiplied by the activity level of the unit from which the connection emanates.

$$EW_{ij} = \frac{\partial E}{dW_{ij}} = \frac{\partial E}{\partial x_j} \times \frac{\partial x_j}{dW_{ij}} = EI_j y_i \tag{4.6}$$

4. Compute how fast the error changes as the activity of a unit in the previous layer is changed. This crucial step allows back propagation to be applied to multi-layer networks. When the activity of a unit in the previous layer changes, it affects the activities of all the output units to which it is connected. So to compute the overall effect on the error, we add together all these separate effects on output units. But each effect is simple to calculate. It is the answer in step 2 multiplied by the weight on the connection to that output unit.

$$EA_i = \frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j} \times \frac{\partial x_j}{\partial y_i} = \sum_j EI_j W_{ij} \tag{4.7}$$

By using steps 2 and 4, we can convert the *EA's* of one layer of units into *EA*'s for the previous layer. This procedure can be repeated to get the *EA's* for as many previous layers as desired. Once we know the *EA* of a unit, we can use steps 2 and 3 to compute the *EW's* on its incoming connections.

4.5 DRAWBACKS

Although widely used, the back propagation algorithm has not escaped criticism. The method of backwards-calculating weights does not seem to be biologically plausible; neurons do not seem to work backward to adjust the efficacy of their synaptic weights. Thus, the back propagation-learning algorithm is not viewed by many as a learning process that emulates the biological world but as a method to design a network with learning.

Second, the algorithm uses a digital computer to calculate weights. When the final network is implemented in hardware, however, it has lost its plasticity. This loss is in contrast with the initial motivation to develop neural networks that emulate brain like networks and are adaptable (plastic) enough to learn new patterns. If changes are necessary, a computer calculates anew the weight values and updates them. This means that the neural network implementation still depends on a digital computer.

The algorithm suffers from extensive calculations and, hence, slows training speed. The time required to calculate the error derivatives and to update the weights on a given training exemplar is proportional to the size of the network. The amount of computation is proportional to the number of weights. In large networks, increasing the number of training patterns causes the learning time to increase faster that the network. The computational speed inefficiency of this algorithm has triggered an effort to explore techniques that accelerated the learning time by at least a factor of 2. Even these accelerated techniques, however, do not make the back propagation learning algorithm suitable in many real time applications. [20]

Despite it wide applicability, the error back propagation algorithm cannot be applied to all neural network systems which can be imagined. In particular, the algorithm requires that the activation functions of each of the neurons in the network be both continuous and differentiable. Several historically important neural network architectures use activation functions which do not satisfy this condition: these include the discontinuous linear threshold activation function of the original perceptron of Rosenblatt and the continuous but non-differentiable linear ramp activation function of the units in the brain-state-in-e-box model of Anderson et al.

Chapter 5

HARDWARE IMPLEMENTATION

5.1 INTRODUCTION

In this thesis I have proposed an expandable on-chip back-propagation (BP) learning neural network. The network has four neurons and 16 synapses. Large-scale neural networks with arbitrary layers and discretional neurons per layer can be constructed by combining a certain number of such unit networks. A novel neuron circuit with programmable parameters, which generates not only the sigmoid function but also its derivative, is proposed. [16]

Recently, in the field of neural networks, great attention has been paid to on-chip learning neural networks, especially in those situations when the system needs high speed, small volume and reduced weight. The back-propagation (BP) algorithm often provides a practical approach to a wide range of problems. There have been many examples of implementations of general-purpose neural networks with on-chip BP learning, Different situations may need neural networks of different scales. Thus general purpose neural networks are required to be expandable. [21] The neuron is one of the most important components in an artificial neural network. It is used to realize nonlinear transformation. In much of the literature, the nonlinear function is found to be the sigmoid. In on-chip back-propagation learning, both a nonlinear function and its derivative are required. It can realize the sigmoid function with programmable threshold and gain factor. [22]

5.2 CHIP ARCHITECTURE

Figure 5.1 shows the circuit architecture of the expandable on-chip learning BP network. It consists of a neuron array including ‘N’s, a synapse array including ‘S’s and an error generator array including ‘E’s. Large-scale neural networks with arbitrary layers and discretional neurons per layer can be constructed by combining a certain number of such unit networks. [23], [24], [25]

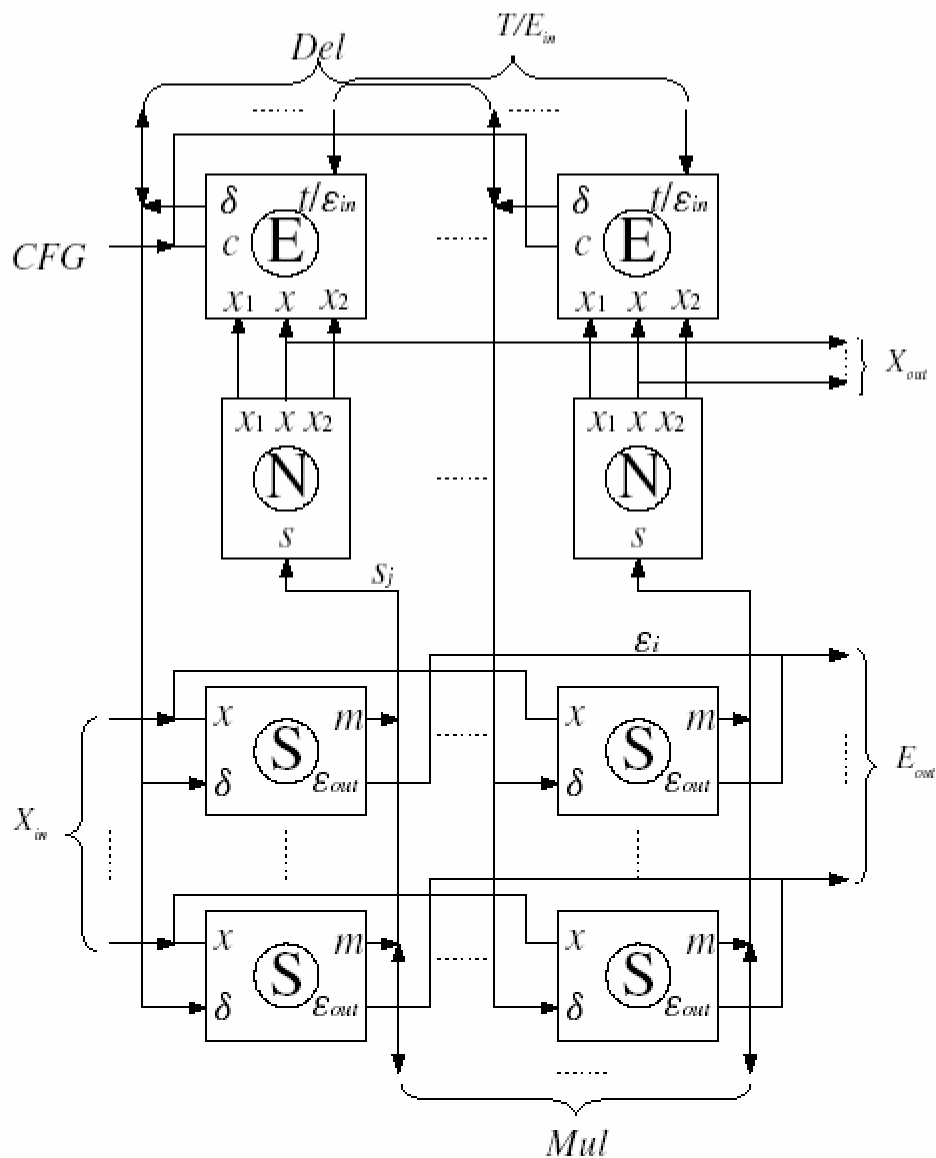


Fig-5.1: Circuit architecture of the prototype chip

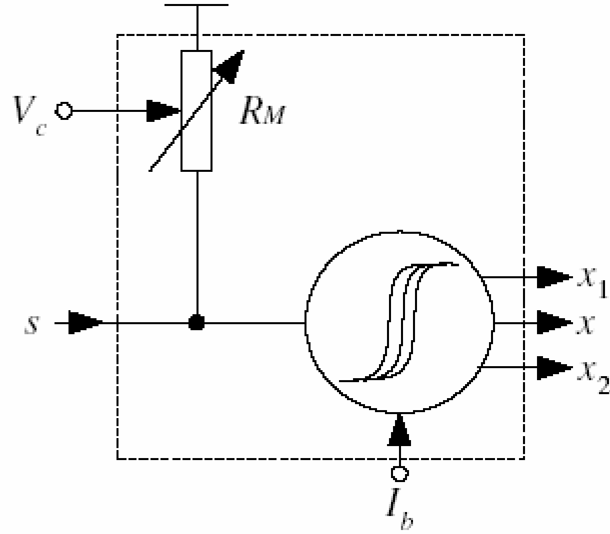


Fig-5.2: Block diagram of neuron

The block diagram of the neuron is shown in the figure 5.2. [26], [27], [28] It transforms a current input *s* to a voltage output *x* according to a sigmoid function with programmable gain and threshold. The sigmoid function can be expressed by

$$f(s) = \frac{1}{1+e^{-\alpha(s+\theta)}} \tag{5.1}$$

where α is the gain factor, θ is the threshold and *s* is the sum of the weighted inputs. [29]

