CHAPTER I

1.1. Introduction

Artificial Intelligence is an alternative solution for processing the data and information like human brain. From the start of20th century, the scientists have started the study of brain functions. 'Artificial Neural Networks' (ANN) represent an area of expert system. It is basically an attempt to simulate the processing of information like human brain. The study of neural networks is also called neuro engineering or neural computing. There is a variety of neural net architectures that have been developed over the past few decades. Beginning from the feed-forward networks and the multi-layer perceptrons (MLP) to the highly dynamic recurrent ANN. There are many different types of neural nets like feedback, feed forward etc. Also, various algorithms have been developed for training these networks to perform the necessary functions. In this thesis, a new approach has been applied to build neural nets using Lab VIEW, virtual instrumentation software developed by National instruments USA. It has been successfully used for data acquisition and control functions. However it has been used here for building neural nets, presently not the library function of Lab VIEW. Both supervised and the unsupervised neural nets have been successfully developed in this thesis using Lab VIEW and it has been shown that Lab VIEW is a very powerful software tool for building neural nets.

Neural nets are parallel processors. They have data flowing in parallel lines simultaneously. Lab VIEW has the unique ability to develop data flow diagrams that are highly parallel in structure. Lab VIEW seems to be a very effective approach for building neural nets.

1.2 Objective

The main objective of this thesis is to build neural net model using Lab VIEW. It is a graphical programming software, tool to develop the various applications. Basically, it is an Application Development Environment (ADE) for building user friendly applications. This thesis concentrates on a Lab VIEW approach to build various neural net structures.

1.3 Motivation

Lab view is a powerful graphical tool for measurement, instrumentation and control structured problems .However, it does not provide neuron models in its library. Neural network

is another power full soft computing tool which takes into accounts the uncertainties and vagueness in the input measurements. An attempt to develop the neural network model for the lab view library. This model can directly be used by the designers for the analysis and predict the performance of the processes.

1.4 Features of neurons

Neurons are the basic element for processing the biological information. It possesses the following features:

(i) Neurons are massively parallel distributed structure

(ii) They have ability to learn and generalize.

These two features of neurons enable them for information - processing capabilities and make it possible for neural networks to solve complex problems that are currently intractable.

In addition, the neurons also possess the following features:

(iii) Uniformity of analysis and design: Basically, neural networks enjoy universality as information processors i.e. the same notation is used in all domains involving the application of neural networks.

(iv) Collective Computation: The network performs routinely many operations in parallel and also a given task in a distributed manner.

(V) Ability to deal with a variety of data situations: The network can deal with information that is fuzzy, probabilistic, noisy and inconsistent.

(vi) Flexibility: The network automatically adjusts to a new environment without using any preprogrammed instructions.

1.5 Features of Lab view

(i) The main feature of Lab View its modularity. A large project can be broken down into functional solvable units to simplify the programming complexity.

(ii)The virtual instruments running on notebook automatically incorporate their portable nature to the engineers and scientists whose needs, applications and requirements change very quickly, need flexibility to create their own solutions

(iii)It adapts a virtual instrument to our particular needs without having to replace the entire device.

(iv) Flexibility: Modification and adaptation to a particular device is easy.

(v) Lower cost: By employing virtual instrumentation solutions, lower capital costs, system development costs, and system maintenance costs are reduced.

(vi) Plug in and networked hardware: There is a wide variety of available hardware that can either be plugged into the computer or accessed through a network.

1.6 Problem formation

(i) The mathematical model of bio neurons has been developed based on its functions and cell structure.

(ii)The Neuron model developed is to be transformed in the Lab VIEW environment.

(iii) Finally the neural network model is developed and transformed to the Lab VIEW environment for information processing on real-time basis.

1.7 Dissection of Thesis

The material of this dissertation has been organized in six chapters. The contents of the chapter are briefly outlined as indicated below.

Chapter-I discusses the objectives of the thesis, motivation and project formation. Chapter-II briefs the work carried out earlier present in the literature.

Chapter-III defines the neuron and its structure. Based on this structure and functional requirements, a mathematical model of neural net is developed.

Chapter-IV gives the Introduction of lab view, its features and its basic development procedure explained in detail.

Chapter-V discusses the fusion of neural net in Lab view.

Chapter -VI presents the conclusion and scope for future work

1.8 conclusions

This chapter presents the introduction of neural networks and objective of this thesis. Also this chapter explains features of Lab view and problem formation of the thesis. This chapter also highlights the need for selection of this topic of research.

CHAPTER II

This chapter contains a brief literature review on the area of

(i) Neural networks

(ii) Lab VIEW and

(iii) Important features of Lab VIEW and neural nets.

2.1. Introduction

Modeling is a transformation of physical system into a mathematical representation by means of algebraic and differential equations such that the response of physical system and simulated response shall conform to each other within limit of tolerance. The solution of these equations depends on how accurately they represent the physical system and assumptions made.

Neuron is an information processing unit that is fundamental to the operation of a neural network. In the neural network the emulation of biological intelligence is achieved either by replacing the human or borrowing the idea how biological system represent and solve the problems. Neural network is a quantitative method for studying the system response in which the parameters values are uncertain. The uncertainties can be better accounted by the neural network.

Lab VIEW is a graphical programming language that uses icons instead of lines of text to create applications. In contrast to text-based programming Languages, where instructions determine program execution, Lab VIEW uses dataflow programming, where the flow of data determines execution. Neural nets are parallel processing program tool in which data flow from one layer to another layer simultaneously. Lab View has the unique ability to develop data flow diagrams that are highly parallel in structure. Thus Lab VIEW is a very effective approach for Building neural nets. However, present version of Lab View does not provide the facility to use neural networks for system study under uncertain defined model. To overcome this drawback, the authors have designed and developed neural net model under Lab View environment for the simulation study of real time process plant.

2.2. Neural Networks

2.2.1. Historical development of neural networks:

Artificial neural systems have been developed from 1943 onwards. In 1943, McCulloch and Pitts outlined the first formal model of an elementary computing neuron. The model included all the necessary elements required to perform logic operations and could thus function as an arithmetic-logic computing element. This neuron model laid the foundation for future developments. The main drawback of this model of computation is that the weights are fixed and hence the model could not learn from examples.

In 1949, Donald Hebb proposed learning scheme for updating neuron connections, which we now refer to as the Hebbian Learning rule. He stated that information can be stored in the form of connections, and postulated a learning technique that had profound impact on the future development in this field. During 1950s, the first neuro computers were built and tested. They adapted connections automatically.