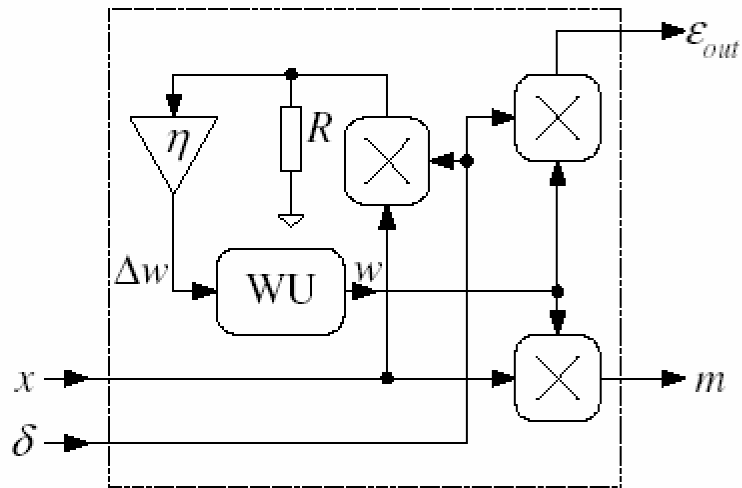


Fig-5.3: Block diagram of synapse

The synapse block shown in the figure performs three functions: [30], [31]

- Multiplying a weight *w* by a weight error signal to generate a signal $\varepsilon\, out$, which is connected to a summing line ε i in the current domain to get a neuron error signal

$$\varepsilon_i = \Sigma_j \ W_{ij} \delta_{ij} \tag{5.2}$$

- Multiplying a weight w by an input *x* to generate a signal *m*, which is connected to a summing line *sj* in the current domain to get a signal

$$S_j = \Sigma_i \ W_{ij} * X_I \tag{5.3}$$

- Updating a weight *w* by a weight unit (WU).

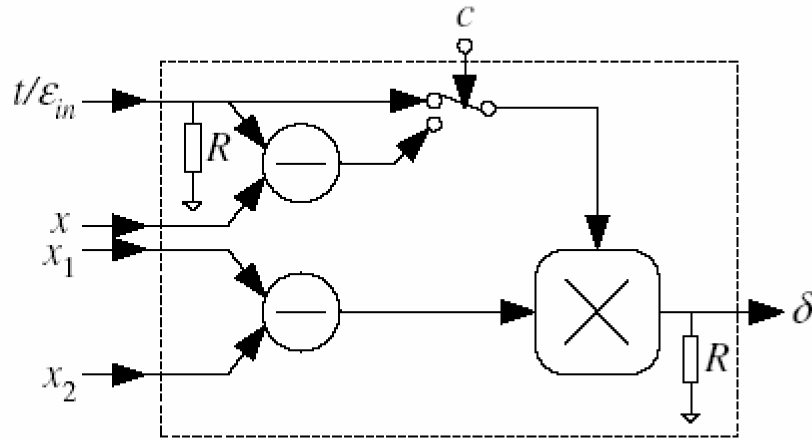


Fig-5.4: Block diagram of error generator

The block diagram of the error generator unit in shown in figure 5.4. It provides a weight error signal δ. The control signal *c* decides whether the corresponding neuron is an output neuron. t/ε in is a twofold port.

- If *c*=1, a target value *t* is imported to the t/ε in port and δ is obtained by multiplying *(t-x)* by *(x1-x2);*

- If *c*=0, a neuron error value ε in is imported to the t/ε in port and δ is achieved by multiplying ε in by (x1-x2). So no additional output chips are needed to construct a whole on-chip learning system.

5.3 DETAILED ANALYSIS OF THE BLOCK DIAGRAM AND HARDWARE IMPLEMENTATION

Detailed block diagram of back propagation basically consists of six blocks named synapse, neuron, error generator output; error generator input, weight update and weight transfer units. Each of these blocks has been modeled as individual entities using behavioral modeling in vhdl. In each entity a start and a stop signal has been included to take care of timing constraints which are of importance when these entities are finally

port mapped into the final entity which uses structural modeling. Detailed block diagram for back propagation algorithm is as follows:

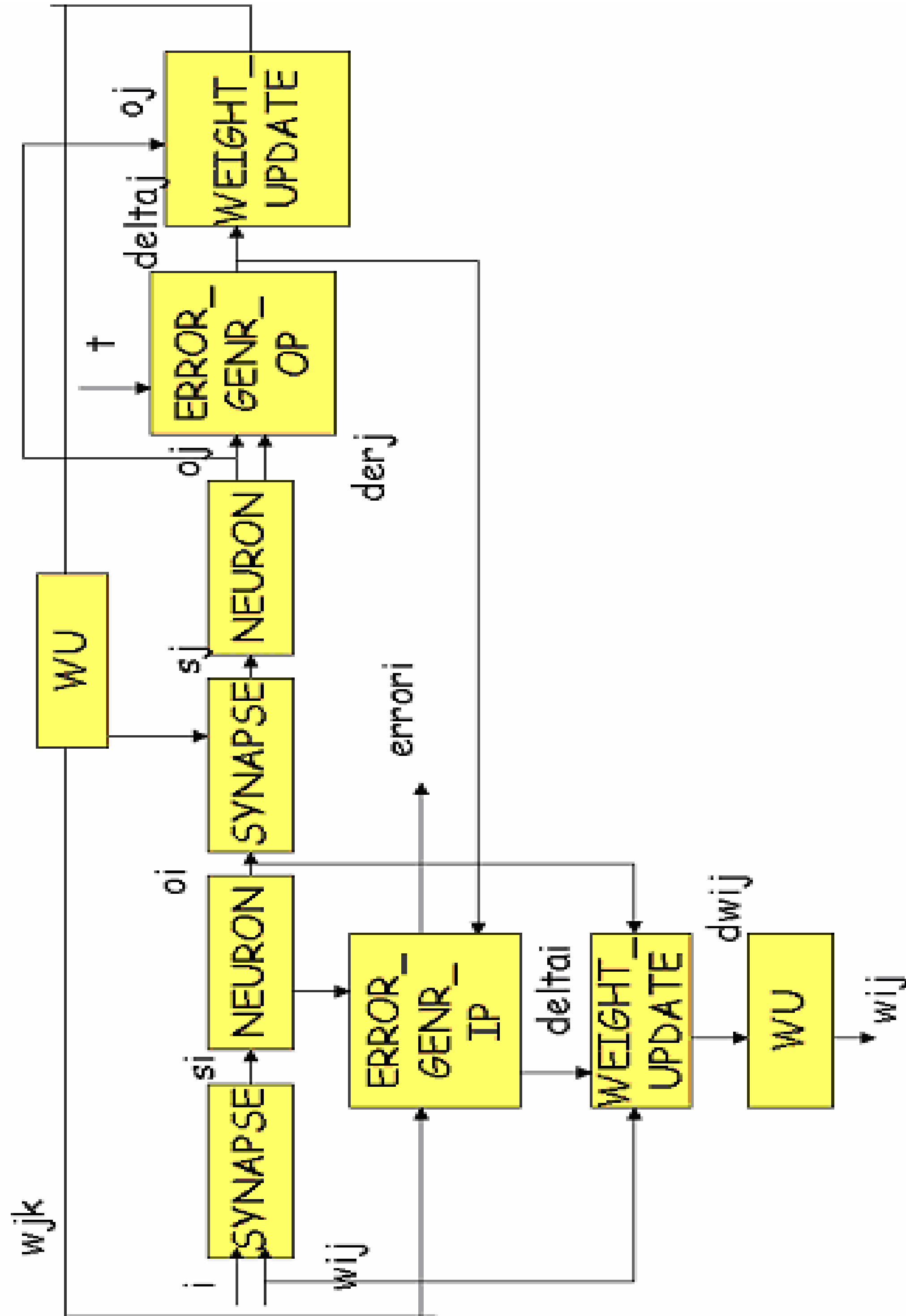


Fig-5.5: Block diagram for Back Propagation

Table 5.1: Table of signals for Back Propagation algorithm

NAME OF SIGNAL	DESCRIPTION OF SIGNAL
i	an array of 4 elements i.e. input to the neural network
wij	weights at the input layer
si	sum of weighted inputs, from synapse at input layer
oi	output of first neuron based on sigmoid function
wjk	weights at the output layer
sj	sum of weighted inputs, from the second neuron
oj	output of second neuron based on sigmoid function
t	an array of 4 elements i.e. the target for which the neural network has to be trained
der , derj	derivative of outputs of the first and second neuron, respectively
deltaj	calculated errors (delta functions) obtained by comparing the output of the second neuron with the target value to be achieved
deltai	calculated errors (delta functions) obtained from error generator at the input
errori	error from the error generator at the input
dwij	updated weight values at the input layer
dwjk	updated weight values at the output layer

5.4 BRIEF DESCRIPTION OF THE ENTITIES

5.4.1 SYNAPSE

Synapses are the elementary structural and functional units that mediate the interactions between the neurons. In electrical descriptions of neural organization, it is

assumed that a synapse is a simple connection that can impose excitation or inhibition, but not both on the receptive neuron.

In the entity synapse inputs i.e. i1, i2, i3, i4 are multiplied by the corresponding weights (w11, w12, w13, w14, w21, w22, w23, w24, w31, w32, w33, w34, w41, w42, w43, w44) and the weighted inputs are summed together to get 4 outputs i.e. si1, si2, si3, si4.

$$s_i = w_{ij} \times i \tag{5.4}$$

5.4.2 NEURON

It transforms a current input s to a voltage output x according to a sigmoid function with programmable gain and threshold. The sigmoid function can be expressed by:

$$f(s) = \frac{1}{1+e^{-\alpha(s+\theta)}} \tag{5.5}$$

where α the gain factor is θ is the threshold and s is the sum of the weighted inputs. Input to neuron are the summed signals of weighted inputs i.e. si1, si2, si3, si4, which gives the output as oi1, oi2, oi3, oi4 according to the sigmoid function.

Further, the synapse and neuron are followed by another synapse and a neuron to give the output as sj1, sj2, sj3, sj4 and oj1, oj2, oj3, oj4, respectively. Also, the derivative of the output from neuron i.e. oj1, oj2, oj3, oj4 has been calculated as deri1, deri2, deri3, deri4.

5.4.3 ERROR GENERATOR AT THE OUTPUT

Error generator at the output calculates the error by subtracting the actual output from the desired one. This difference is then multiplied with the derivative of the output (derj1, derj2, derj3, derj4) from neuron to get a delta function i.e. deltaj1, deltaj2, deltaj3, deltaj3, deltaj4.

$$der_j = o_j \times (1 - o_j) \tag{5.6}$$

$$delta_j = (t - o_j) \times der_j \tag{5.7}$$

5.4.4 WEIGHT UPDATE

Weight update entity updates the weight by using the delta function and the output of the neuron.

$$dw_{ij} = w_{ij} - delta_i \times o_i \tag{5.8}$$

$$dw_{jk} = w_{jk} - delta_j \times o_j \tag{5.9}$$

5.4.5 WEIGHT TRANSFER UNIT

Weight transfer unit transfers the updated weight values to the original weights which are then used in the neural network training.

$$w_{ij} = dw_{ij} \tag{5.10}$$

$$w_{jk} = dw_{ij} \tag{5.11}$$

5.4.6 ERROR GENERATOR AT THE INPUT

Error generator at the input calculates the error by subtracting the actual output from the desired one. This difference is then multiplied with the derivative of the output from neuron to get a delta function i.e. deltai1, deltai2, deltai3, deltai3, deltai4.

$$der_i = o_i \times (1 - o_i) \tag{5.12}$$

$$delta_i = (t - o_i) \times der_i \tag{5.13}$$

$$error_i = w_{jk} \times delta_j \tag{5.14}$$

All the above blocks have been modeled in vhdl using behavioral modeling and then a finally a structural model has been modeled which port maps all these entities within it.

CHAPTER 6

SIMULATION RESULTS

6.1 SYNAPSE

6.1.1 INPUTS AND OUTPUTS FOR THE ENTITY SYNAPSE – S1 IN THE BLOCK DIAGRAM

INPUTS

w11, w12, w13, w14, w21, w22, w23, w24, w31, w32, w33, w34, w41, w42, w43, w44 are the input weights,
i1, i2, i3, i4 are the input signals,
start, m are the input signals introduced to take care of timing constraints during port mapping.

OUTPUTS

s1, s2, s3, s4 are the sum of weighted inputs,
m11, m12, m13, m14, m21, m22, m23, m24, m31, m32, m33, m34, m41, m42, m43, m44 are the intermediate signals obtained by multiplying inputs and the corresponding weights
done is the output signals introduced to take care of timing constraints during port mapping.

6.1.2 RESULTS FOR THE ENTITY SYNAPSE – S1

Signal	Value	Waveform
/synapse/w11	0.01	0.01
/synapse/w12	0.01	0.01
/synapse/w13	0.01	0.01
/synapse/w14	0.01	0.01
/synapse/w21	0.01	0.01
/synapse/w22	0.01	0.01
/synapse/w23	0.01	0.01
/synapse/w24	0.01	0.01
/synapse/w31	0.01	0.01
/synapse/w32	0.01	0.01
/synapse/w33	0.01	0.01
/synapse/w34	0.01	0.01
/synapse/w41	0.01	0.01
/synapse/w42	0.01	0.01
/synapse/w43	0.01	0.01
/synapse/w44	0.01	0.01
/synapse/i1	0.1	0.1
/synapse/i2	0.1	0.1
/synapse/i3	0.1	0.1
/synapse/i4	0.1	0.1
/synapse/s1	0.004	0.004
/synapse/s2	0.004	0.004
/synapse/s3	0.004	0.004
/synapse/s4	0.004	0.004
/synapse/start	1	
/synapse/done	1	
/synapse/m11	0.001	0.001

Fig-6.1: Simulation results for synapse

6.2 NEURON

6.2.1 INPUTS AND OUTPUTS FOR THE ENTITY NEURON – N1

INPUTS

s1, s2, s3, s4 are the sum of weighted inputs,
start is the input signal introduced to take care of timing constraints during portmapping.

OUTPUTS

o1, o2, o3, o4 are the outputs based on sigmoid function,
d1, d2, d3, d4 are the derivatives of the sigmoid function
done is the output signal introduced to take care of timing constraints during portmapping.

6.2.2 RESULTS FOR THE ENTITY NEURON – n1

Signal	Value	Waveform
/neuron/s1	0.004	0.004
/neuron/s2	0.004	0.004
/neuron/s3	0.004	0.004
/neuron/s4	0.004	0.004
/neuron/o1	0.501	0.501
/neuron/o2	0.501	0.501
/neuron/o3	0.501	0.501
/neuron/o4	0.501	0.501
/neuron/d1	0.249999	0.249999
/neuron/d2	0.249999	0.249999
/neuron/d3	0.249999	0.249999
/neuron/d4	0.249999	0.249999
/neuron/start	1	
/neuron/done	1	

Fig-6.2: Simulation results for neuron

6.3 SYNAPSE

6.3.1 INPUTS AND OUTPUTS FOR THE ENTITY SYNAPSE – s2

INPUTS

w11, w12, w13, w14, w21, w22, w23, w24, w31, w32, w33, w34, w41, w42, w43, w44 are the input weights,
i1, i2, i3, i4 are the input signals,
start, m are the input signals introduced to take care of timing constraints during portmapping.

OUTPUTS

s1, s2, s3, s4 are the sum of weighted inputs,
m11, m12, m13, m14, m21, m22, m23, m24, m31, m32, m33, m34, m41, m42, m43, m44 are the intermediate signals obtained by multiplying inputs and the corresponding weights
done is the output signals introduced to take care of timing constraints during portmappnig.

6.3.2 RESULTS FOR THE ENTITY SYNAPSE -S2

Signal	Value	Wave
/synapse/w11	0.01	0.01
/synapse/w12	0.01	0.01
/synapse/w13	0.01	0.01
/synapse/w14	0.01	0.01
/synapse/w21	0.01	0.01
/synapse/w22	0.01	0.01
/synapse/w23	0.01	0.01
/synapse/w24	0.01	0.01
/synapse/w31	0.01	0.01
/synapse/w32	0.01	0.01
/synapse/w33	0.01	0.01
/synapse/w34	0.01	0.01
/synapse/w41	0.01	0.01
/synapse/w42	0.01	0.01
/synapse/w43	0.01	0.01
/synapse/w44	0.01	0.01
/synapse/i1	0.501	0.501
/synapse/i2	0.501	0.501
/synapse/i3	0.501	0.501
/synapse/i4	0.501	0.501
/synapse/s1	0.02004	0.02004
/synapse/s2	0.02004	0.02004
/synapse/s3	0.02004	0.02004
/synapse/s4	0.02004	0.02004
/synapse/start	1	
/synapse/done	1	
/synapse/m11	0.00501	0.00501
/synapse/m12	0.00501	0.00501
/synapse/m13	0.00501	0.00501
/synapse/m14	0.00501	0.00501
/synapse/m21	0.00501	0.00501
/synapse/m22	0.00501	0.00501
/synapse/m23	0.00501	0.00501
/synapse/m24	0.00501	0.00501
/synapse/m31	0.00501	0.00501
/synapse/m32	0.00501	0.00501
/synapse/m33	0.00501	0.00501
/synapse/m34	0.00501	0.00501
/synapse/m41	0.00501	0.00501
/synapse/m42	0.00501	0.00501
/synapse/m43	0.00501	0.00501
/synapse/m44	0.00501	0.00501
/synapse/m	true	

Fig-6.3: Simulation results for synapse

6.4 NEURON

6.4.1 INPUTS AND OUTPUTS FOR THE ENTITY NEURON - N2

INPUTS

s1, s2, s3, s4 are the sum of weighted inputs,

start is the input signal introduced to take care of timing constraints during portmapping.