In 1954 a learning machine was developed by Marvin minsky, in which the connection strengths could be adapted automatically [minsky, 1954]. But it was in 1958 that Rosenblatt proposed the perceptron model, which has weights adjustable by the perceptron learning law [Rosenblatt, 1958]. It was a trainable machine capable of learning to classify certain patterns by modifying the connections between the threshold elements.

In 1960s Widrow and his group proposed an Adeline model for a computing element and an LMS learning algorithm to adjust the weights of an Adeline model [wodrow and Hoff, 1960]. The convergence of the LMS algorithm was proved. The algorithm was successfully used for adaptive signal processing situations

Bernard in the same year, 1960 introduced, a device called ADALINE, and a new powerful learning called the Widrow-Hoff learning rule was developed by Bernard widrow and the rule minimized the summed squared error during training involving pattern classification.

One of the earliest trainable layered neural networks with multiple adaptive elements was the madaline (multiple-Adeline) structure proposed by Widrow and his students in the year 1962.[Wodrow, 1962]

In 1969 Minsky and papert to demonstrate that there are fundamental limits on single-layer perceptrons computing.

In 1969, an elegant paper on non holographic associative memory by Willshaw, Buneman, and Longuet- Higgins published. This paper presents two ingenious network models: A simple optical system realizing a correlation memory, and a closely related neural network suggested by the optical memory. Other significant contributions to the early development of associative memory include papers by Anderson (1972), Kohonen (1972), and nakano (1972), who independently and in the same year introduced the idea of a correlation matrix memory based on the outer product learning rule.

An important activity that did emerge in the 1970s was self- organizing maps using competitive learning. The computer simulation work done by Vonder Malsburg (1973) was perhaps the first to demonstrate self- organization.

In the 1980s major contributions to the theory and design of neural networks were made on several fronts, and with it there was a resurgence of interest in neural networks. Gross Berg (1980) established a new principle of self-organization known as adaptive resonance theory (ART). In 1982, Hopfield used the idea of an energy function to formulate a new way of understanding the computation performed by recurrent networks with symmetric synaptic connections. Moreover, he established the isomorphism between such a recurrent network and an Isling model used in statistical physics. This analogy paved the way for a deluge of physical theory to enter neural modeling, thereby, transforming the field of neural networks. Another important development in 1982 was the publication of Kohonen's paper on self- organizing maps (Kohonen, 1982) using a one or two dimensional lattice structure, which was different in some respects from the earlier work by Willshaw and Vonder Malsburg. In 1983, Krikpatrick, Gelatt, and Vecchi described a new procedure called simulated annealing, for solving combinatorial optimization. The idea of stimulate annealing was later used by Ackley, Hinton, and Sejnowski (1985) in the development of Boltzman machine, which was the first successful realization of a multilayer neural network.

In1986 the development of the back propagation algorithm was reported by Rumel Hart, Hinton, and Williams.

In 1988 Linsker described a new principle for self-organization in a perceptual network (Linsker, 1988a). The principle is designed to preserve maximum information about input activity patterns, subject to such constraints as synaptic connections and synapse dynamic range. Also in 1988, Broom, Head and Lowe described a procedure for the design of layered feed forward networks using radial basis functions, which provide an alternative to multilayer perceptrons.

In 1990, Poggio and Girosi (1990a) further enriched the theory of radial basis functions by applying by applying Tikhonov's regulation theory. In the early 1990s, Vapnik and co workers invented a computationally powerful class of supervised learning networks, called support vector machines, for solving pattern recognition, regression, and density estimation problems. (Boser, Guyon, and Vapnik, 1992; Cortes and Vapnik 1995; Vapnik 1995, 2008). This new method is based on results in the theory of learning with finite sample sizes.

In the year 1995, Freeman explains the concept of chaos which is key aspect of physical phenomena. According to freeman, patterns of neural activity are not imposed from outside the brain rather they are constructed from within.

2.3. Different types of neural net architectures:

Artificial neural networks can model the behavior of biological neural networks. The aim behind the development of artificial neural networks was to exploit parallel-processor computing in place traditional serial computation.

The term architecture refers to the connectivity of a network. Depending on the kind of interconnection of the neurons and the training algorithm to fit the weights, different neural network types can be defined. These can be divided into three major groups as described below.

1. Feed forward networks

There is no feedback within the network. The coupling takes place from one layer to the next. The information flows, in general, in the forward direction in the feed forward networks. Figure 2.1 shows the structure of the feed forward network.

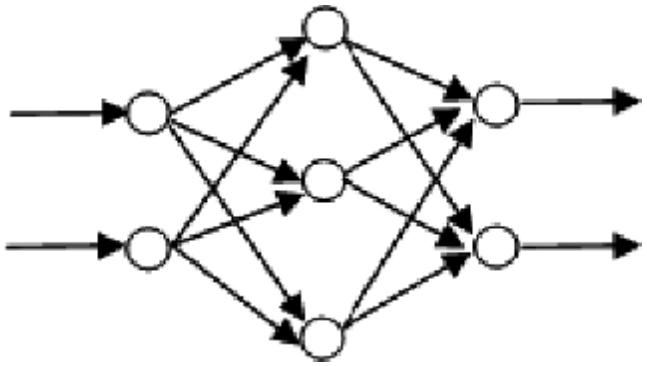

Fig 2.1 Structure of the feed forward network

2. Feedback networks

In this kind of network, the output of a neuron is either directly or indirectly fed back to its input via other linked neurons. This kind of network is frequently used in complex pattern recognition tasks, e.g., speech recognition etc. Figure 2.2 shows the structure of the feedback network.

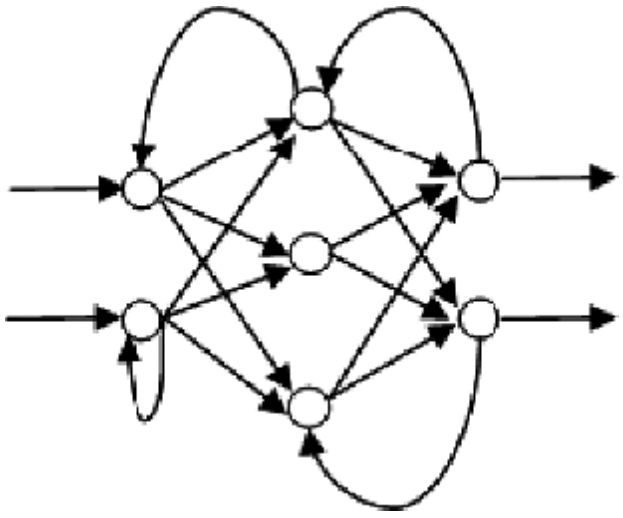

Fig 2.2 Structure of feedback network

3. Lateral networks

In this kind of network, there exist couplings of neurons within one layer. There is no essentially explicit feedback path amongst the different layers. This can be thought of as a compromise between the forward and feedback network. Figure 3.9 shows the structure of the lateral network.