OUTPUTS

o1, o2, o3, o4 are the outputs based on sigmoid function,

d1, d2, d3, d4 are the derivatives of the sigmoid function

done is the output signal introduced to take care of timing constraints during portmapping.

6.4.2 RESULTS FOR THE ENTITY NEURON - N2:

Signal	Value	Waveform
/neuron/s1	0.02004	0.02004
/neuron/s2	0.02004	0.02004
/neuron/s3	0.02004	0.02004
/neuron/s4	0.02004	0.02004
/neuron/o1	0.50501	0.50501
/neuron/o2	0.50501	0.50501
/neuron/o3	0.50501	0.50501
/neuron/o4	0.50501	0.50501
/neuron/d1	0.249975	0.249975
/neuron/d2	0.249975	0.249975
/neuron/d3	0.249975	0.249975
/neuron/d4	0.249975	0.249975
/neuron/start	1	
/neuron/done	1	

Fig-6.4: Simulation results for neuron

6.5 ERROR GENERATOR AT OUTPUT

6.5.1 INPUTS AND OUTPUTS FOR THE ENTITY ERROR GENERATOR AT THE OUTPUT – E1

INPUTS

o1, o2, o3, o4 are the outputs of the neuron based on sigmoid function,
d1, d2, d3, d4 are the derivatives of the sigmoid function,
t1, t2, t3, t4 are the targets for which the neural network has to be trained,
start is the input signal introduced to take care of timing constraints during portmapping.

OUTPUTS

delta1, delta2, delta3, delta4 are the calculated errors (delta functions) obtained by comparing the output of the second neuron with the target value to be achieved,
done is the output signal introduced to take care of timing constraints during port mapping

6.5.2 RESULTS FOR THE ENTITY ERROR GENERATOR AT THE OUTPUT – E1

Signal	Value
/error_genr_op/o1	0.50501
/error_genr_op/o2	0.50501
/error_genr_op/o3	0.50501
/error_genr_op/o4	0.50501
/error_genr_op/d1	0.249975
/error_genr_op/d2	0.249975
/error_genr_op/d3	0.249975
/error_genr_op/d4	0.249975
/error_genr_op/t1	0.101
/error_genr_op/t2	0.101
/error_genr_op/t3	0.101
/error_genr_op/t4	0.101
/error_genr_op/delta1	-0.100992
/error_genr_op/delta2	-0.100992
/error_genr_op/delta3	-0.100992
/error_genr_op/delta4	-0.100992
/error_genr_op/start	1
/error_genr_op/done	1

Fig-6.5: Simulation results for error generator at the output

6.6 WEIGHT UPDATE

6.6.1 INPUTS AND OUTPUTS FOR THE ENTITY WEIGHT_UPDATE - W1

INPUTS

delta1, delta2, delta3, delta4 are the calculated errors (delta functions) obtained by comparing the output of the second neuron with the target value to be achieved,
o1, o2, o3, o4 are the outputs of the neuron based on sigmoid function,
tw11, tw12, tw13, tw14, tw21, tw22, tw23, tw24, tw31, tw32, tw33, tw34, tw41, tw42, tw43, tw44 are the input weights,
start is the input signal introduced to take care of timing constraints during portmapping.

OUTPUTS

w11, w12, w13, w14, w21, w22, w23, w24, w31, w32, w33, w34, w41, w42, w43, w44 are the output weights,
done is the output signal introduced to take care of timing constraints during portmapping.

6.6.2 RESULTS FOR THE ENTITY WEIGHT_UPDATE – W1:

Signal	Value	Waveform
/weight_update/delta1	-0.100992	-0.100992
/weight_update/delta2	-0.100992	-0.100992
/weight_update/delta3	-0.100992	-0.100992
/weight_update/delta4	-0.100992	-0.100992
/weight_update/o1	0.50501	0.50501
/weight_update/o2	0.50501	0.50501
/weight_update/o3	0.50501	0.50501
/weight_update/o4	0.50501	0.50501
/weight_update/w11	0.061002	0.061002
/weight_update/w12	0.061002	0.061002
/weight_update/w13	0.061002	0.061002
/weight_update/w14	0.061002	0.061002
/weight_update/w21	0.061002	0.061002
/weight_update/w22	0.061002	0.061002
/weight_update/w23	0.061002	0.061002
/weight_update/w24	0.061002	0.061002
/weight_update/w31	0.061002	0.061002
/weight_update/w32	0.061002	0.061002
/weight_update/w33	0.061002	0.061002
/weight_update/w34	0.061002	0.061002
/weight_update/w41	0.061002	0.061002
/weight_update/w42	0.061002	0.061002
/weight_update/w43	0.061002	0.061002
/weight_update/w44	0.061002	0.061002
/weight_update/tw11	0.01	0.01
/weight_update/tw12	0.01	0.01
/weight_update/tw13	0.01	0.01

Fig-6.6: Simulation results for weight update unit

6.7 WEIGHT TRANSFER

6.7.1 INPUTS AND OUTPUTS FOR THE ENTITY WU1

INPUTS

tw11, tw12, tw13, tw14, tw21, tw22, tw23, tw24, tw31, tw32, tw33, tw34, tw41, tw42, tw43, tw44 are the updated input weights,

start is the input signal introduced to take care of timing constraints during portmapping.

OUTPUTS

w11, w12, w13, w14, w21, w22, w23, w24, w31, w32, w33, w34, w41, w42, w43, w44 are the output weights to which updated weight values have been transfered,

done is the output signal introduced to take care of timing constraints during portmapping.

6.7.2 RESULTS FOR THE ENTITY WU1

Signal	Value	Waveform
/wu/w11	0.061002	0.061002
/wu/w12	0.061002	0.061002
/wu/w13	0.061002	0.061002
/wu/w14	0.061002	0.061002
/wu/w21	0.061002	0.061002
/wu/w22	0.061002	0.061002
/wu/w23	0.061002	0.061002
/wu/w24	0.061002	0.061002
/wu/w31	0.061002	0.061002
/wu/w32	0.061002	0.061002
/wu/w33	0.061002	0.061002
/wu/w34	0.061002	0.061002
/wu/w41	0.061002	0.061002
/wu/w42	0.061002	0.061002
/wu/w43	0.061002	0.061002
/wu/w44	0.061002	0.061002
/wu/tw11	0.061002	0.061002
/wu/tw12	0.061002	0.061002
/wu/tw13	0.061002	0.061002
/wu/tw14	0.061002	0.061002
/wu/tw21	0.061002	0.061002
/wu/tw22	0.061002	0.061002
/wu/tw23	0.061002	0.061002
/wu/tw24	0.061002	0.061002
/wu/tw31	0.061002	0.061002
/wu/tw32	0.061002	0.061002
/wu/tw33	0.061002	0.061002
/wu/tw34	0.061002	0.061002
/wu/tw41	0.061002	0.061002
/wu/tw42	0.061002	0.061002
/wu/tw43	0.061002	0.061002
/wu/tw44	0.061002	0.061002
/wu/start	1	
/wu/done	1	

Fig-6.7: Simulation results for weight transfer unit

6.8 ERROR GENERATOR AT INPUT

6.8.1 INPUTS AND OUTPUTS FOR THE ENTITY ERROR GENERATOR AT THE INPUT – E2

INPUTS

d1, d2, d3, d4 are the derivatives of the sigmoid function from the first neuron,
w11, w12, w13, w14, w21, w22, w23, w24, w31, w32, w33, w34, w41, w42, w43, w44 are the weights for first neuron which will be updated,
di1, di2, di3, di4 are the errors calculated (delta functions) from error generator at the output,
start, temp are the input signals introduced to take care of timing constraints during portmapping.

OUTPUTS

do1, do2, do3, do4 are the delta functions calculated by the error generator at the input,
e1, e2, e3, e4 are the errors calculated by the error generator at the input,
done is the output signal introduced to take care of timing constraints during portmapping.