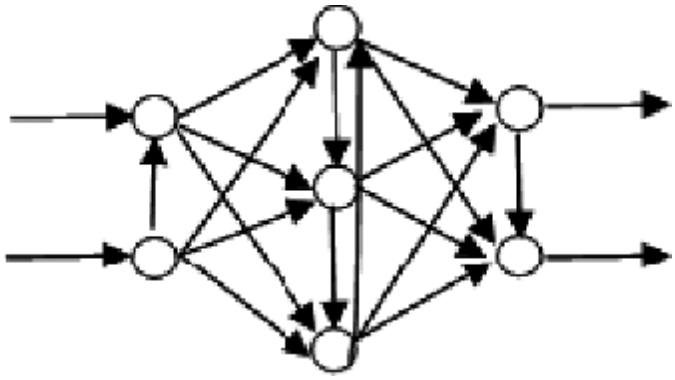

Fig 2.3 Structure of Lateral network

2.4. Lab VIEW

Development history:

Nearly all of the early instrument control programs were written in BASIC, because it had been the dominant language used with dedicated instrument controllers. It required engineers and other users to become programmers before becoming instrument users, so it was hard for them to exploit potential that computerized instrumentation could bring. Therefore, an important milestone in the history of virtual instrumentation happened in 1986, when National Instruments introduced Lab VIEW 1.0 on a PC platform. Lab VIEW introduced graphical user interfaces and visual programming into computerized instrumentation, joining simplicity of a user interface operation with increased capabilities of computers. Today, the PC is the platform on which most measurements are made, and the graphical user interface has made measurements user-friendlier. As a result, virtual instrumentation made possible decrease in price of an instrument. As the virtual instrument depends very little on dedicated hardware, a customer could now use his own

computer, while an instrument manufactures could supply only what the user could not get in the general market.

2.5. Lab VIEW and its Importance:

National Instruments Lab VIEW is a highly productive graphical programming environment that combines easy to use graphical developments with the flexibility of a powerful programming language. It is a general purpose programming language used for developing projects graphically. It is a revolutionary programming language that depicts program code graphically rather than textually. One major benefit of using graphical programming rather than text-based languages is that one writes program codes simply by connecting icons. In addition, graphical programming solutions offer the performance and flexibility of text-based programming environments, but conceal many programming intricacies such as memory allocation and syntax.

Lab VIEW involves structured dataflow diagramming. It is, in fact, a much richer computational model than the control flow of popular text-based languages because it is inherently parallel, while C/C++ is not. Data flow diagrams in general are so useful for building general purpose applications, that, instead of working with text, one can directly modify and edit dataflow diagrams. This is briefly being done at same level in Lab VIEW. Because it is the flow of data between objects on a block diagram, and not sequential lines of text, that determines execution order in Lab VIEW, one can create diagrams that simultaneously execute multiple operations. Consequently, Lab VIEW is a multitasking system capable of concurrently running multiple execution threads and multiple VI's (Virtual Instruments). Basically, in any programming language there are editors and compilers. In text based programming languages, the editor edits the program in its own style and finally produces ASCII characters which are passed onto the compiler. But in graphical and pictorial programming, an image is produced by the editor, which is parsed by the compiler. But Lab VIEW has a better approach than the above mentioned two methods when it comes to editing and compiling. The editor parses the image as it is being constructed so it is highly interactive. As a result, a higher and richer version of graphics is visible on screen. Also, another significant advantage of Lab VIEW is the relatively high speed of the compiler. The editor has a rich set of operations to quickly create elaborate user interfaces by direct manipulation. The fact that every module, or VI, has a user interface means

that interactive testing is simple to do at each step, without writing any extra code. The fraction of an application that has to be completed before meaningful testing can take place is much smaller in Lab VIEW than in traditional programming tools, making design much faster. Even the data types on the diagram are easy to use. Strings and arrays can be routed and operated on without worrying about the details of memory allocation, which means that errors, such as losing or overwriting memory, just do not exist. The net result of all of these capabilities in Lab VIEW greatly increases its potential. Lab VIEW accelerates development over traditional programming by a significant factor. With the modularity and hierarchical structure of Lab VIEW, one can prototype, design, and modify systems in a relatively short amount of time.

Lab VIEW has many interesting features that make it a very useful tool for building neural nets. Lab VIEW programs are called Virtual Instruments (VI), because they appear very similar to actual laboratory instruments. There are two parts to a VI – the 'Front Panel' and the 'Block Diagram'. The front panel constitutes one part of the program in which the GUI is developed. Controls, constants and indicators form an integral part of the front panel. Examples for controls and indicators include knobs, horizontal and vertical sliders, buttons, input devices, graphs and charts. Lab VIEW utilizes a powerful Graphical User Interface (GUI). The Front panel acts as the user interface. The front panel, as the name suggests acts as the front-end of the virtual instrument, while the block diagram is the background code. Thus, the block diagram constitutes the actual program part of Lab VIEW. It is highly graphical, and no text codes are involved. The block diagram follows a similar idea as the 'data flow diagram' concept, in which the logic is presented as a diagram rather than as text code.

After building the front panel, codes could be added by using graphical representations of functions to control the front panel objects. The block diagram contains this graphical source code. Front panel objects appear as terminals on the block diagram.

The terminals represent the data type of the control (input) or indicator (output). The front panel controls or indicators can be configured to appear as icons or data type terminals on the block diagram. By default, front panel objects appear as icon terminals. The terminals in the block diagram represent the data type as double precision, floating point or any other format of that type. Terminals are entry and exit ports that exchange information between the front panel and the block diagram. The data that a user enters into the front panel controls' enter the block diagram through the control terminals. After the functions complete their calculations in the

block diagram, the data flow to the indicator terminals, where they exit the block diagram, reenter the front panel, and appear as front panel indicators. Nodes are objects on the block diagram that have inputs and/or outputs and perform operations when a VI runs. They are analogous to statements, operators, functions, and subroutines in text-based programming languages.

2.6. Mat Lab vs. lab VIEW

One benefit of Lab VIEW over other development environments is the extensive support for accessing instrumentation hardware. Drivers and abstraction layers for many different types of instruments and buses are included or are available for inclusion. These present themselves as graphical nodes. The abstraction layers offer standard software interfaces to communicate with hardware devices. The provided driver interfaces save program development time. The sales pitch of National Instruments is, therefore, that even people with limited coding experience can write programs and deploy test solutions in a reduced time frame when compared to more conventional or competing systems.