6.8.2 RESULTS FOR THE ENTITY ERROR GENERATOR AT THE INPUT – E2

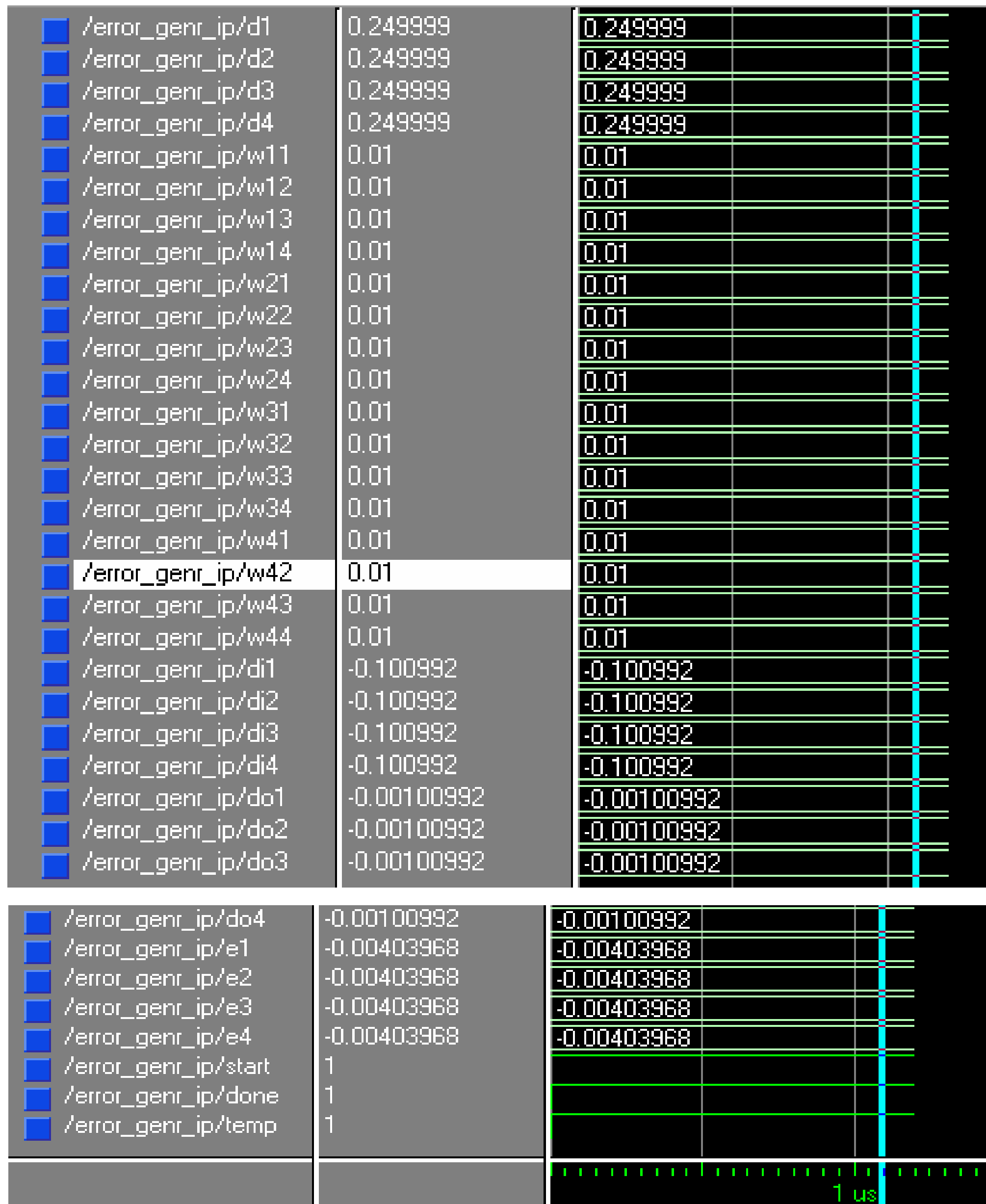


Fig-6.8: Simulation results for error generator at the input

6.9 WEIGHT UPDATE

6.9.1 INPUTS AND OUTPUTS FOR THE ENTITY WEIGHT_UPDATE – W2

INPUTS

delta1, delta2, delta3, delta4 are the calculated errors (delta functions) obtained by comparing the output of the second neuron with the target value to be achieved,
o1, o2, o3, o4 are the outputs of the neuron based on sigmoid function,
*tw11, tw1*2, tw13, tw14, tw21, tw22, tw23, tw24, tw31, tw32, tw33, tw34, tw41, tw42, tw43, tw44 are the input weights,
start is the input signal introduced to take care of timing constraints during port mapping.

OUTPUTS

w11, w12, w13, w14, w21, w22, w23, w24, w31, w32, w33, w34, w41, w42, w43, w44 are the output weights,
done is the output signal introduced to take care of timing constraints during port mapping.

6.9.2 RESULTS FOR THE ENTITY WEIGHT_UPDATE – W2

Signal	Value	Waveform
/weight_update/delta1	-0.00100992	-0.00100992
/weight_update/delta2	-0.00100992	-0.00100992
/weight_update/delta3	-0.00100992	-0.00100992
/weight_update/delta4	-0.00100992	-0.00100992
/weight_update/o1	0.501	0.501
/weight_update/o2	0.501	0.501
/weight_update/o3	0.501	0.501
/weight_update/o4	0.501	0.501
/weight_update/w11	0.010506	0.010506
/weight_update/w12	0.010506	0.010506
/weight_update/w13	0.010506	0.010506
/weight_update/w14	0.010506	0.010506
/weight_update/w21	0.010506	0.010506
/weight_update/w22	0.010506	0.010506
/weight_update/w23	0.010506	0.010506
/weight_update/w24	0.010506	0.010506
/weight_update/w31	0.010506	0.010506
/weight_update/w32	0.010506	0.010506
/weight_update/w33	0.010506	0.010506
/weight_update/w34	0.010506	0.010506
/weight_update/w41	0.010506	0.010506
/weight_update/w42	0.010506	0.010506
/weight_update/w43	0.010506	0.010506
/weight_update/w44	0.010506	0.010506
/weight_update/tw11	0.01	0.01
/weight_update/tw12	0.01	0.01
/weight_update/tw13	0.01	0.01
/weight_update/tw14	0.01	0.01
/weight_update/tw21	0.01	0.01
/weight_update/tw22	0.01	0.01
/weight_update/tw23	0.01	0.01
/weight_update/tw24	0.01	0.01
/weight_update/tw31	0.01	0.01
/weight_update/tw32	0.01	0.01
/weight_update/tw33	0.01	0.01
/weight_update/tw34	0.01	0.01
/weight_update/tw41	0.01	0.01
/weight_update/tw42	0.01	0.01
/weight_update/tw43	0.01	0.01
/weight_update/tw44	0.01	0.01
/weight_update/start	1	
/weight_update/done	1	

Fig-6.9: Simulation results for weight update unit

6.10 WEIGHT TRANSFER

6.10.1 INPUTS AND OUTPUTS FOR THE ENTITY WU2

INPUTS

tw11, tw12, tw13, tw14, tw21, tw22, tw23, tw24, tw31, tw32, tw33, tw34, tw41, tw42, tw43, tw44 are the updated input weights,

start is the input signal introduced to take care of timing constraints during portmapping.

OUTPUTS

w11, w12, w13, w14, w21, w22, w23, w24, w31, w32, w33, w34, w41, w42, w43, w44 are the output weights to which updated weight values have been transferred,

done is the output signal introduced to take care of timing constraints during port mapping.

6.10.2 RESULTS FOR THE ENTITY WU2

Signal	Value	Waveform
/wu/w11	0.010506	0.010506
/wu/w12	0.010506	0.010506
/wu/w13	0.010506	0.010506
/wu/w14	0.010506	0.010506
/wu/w21	0.010506	0.010506
/wu/w22	0.010506	0.010506
/wu/w23	0.010506	0.010506
/wu/w24	0.010506	0.010506
/wu/w31	0.010506	0.010506
/wu/w32	0.010506	0.010506
/wu/w33	0.010506	0.010506
/wu/w34	0.010506	0.010506
/wu/w41	0.010506	0.010506
/wu/w42	0.010506	0.010506
/wu/w43	0.010506	0.010506
/wu/w44	0.010506	0.010506
/wu/tw11	0.010506	0.010506
/wu/tw12	0.010506	0.010506
/wu/tw13	0.010506	0.010506
/wu/tw14	0.010506	0.010506
/wu/tw21	0.010506	0.010506
/wu/tw22	0.010506	0.010506
/wu/tw23	0.010506	0.010506
/wu/tw24	0.010506	0.010506
/wu/tw31	0.010506	0.010506
/wu/tw32	0.010506	0.010506
/wu/tw33	0.010506	0.010506
/wu/tw34	0.010506	0.010506
/wu/tw41	0.010506	0.010506
/wu/tw42	0.010506	0.010506
/wu/tw43	0.010506	0.010506
/wu/tw44	0.010506	0.010506
/wu/start	1	
/wu/done	1	

Fig-6.10: Simulation results for weight transfer unit

6.11.1 INPUTS AND OUTPUTS FOR THE FINAL ENTITY

INPUTS

i1, i2, i3, i4 are the input signals,
t1, t2, t3, t4 are the target signals,
start is the input signal introduced to take care of timing constraints during portmapping,
iw11, iw12, iw13, iw14, iw21, iw22, iw23, iw24, iw31, iw32, iw33, iw34, iw41, iw42, iw43, iw44 are the weights for the input layer,
tw11, tw12, tw13, tw14, tw21, tw22, tw23, tw24, tw31, tw32, tw33, tw34, tw41, tw42, tw43, tw44 are the weights for the output layer.