Most of the concepts discussed for Lab VIEW are valid for MATLAB. The main differences between MATLAB and Lab VIEW are:

1) MATLAB has its own language and commands

2) Unlike Lab VIEW, mainly commands and scripts are needed to run the code

3) To interface the DAQ and other instruments with MATLAB need MATLAB drivers

2.6 Conclusion

This chapter presents the introductions of neural networks development history and Lab VIEW development history. Also this chapter explains the fundamentals of neural networks and Lab VIEW.

CHAPTER III

3.1 Introduction

Artificial neural networks (ANN) are systems that are designed to make use of some organizational principles assembling those of human brain. They have a large number of highly interconnected processing elements that usually operate in parallel and are configured in regular architecture. The collective behavior of an ANN demonstrates the ability to learn, recall and generalize from training pattern of data, like human brain. It is inspired by modeling, networks of real neurons in the brain. Hence the processing elements in ANN are called "Artificial neurons", or simply neurons. In this chapter an attempt has been made for developing mathematical model of the neural network which shall be used for Lab VIEW programming in the subsequent chapters.[1-3]

3.2 What is neural network?

An Artificial neural network (ANN) is an information-processing paradigm that is inspired by the way biological neurons process information.Thus,neural is composed of a large number of highly inter connected processing elements (neurons) working in unison to solve specific problems.ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological system involves adjustments to the synaptic connections that exist between the neurons.[1]

3.3 Neural network construction and working principle

To construct a computer capable of "human like thought researchers used the only working model they had available the human brain. To construct an artificial neural network the brain is not considered as a whole. Taking the human brain as a whole would be complex; rather the individual cells that make up the human brain are studied. At the most basic level the human brain is composed primarily of neuron cells.

The neurons forming the nervous system of any organism into three broad categories.

(i). Neurons which receive stimuli from the external environment i.e. sensory neurons.

(ii) Neurons which controls the actions of the organism mainly via contact with muscle cells, i.e. motor neurons.

(iii) Neurons which bring in contact other neurons, i.e. inter neurons.

A neuron cell as shown in the below fig is the basic building block of the human brain.

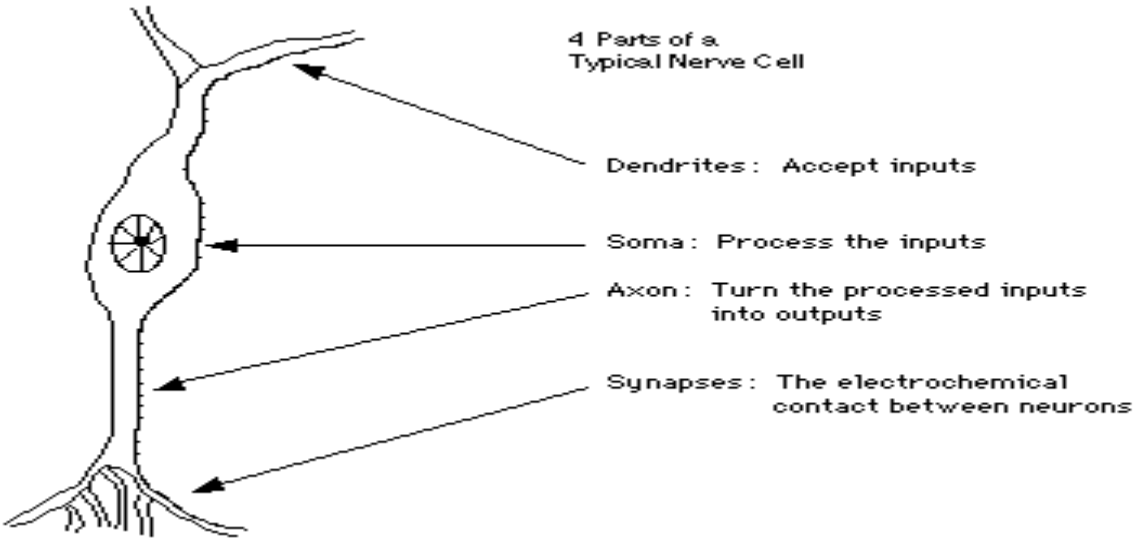

Fig3.1 Schematic diagram of a biological neuron

The main parts in a neuron cell are

1. Dendrites: accept inputs
2. Soma : process the inputs
3. Axon: turn the processed inputs into outputs
4. Synapse: the electrochemical contact between neurons

A accepts signals from the dendrites. When a neuron accepts a signal, its electric potential inside the body of cell raises or lowers. If this potential rises above the threshold value, it transmits to the other neuron through axon. This process is called firing of the neuron. .

Ultimately the signal will leave the neuron as it travels to the axon terminals. The signal is then transmitted to other neurons or nerves.

First the signals reaching a synapse and received by dendrites are electric impulses. Such signal transmission involves a complex chemical process in which specific transmitter substances are released from the sending side of the junction. This raises or lowers the electric potential inside the body of the receiving cell. The receiving cell fires if its electric potential reaches a threshold, and a pulse or action potential of fixed strength and duration is sent out through the axon to the axonal arborization to synaptic junctions to other neurons. After firing, a neuron has to wait for a period of time called the refractory period before it can fire again. Synapses are excitatory if they let passing impulses cause the firing of the receiving neuron, or inhibitory if they let passing impulses hinder the firing of the neuron.[1-6]

3.4. Characteristics of Artificial Neural networks:

An ANN is a parallel distributed information processing structure with the following characteristics.

1. It is a neutrally inspired mathematical model.

2. It consists of a large number of highly interconnected processing elements.

3. Its connections (weights) hold the knowledge.

4. A processing element can dynamically respond to its input stimulus, and the response completely depends on its local information .i.e. the input signals arrive at the processing element via implementing connections and connection weights.

5. It has the ability to learn, recall, and generalize from training data by assigning or adjusting the connection weights.

6. Its collective behavior demonstrates the computational power, and no single neuron carries specific information (Distributed representation property).

Because of these characteristics, other commonly used names for ANNs are parallel distributed processing models, connectionist models, self- organizing systems, neuro-computing systems, and neuromorphic systems.[2]

3.5. Electrical representation of neural network:

The information- processing abilities of neurons seem to have most natural interpretation in terms of electrical events. Neuron is surrounded by a thin membrane, and that large electrical and chemical differences exist between the inside and outside of the cell.