OUTPUTS

done1, done2, done3, done4, done5,done6, done7, done8, done9, done10, done11, done12, done13, done14, done15, done16, done17, done18, done19, done20, done21, done22, done23, done24, done25, done26, done27, done28, done29, done30, done31, done32, done33, done34, done35, done36, done37, done38, done39, done40 are the output signals introduced to take care of timing constraints during port mapping,
si1, si2, si3, si4 are the output signals from the first synapse,
oi1, oi2, oi3, oi4 are the output signals from the first neuron,
di1, di2, di3, di4 are the derivatives of the output from neuron,
sj1, sj2, sj3, sj4 are the output signals from the first synapse,
oj1, oj2, oj3, oj4 are the output signals from the first neuron,
dj1, dj2, dj3, dj4 are the derivatives of the output from neuron,
deltaj1, deltaj2, deltaj3, deltaj4 are the calculated errors (delta functions) obtained by comparing the output of the second neuron with the target value to be achieved,

deltai1, deltai2, deltai3, deltai4 are the delta functions calculated by the error generator at the input,

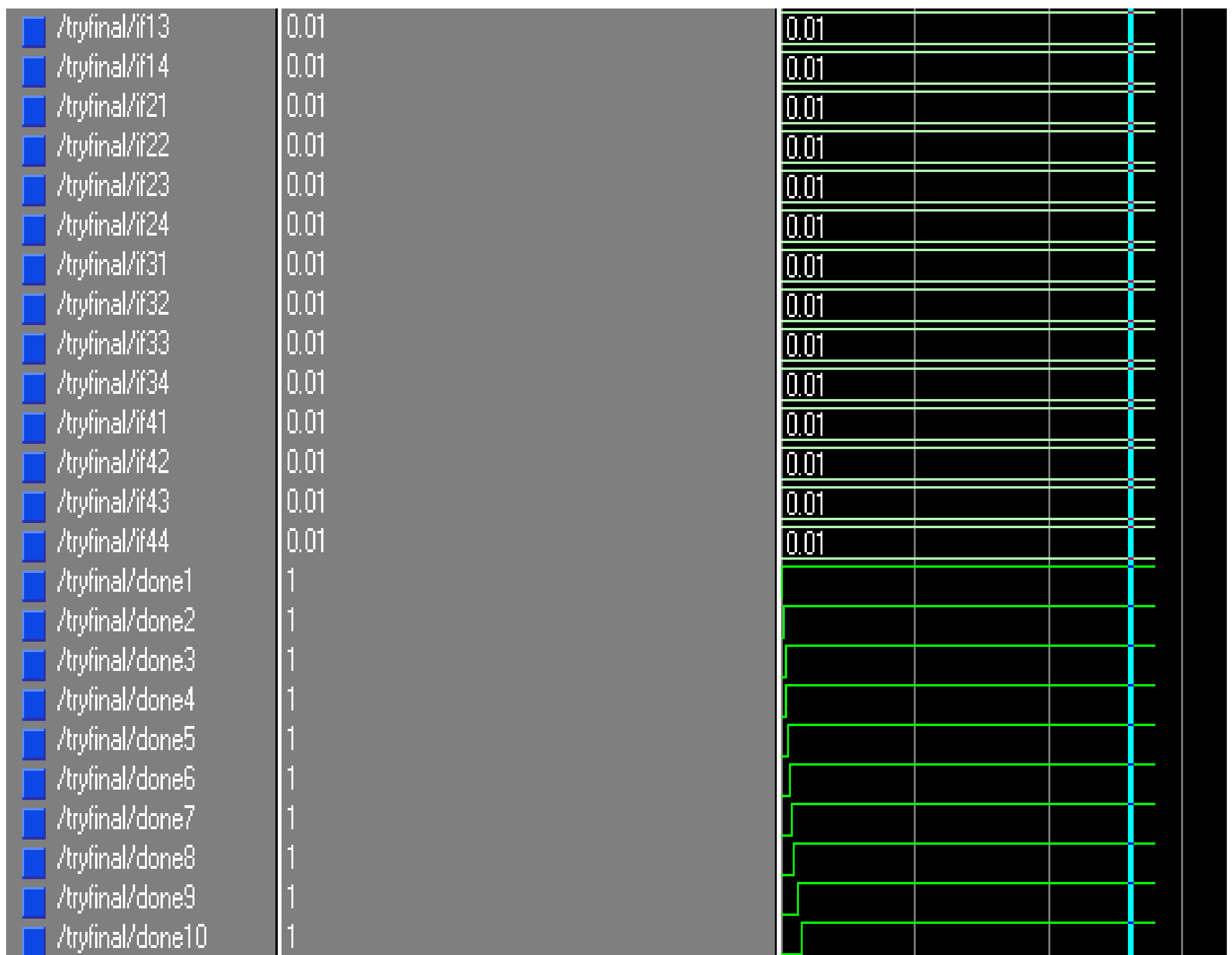
*errori1, errori2, errori3, er*rori4 are the error functions calculated by the error generator at the input,

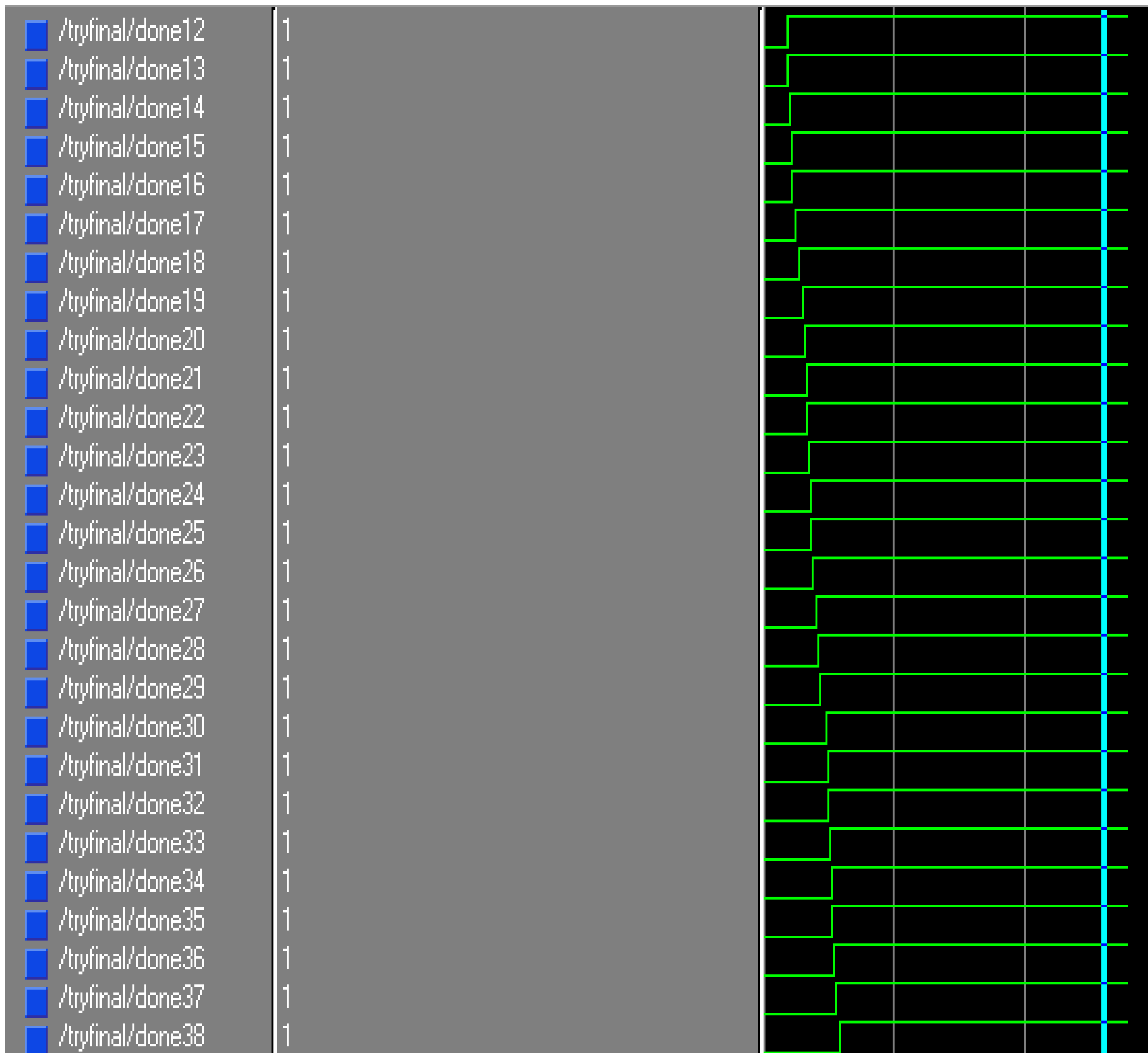
w11, w12, w13, w14, w21, w22, w23, w24, w31, w32, w33, w34, w41, w42, w43, w44 are the updated weights for the input layer,

w11, w12, w13, w14, w21, w22, w23, w24, w31, w32, w33, w34, w41, w42, w43, w44 are the updated weights for the output layer.