A standard model of the neuron membrane approximates it by a series of resistors and capacitors as shown in below fig2.2

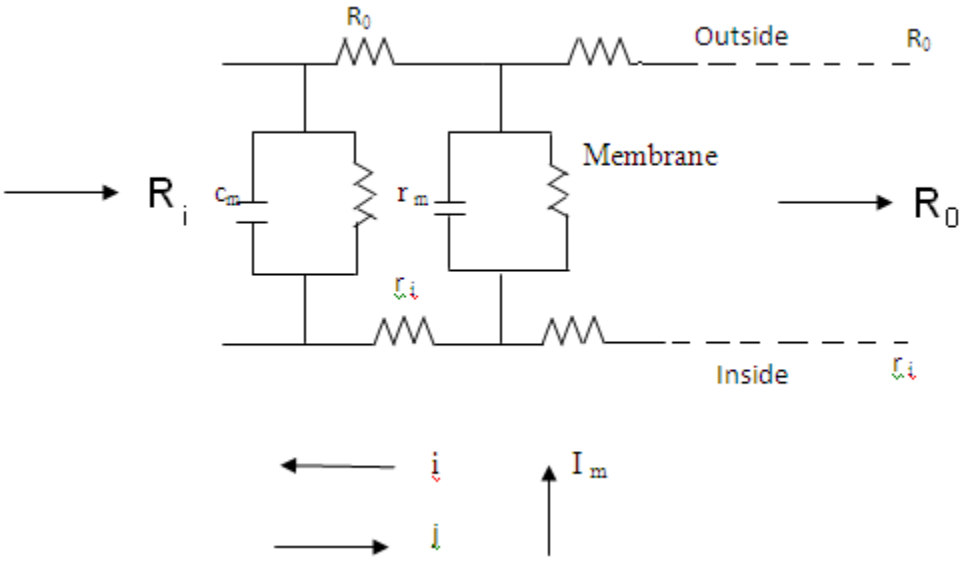

Fig 3.2.Electrical membrane model

The network extends to infinity on both ends. The resistive terms R_i and R_o are the resistors associated with the inside and outside resistances of the axoplasm and external medium. The terms R_m and C_m are the resistance and capacitance of the membrane. Each repeating element in the membrane model has its own resistance and capacitance, and each stands for a certain small length of membrane.

Behavior of the circuit:

(i) Time domain

With membrane resistance, r_m and voltage v, then by ohm's law,

The current in the circuit is $i = V/R_m$…….. (1)

From the elementary physics, we know that Q=CV…….. (2)

Where Q is the charge stored in the capacitor, C is the capacitance in farads, and V is the voltage across the capacitor in volts.

$$i=dQ/dt=CdV/dt \quad \text{…….(3)}$$

Therefore the total current across membrane will be

$$I_m = V/r_m + c_m\, dV/dT \quad \text{……} \quad (4)$$

Where C_m is the membrane capacitance. Qualitatively, it is easy to see that the larger the capacitor, the greater the current required to change the voltage required in the circuit.

(ii) Space domain

Assume we have the network of resistors and capacitors as shown in above fig2.2. A voltage is established at a particular point on the network. Current flows down the axoplasm and some will leak out through the membrane. Suppose that flow through the membrane in distance Δx is sufficient to cause a drop in voltage. Suppose the total drop in voltage in this segment of the resistor network is ΔV. Notice that this voltage drop is measured as a function of x. because we measure resistance in ohms per unit of length, total resistance in the circuit is Δx(ri+ro), the product of length times resistance per unit length. Then the current I, down the axon and returning through the external medium is given by

i= voltage/resistance……. (1)

$i = \Delta V/\Delta x\,(ri+ro)$…… (2)

The distances and voltages involved are very small,

$i = (1/ro+ri)\,(dV/dx)$……. (3)

Current at a point in the network must be equal to the sum of the flow through the membrane and the flow down the axon. Therefore, the change is current going down the axon must be equal to the current through the membrane at that point.

$$di/dx=i_m \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4)$$

By differentiating the above equation we get the current.

$$i_m =(1/ro+ri)\ (dV/dx)^2 \ldots\ldots\ldots\ldots\ldots\ldots (5)$$

Clearly membrane current is membrane current, so we can equate two terms.

Time constant $\tau= r_m c_m$ and …… (6)

Space constant $\lambda^2 = r_m/ro+ri$ ……. (7)

Equating the two expressions for membrane current, we have an expression for cable equation

$V= \lambda^2 \partial^2 V/\partial x^2 - \tau\ \partial V/\partial t$ …… (8)

Here consider two special cases

First assume that in the dim past we established a current distribution with voltage V0 at a single point on the membrane, and the current is no longer changing with time. Then we can solve the resulting differential equation,

$$V = \lambda^2 \partial^2 V/\partial x^2 \text{ and}$$

$$V(x) = V_0\ e^{-x/\lambda} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (9)$$

This means that an exponential falloff of voltage will occur due to simulation at a point. For real neurons, typical values of length constant would be on the order of a few millimeters.

Let us assume that there is no change of voltage in the axon with x. it is easy to show that a change in voltage with a negative exponential, with time constant τ. If the initial voltage on the membrane is zero and we apply a 1-v step of voltage, the final voltage will be 1V. Voltage as a function of time V (t), will be

$$V(t) = (1 - e^{-t/\tau}) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (10)$$

A typical value of membrane time constant might be 1 or 2 msec. This suggests that signals changing at a rate of more than 1 KHZ will have trouble passing through a neuron, whatever form the actual transmission takes. [9],[10]

3.6 Problems suited to neural networks

Neural networks can often solve problems with fewer lines of code than a traditional programming algorithm. It is important to understand what these problems are. Neural networks are particularly adept at solving problems that cannot be expressed as a series of steps. Neural networks are particularly useful for recognizing patterns, classification into groups, series prediction and data mining. Pattern recognition is perhaps the most common use for neural networks. The neural network is presented a pattern. This could be an image, a sound, or any other sort of data. The neural network then attempts to determine if the input data matches a pattern that the neural network has memorized.[4]

3.7 Problems not suited to neural networks

It is important to understand that a neural network is just a part of a larger program. A complete program is almost never written just as a neural network. Most programs do not require neural network. Programs that are easily written out as a flowchart are an example of programs that are not well suited to neural networks. If your program consists of well defined steps, normal programming techniques will suffice. Another criterion to consider is whether the logic of your program is likely to change. The ability for a neural network to learn is one of the primary features of the neural network. If the algorithm used to solve your problem is an unchanging business rule there is no reason to use a neural network. It might be detrimental to your program if the neural network attempts to find a better solution, and begins to diverge from the expected output of the program. Finally, neural networks are often not suitable for problems where you must know exactly how the solution was derived. A neural network can become very adept at solving the problem for which the neural network was trained. But the neural network cannot explain its reasoning. The neural network knows because it was trained to know. The neural network cannot explain how it followed a series of steps to derive the answer.[4]

3.8 Neural network Structure

A neural network is composed of several different elements. Neurons are the most basic unit. Neurons are interconnected. These connections are not equal, as each connection has a connection weight. Groups of networks come together to form layers.

3.8.1. The Neuron

The neuron is the basic building block of the neural network. A neuron is a communication Conduit that both accepts input and produces output. The neuron receives its input either from other neurons or the user program. Similarly the neuron sends its output to other neurons or the user program. When a neuron produces output, that neuron is said to activate, or fire. A neuron will activate when the sum if its inputs satisfies the neuron's activation function. Consider a neuron that is connected to k other neurons. The variable w represents the weights between this neuron and the other k neurons. The variable x represents the input to this neuron from each of the other neurons. Therefore we must calculate the sum of every input x multiplied by the corresponding weight w. This is shown in the following equation. This sum must be given to the neurons activation function. An activation function is just a simple Java method that tells the neuron if it should fire or not. For example, if you chose to have your neuron only activate when the input to that neuron is between 5 and 10, the following activation method might be used. [1-4]

3.8.2. Neuron Connection Weights

Neurons are usually connected together. These connections are not equal, and can be assigned individual weights. These weights are what give the neural network the ability to recognize certain patterns. Adjust the weights, and the neural network will recognize a different pattern. Adjustment of these weights is a very important operation. [1-2]

3.8.3. Neuron Layers

Neurons are often grouped into layers. Layers are groups of neurons that perform similar functions. There are three types of layers. The input layer is the layer of neurons that receive

input from the user program. The layer of neurons that send data to the user program is the output layer. Between the input layer and output layer can are hidden layers. Hidden layer neurons are only connected only to other neurons and never directly interact with the user program. Figure shows a neural network with one hidden layer. Here you can see the user program sends a pattern to the input layer. The input layer presents this pattern to the Hidden layer. The hidden layer then presents information on to the output layer. Finally the User program collects the pattern generated by the output layer.

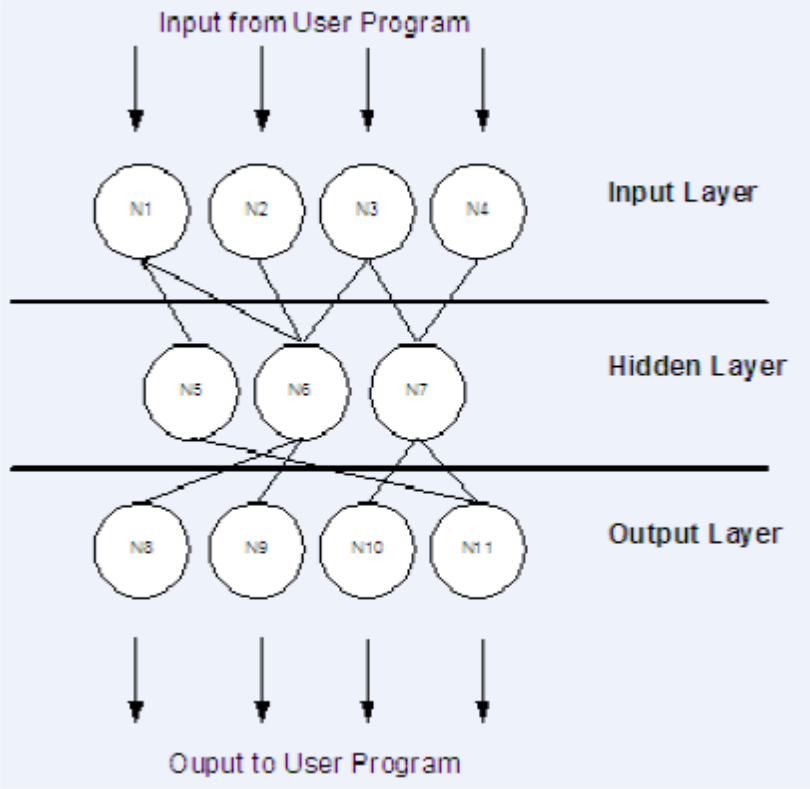

Fig3.3. Neural network layers

The input and output layers are not just there as interface points. Every neuron in a neural Network has the opportunity to affect processing. Processing can occur at any layer in the neural network. Not every neural network has this many layers. The hidden layer is optional. The input and output layers are required, but it is possible to have on layer act as both an input and output layer. Later in this chapter you will be shown a Hopfield neural network. This is a Single layer (combined input and output) neural network. Now that you have seen how a neural network is

constructed you will be shown how neural networks are used in pattern recognition. Finally, this chapter will conclude with an implementation of a single layer Hopfield neural network that can recognize a few basic patterns.[1-2]

3.9 Main advantages of neural networks

Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an expert in the category of information it has been given to analyse.This expert can then be used to provide projections given new situations of interest and answer "what if "questions

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.

2. Self-Organization: An ANN can create its own organization or representation of the information it receives during learning time.

3. Real time operation: ANN computations may be carried out in parallel and special hardware devices being designed and manufactured which take advantage of this capability.

4. 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.[1-3]

3.10. Mathematical model representation of neural network:

The first mathematical model of a neuron was proposed by McCulloch and Pitts (M-P) in 1943. It was a binary device using binary inputs, binary output, and a fixed activation threshold. In general, a model of an artificial neuron is based on the following parameters which describe a neuron. Figure (2) shows a simple mathematical model of the above mentioned biological neuron proposed by McCulloch and Pitts, usually called an M-P neuron.