6.11.2 RESULTS FOR THE FINAL ENTITY

Signal	Value	Waveform
/tryfinal/i1	0.1	0.1
/tryfinal/i2	0.1	0.1
/tryfinal/i3	0.1	0.1
/tryfinal/i4	0.1	0.1
/tryfinal/t1	0.101	0.101
/tryfinal/t2	0.101	0.101
/tryfinal/t3	0.101	0.101
/tryfinal/t4	0.101	0.101
/tryfinal/start	1	
/tryfinal/iw11	0.01	0.01
/tryfinal/iw12	0.01	0.01
/tryfinal/iw13	0.01	0.01
/tryfinal/iw14	0.01	0.01
/tryfinal/iw21	0.01	0.01
/tryfinal/iw22	0.01	0.01
/tryfinal/iw23	0.01	0.01
/tryfinal/iw24	0.01	0.01
/tryfinal/iw31	0.01	0.01
/tryfinal/iw32	0.01	0.01
/tryfinal/iw33	0.01	0.01
/tryfinal/iw34	0.01	0.01
/tryfinal/iw41	0.01	0.01
/tryfinal/iw42	0.01	0.01
/tryfinal/iw43	0.01	0.01
/tryfinal/iw44	0.01	0.01
/tryfinal/if11	0.01	0.01
/tryfinal/if12	0.01	0.01


/tryfinal/if13
0.01
0.01
/tryfinal/if14
0.01
0.01
/tryfinal/if21
0.01
0.01
/tryfinal/if22
0.01
0.01
/tryfinal/if23
0.01
0.01
/tryfinal/if24
0.01
0.01
/tryfinal/if31
0.01
0.01
/tryfinal/if32
0.01
0.01
/tryfinal/if33
0.01
0.01
/tryfinal/if34
0.01
0.01
/tryfinal/if41
0.01
0.01
/tryfinal/if42
0.01
0.01
/tryfinal/if43
0.01
0.01
/tryfinal/if44
0.01
0.01
/tryfinal/done1
1
/tryfinal/done2
1
/tryfinal/done3
1
/tryfinal/done4
1
/tryfinal/done5
1
/tryfinal/done6
1
/tryfinal/done7
1
/tryfinal/done8
1
/tryfinal/done9
1
/tryfinal/done10
1


/tryfinal/done12 1
/tryfinal/done13 1
/tryfinal/done14 1
/tryfinal/done15 1
/tryfinal/done16 1
/tryfinal/done17 1
/tryfinal/done18 1
/tryfinal/done19 1
/tryfinal/done20 1
/tryfinal/done21 1
/tryfinal/done22 1
/tryfinal/done23 1
/tryfinal/done24 1
/tryfinal/done25 1
/tryfinal/done26 1
/tryfinal/done27 1
/tryfinal/done28 1
/tryfinal/done29 1
/tryfinal/done30 1
/tryfinal/done31 1
/tryfinal/done32 1
/tryfinal/done33 1
/tryfinal/done34 1
/tryfinal/done35 1
/tryfinal/done36 1
/tryfinal/done37 1
/tryfinal/done38 1

Signal	Value
/tryfinal/done40	1
/tryfinal/si1	{0.004 0.00384923 0.00425489 0.00513215}
/tryfinal/si2	{0.004 0.00384923 0.00425489 0.00513215}
/tryfinal/si3	{0.004 0.00384923 0.00425489 0.00513215}
/tryfinal/si4	{0.004 0.00384923 0.00425489 0.00513215}
/tryfinal/oi1	{0.501 0.500962 0.501064 0.501283}
/tryfinal/oi2	{0.501 0.500962 0.501064 0.501283}
/tryfinal/oi3	{0.501 0.500962 0.501064 0.501283}
/tryfinal/oi4	{0.501 0.500962 0.501064 0.501283}
/tryfinal/di1	{0.249999 0.249999 0.249999 0.249998}
/tryfinal/di2	{0.249999 0.249999 0.249999 0.249998}
/tryfinal/di3	{0.249999 0.249999 0.249999 0.249998}
/tryfinal/di4	{0.249999 0.249999 0.249999 0.249998}
/tryfinal/sj1	{0.02004 -0.0560954 -0.126544 -0.19171}
/tryfinal/sj2	{0.02004 -0.0560954 -0.126544 -0.19171}
/tryfinal/sj3	{0.02004 -0.0560954 -0.126544 -0.19171}
/tryfinal/sj4	{0.02004 -0.0560954 -0.126544 -0.19171}
/tryfinal/oj1	{0.50501 0.48598 0.468406 0.452219}
/tryfinal/oj2	{0.50501 0.48598 0.468406 0.452219}
/tryfinal/oj3	{0.50501 0.48598 0.468406 0.452219}
/tryfinal/oj4	{0.50501 0.48598 0.468406 0.452219}
/tryfinal/dj1	{0.249975 0.249803 0.249002 0.247717}
/tryfinal/dj2	{0.249975 0.249803 0.249002 0.247717}
/tryfinal/dj3	{0.249975 0.249803 0.249002 0.247717}
/tryfinal/dj4	{0.249975 0.249803 0.249002 0.247717}
/tryfinal/deltaj1	{0.0752338 0.0723155 0.069325 0.0663481}
/tryfinal/deltaj2	{0.0752338 0.0723155 0.069325 0.0663481}

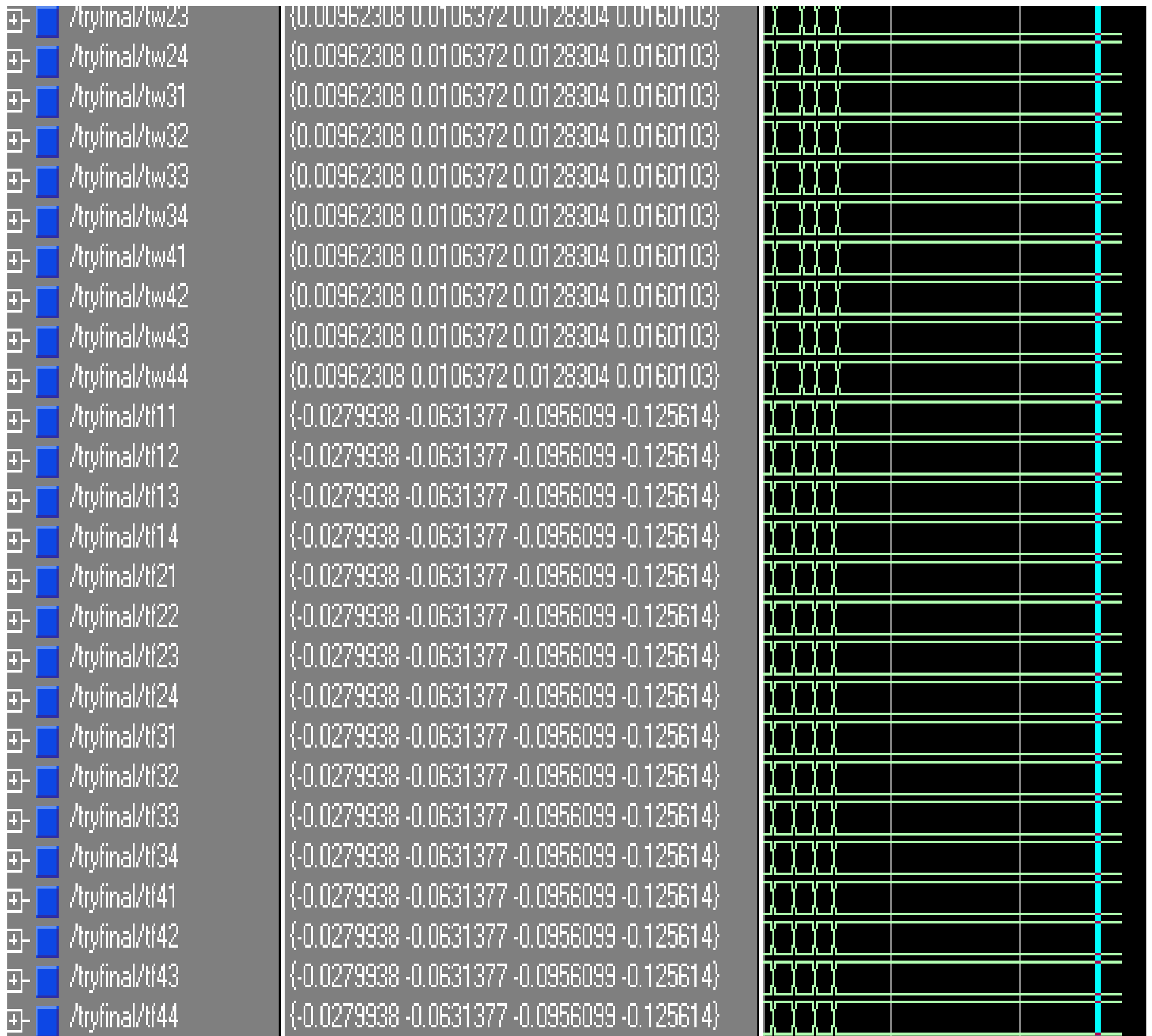


Fig-6.11: Simulation results for the final entity

As can be seen above from the outputs of the second neuron (*oj1, oj2, oj3, oj4*) that the value is slowly approaching the target *(t1, t2, t3, t4*) concluded that the neural network is getting trained.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

This thesis describes the VHDL implementation of a supervised learning algorithm for artificial neural networks. The algorithm is the Error Back propagation learning algorithm for a layered feed forward network and this algorithm has many successful applications for training multilayer neural networks. VHDL implementation creates a flexible, fast method and high degree of parallelism for implementing the algorithm.