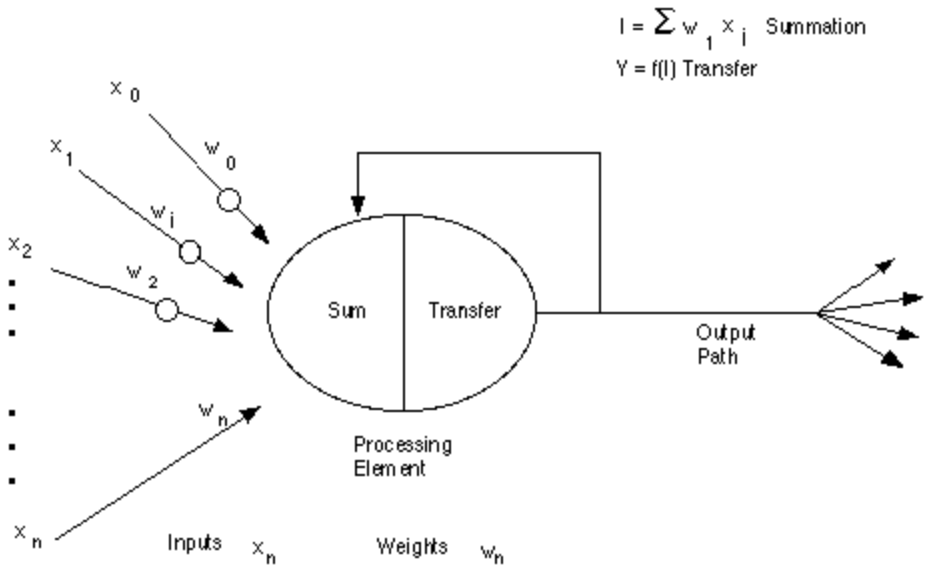

Fig 3.4. Mathematical model of an M-P neuron

The basic elements of a neuron model are

1. A set of synapses or connecting links
2. An adder
3. An activation function
4. An output of the neuron

1. A set of synapses or connecting links: A set of synapses or connecting links, each of which is characterized by a weight or strength of its own. Unlike a synapse in the brain, the synaptic weight of an artificial neuron may lie in a range that includes negative as well as positive values.

2. An adder: An adder for summing the input signals, weighted by the respective synapses of the neuron.

$$\text{Summation function} = \sum_{i=1,\, n} xi.wi \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (1)$$

Where x_i is input of the neuron, w_i is the weight vector of the neuron

3. An activation function: An activation function limiting the amplitude of the output of a neuron. The activation function is also referred to as a squashing function in that it squashes (limits) the permissible amplitude range of the output signal to some finite value

Activation function A=s (u)… …… (2)

4. 4. An output function: An output function calculates the output signal value emitted through the output (the axon) of the neuron

Output function= g (A)…… (3)

3.11 Different types of activation functions:

1. Sigmoid function:

This is any s- shaped non linear transformation function g(u) that is characterized by the following.

(a) Bounded, i.e. its values are restricted between two boundaries, for example, [0, 1], [1,-1].

(b) Monotonically increasing, i.e. the value g(u) never decreases when u increases

(c) Continuous and smooth, therefore differentiable everywhere in its domain.

One important thing to note about the sigmoid layer is that it only positive values are returned. If you need negative numbers as a result from the neural network, the sigmoid function will be unsuitable.

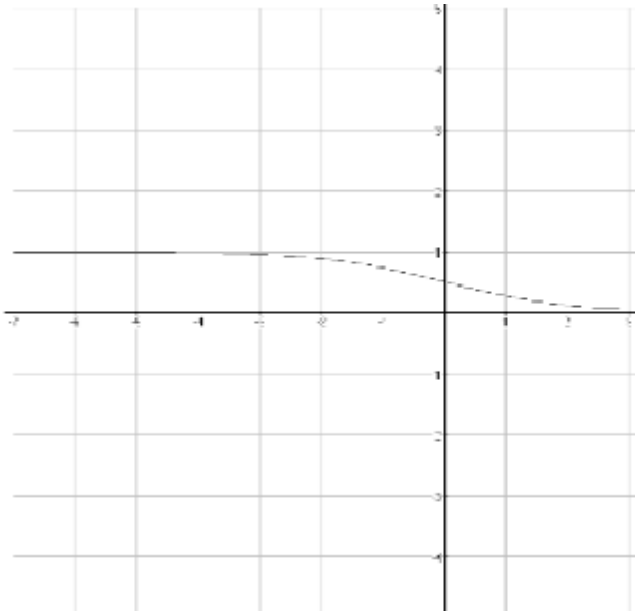

Fig3.5 response of sigmoid function

2. The Tanh function

The main advantage of using tanh layer is it is suitable for both positive as well as negative values.

$$y(u) = \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}} \quad \text{..... (5)}$$

The graph for the

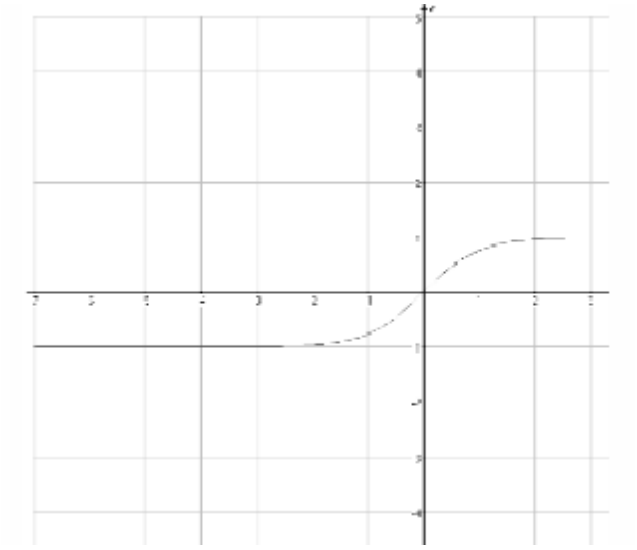

Fig 3.6 tanh response

3. Linear function:

The Linear Layer is essentially no layer at all. The linear layer does no modification on the Pattern before outputting it. The function for the linear layer is given as follows. The Linear Layer is useful in situations when you need the entire range of numbers to be outputted. Usually you will want to think of your neurons as active or non-active. Because the Tanh Layer and Sigmoid Layer both have established upper and lower bounds they tend to be used for more Boolean(on or off) type operations. The Linear Layer is useful for presenting a range. The graph of the linear layer is given in Figure

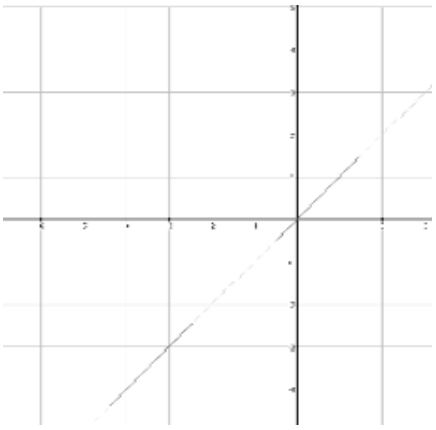

Fig3.7. liner function response

4. The Gaussian function:

An output signal from a neuron can be represented by a single static potential, or by a pulse, which either occurs or does not occurs. [1-4]

3.12. Neuron models

(i) Single-input neuron

A single input neuron is shown in below fig. The scalar input p is multiplied by the scalar weight w to form WP, one of the terms that are sent to the summer. The other input, 1 is multiplied by a bias b and then passed, to the summer. The other input 1 is multiplied by a bias b and then passed to the summer. The summer output n, often referred to as the net input, goes into a transfer function f, which produces the scalar neuron output a.