The proposed neural network has two layers with four neurons each and sixteen synapses per layer. Different situations may need neural networks of different scales. Such a situation can be overcome by combining a certain number of such unit neural networks. Back propagation algorithm logic has been divided into individual modules and these modules have been implemented in VHDL using behavioral modeling.

The different modules are: synapse, neuron, error generator at the input, error generator at the output, weight update unit and a weight transfer unit. At the synapse inputs are multiplied by the corresponding weights and the weighted inputs are summed together to get four outputs. After the synapse is the neuron that calculates the output in accordance with the transfer function sigmoid and its derivative has also been calculated. Further the neuron is followed by the synapse and the neuron because in this paper two layer network has been considered. Then the network is followed by the error generator at the output, which compares the output of the neuron with the target signal for which the network has to be trained. Similarly, there is error generator at the input, which updates the weights of the first layer taking into account the error propagated back from the output layer.

Finally, a weight transfer unit is present just to pass on the values of the updated weights to the actual weights.

Then a final entity having structural modeling has been formulated in which all the entities are port mapped. The results constitute simulation of VHDL codes of different modules in MODELSIM SE 5.5c. The simulation of the structural model shows that the neural network is learning and the output of the second layer is approaching the target.

7.2 FUTURE SCOPE

An algorithm similar to back propagation algorithm, which may be used to train networks whose neurons may have discontinuous or non-differentiable activation functions. These new algorithms should also have the capability to speed up the convergence of back propagation algorithm. Modified forms of back propagation algorithm such as Quick BP, RPROP, SAR-PROP, and MGF-PROP can provide a great help.

- Taking into consideration massively expensive parallel architecture required for hardware implementation of the algorithm, this problem can be overcomed by using analog computation elements such as multipliers and adders.
- Second solution to this problem is the use of fixed-point computation for limited precision architectures, where a look up table can be used for squishing the functions.

REFERENCES

[1] Simon Haykin, "Neural Networks", Second edition by, Prentice Hall of India, 2005.

[2] Christos Stergiou and Dimitrios Siganos, "Neural Networks", Computer Science Deptt. University of U.K., Journal, Vol. 4, 1996.

[3] Robert J Schalkoff, "Artificial Neural Networks", McGraw-Hill International Editions, 1997.

[4] Uthayakumar Gevaran, "Back Propagation", Brandeis University, Department of Computer Science.

[5] Jordan B.Pollack, "Connectionism: Past, Present and Future", Computer and Information Science Department, The Ohio State University, 1998.

[6] "Artificial Neural Network", Wikipedia Encyclopedia, Wikimedia Foundation, Inc., 2006.

[7] Pete Mc Collum, "An Introduction to Back Propagation Neural Networks", The Newsletter of the Seattle Robotics Society.

[8] Andrew Blais and David Mertz, "An Introduction to Neural Networks – Pattern Learning with Back Propagation Algorithm", Gnosis Software, Inc., July 2001.

[9] Harry B.Burke, "Evaluating Artificial Neural Networks for Medical Applications", New York Medical College, Deptt. Of Medicine, Valhalla, IEEE, 1997.

[10] Wan Hussain, Wan Ishak, "The Potential of Neural Networks in Medical Applications", Generation 5 Society, 2005.

[11] J. Bhasker, "A VHDL Primer", Third edition, Pearson Education, Inc., 2004.

[12] Peter J. Ashenden, "The Designer's Guide to VHDL", Second Edition, Morgan Kaufmann Publishers, 2004.

[13] "A Brief History of VHDL", Doulos Ltd., 2005

[14] www.peakfpga.com/vhdlref/refguide

[15] tams-www.nformatik.uni-hamburg.de/vhdl/vhdl.html, 2006.

[16] Chun Lu, Bingxue Shi and Lu Chen, "An expandable on-chip Back Propagation learning neural network chip", Int. J.Electronics, Vol 90, No. 5, 331-340, 2003.

[17] Tom Baker and Dan Hammerstrom, "Characterization of Artificial Neural Network Algorithms", Dept. of Computer Science and Engineering, Oregon Graduate Center, IEEE, 1989.

[18] Wen Jin-Wei, Zhao Jia-Li, Luo Si-Wei and Han Zhen, "The Improvement of Back Propagation Neural Network Learning Algorithm", Department of Computer Science Technology, Northern Jiaotong University Beijing, PR China, Proceedings of ICSP2000.

[19] Mohammed A. Otair, Jordan University of Science and Technology, Irbed, Jordan; Walid A. Salameh, Princess Summaya, University for Science and Technology, Amman, Jordan, " Speeding up Back Propagation Neural Networks", Proceedings of the 2005 Informing Science and IT Education Joint Conference, 2005.

[20] S. C. Ng, S. H. Leung and A. Luk, "The Generalized Back Propagation Algorithm with convergence analysis", Department of Computing and Mathematics, Hong Kong Technical College (Chai Wan), Department of Electronic Engineering, City University of Hong Kong, St. B&P Neural Investments Pty. Ltd., Australia, IEEE, 1999.

[21] Nazeih M.Botros and M.Abdul-Aziz, "Hardware Implementation of an Artificial Neural Network using Field Programmable Gate Arrays (FPGA's)", IEEE Transactions on Industrial Electronics, Vol 41, No. 6,Dec 1994.

[22] Rafael Gadea, Franciso Ballester, Antonio Mocholi, Joaquin Cerda, "Artificial Neural Network Implementation on a Single FPGA of a Pipelined On-Line Back Propagation", 13th International Symposium on System Synthesis, 2000.

[23] Cesare Alippi, Meyer E.Nigri, "Hardware Requirements for Digital VLSI Implementation of Neural Networks", Deptt. of Electronics, Italy, Deptt. Of Computer Science, U.K., IEEE, 2000.

[24] Christopher Cantrell, Dr. Larry Wurtz, "A parallel Bus Architecture for Ann's", University of Albana, Deptt. OF electrical Engg, Tuscaloosa, IEEE, 1993.

[25] Dimokritos A. Panagiotopoulos, Sanjeev K. Singh and Robert W. Newcomb, "VLSI Implementation of a Functional Neural Network", Technological Institute of Thessaloniki, Univ. of Maryland, USA, IEEE 1997.

[26] Annema, A.J., “Hardware realization of a neuron transfer function and its derivative”, Electronics Letters, 30, 576-577, 1994.

[27] Lu, C., and Shi, B.X., “Circuit design of on-chip Back Propagation learning neural network with programmable neuron characteristics”, Chinese Journal of Semiconductors, 21, 1164-1169, 2000.

[28] Lu, C., and Shi, B.X., “Circuit Design of an adjustable neuron activation function and its derivative”, Electronics Letters, 36, 553-555, 2000.

[29] D.A. Findlay, “Training Networks with Discontinuous Activation Functions”, Plessey Research and Technology, U.K.

[30] Dr. Hussein Chible, “Analog Circuit for Synapse Neural Networks VLSI implementation”, Lebanese Univ., IEEE 2000.

[31] Al-Ruwaihi, K.M., “Current-mode programmable synapse circuits for analogue ULSI neural networks”, Int.J.Electronics, 86,189-205, 1999.

LIST OF PUBLICATIONS

- Submitted a paper on “**BACK PROPAGATION ALGORITHM IMPLEMENTATION IN VHDL”** at CHITKARA INSTITUTE OF ENGG & TECH, CHANDIGARH in June 2006.

- Presented a paper on **“CHARACTER RECOGNITION USING NEURAL NETWORKS”** in conference on **ELECTRONIC CIRCUITS AND COMMUNICATION SYSTEMS** at THAPAR INSTITUTE OF ENGG. & TECH., PATIALA in February 2006.

- Presented a paper on **“FREE SPACE OPTICS”** in conference on **ELECTRONIC CIRCUITS AND COMMUNICATION SYSTEMS** at THAPAR INSTITUTE OF ENGG. & TECH., PATIALA in February 2006.