By comparing this model with biological neuron, the weight w corresponds to the strength of a synapse, the cell body is represented by the summation and the transfer function, and the neuron output a represents the signal on the axon.[9]

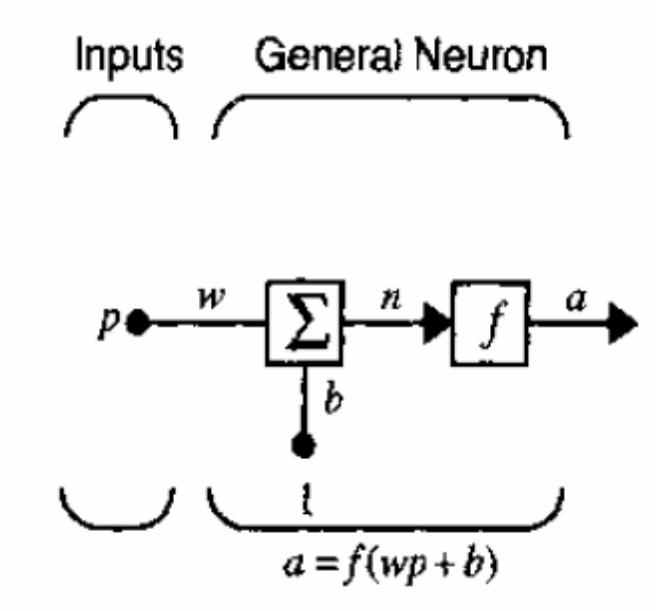

Fig3.8. Single-input neuron

The neuron output is calculated as a =f (WP+b).

(ii) Multiple- input neuron:

Typically a neuron has more than one input is termed as multiple-input neuron. A neuron with R inputs is shown in below fig The individual inputs p_1, p_2, p_3,…… pr are each weighted by corresponding elements $w_{1,1}$, $w_{1,2}$…… $w_{1,r}$ of the weight matrix W [9]

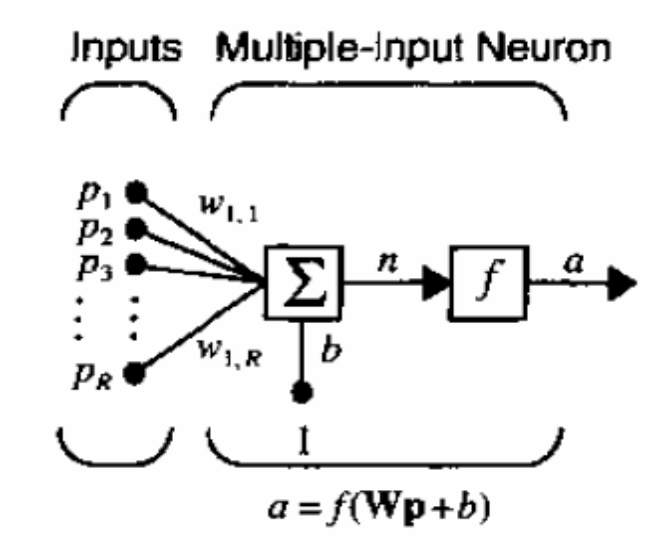

Fig3.9. Multiple- input neuron

The neuron has a bias b, which is summed with the weighted inputs to form the net input n

$N= w_{1,1} p_1 + w_{1,2} p2 + \ldots\ldots\ldots\ldots + w_{1,r} pr + b$

This expression can be written in matrix form, n =WP +b

The neuron output can be written as a =f (WP+b)

3.13 Classification of neural networks:

Neural networks can be broadly classified into two types: they are (I) Feed forward networks (II) Recurrent/feedback networks

(I) feed forward networks

There are no connections back from the output to the input neurons; the network does not keep a memory of its previous output values and the activation states of its neurons; the perceptron-like networks are feed forward types.

(II)Recurrent/ Feedback networks

There are connections from output to input neurons; such a network keeps a memory of its previous states, and the next state depends not only on the input signals but on the previous states of the network; the Hopfield network is of this type.

Feed forward networks can be again classified into the following categories

1. Single layer perceptron, 2.Multi layer perceptron, 3.Radial basis function networks.

Feedback networks can be again classified into the following categories

1. Competitive networks, 2.Kohonens SOM networks, 3.Hopfield network, 4.ART models.

1. Single layer feed forward ANN

A layer is a set of neurons or computational nodes at the same level. A set of inputs can be applied to this single layer of neurons. This would then become a single layer FF network. This single layer could be called the "output layer" (OL). Each node in the OL is called an output neuron.[1]

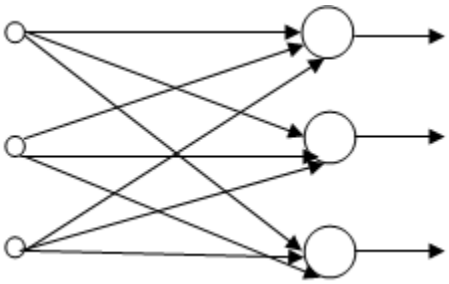

Fig 2.7 single layer FF ANN

2. Multilayer feed forward ANN

This structure can be extended to a multi-layer FF ANN by adding one or more layers to the existing network. These additional layers then become "hidden layers" (HL). Each node in the HL is called a hidden neuron. They appear as intermediate neurons between the input and the output layers.[1]

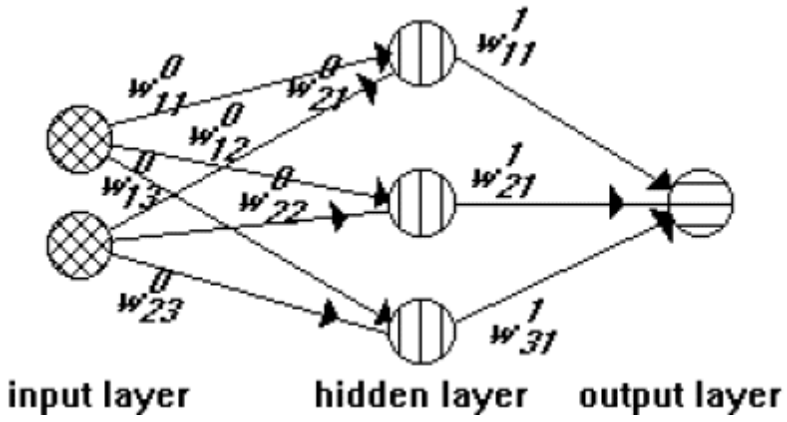

Fig 2.8 Multi layer FF ANN

In Figure 2.8, the three layers of an ANN are shown. The first layer is the input layer where the input data vector is passed into the network. In Fig. 2.8, the input is a 2 dimensional vector. Following that is the hidden layer containing 3 hidden neurons. There are weights acting

on the input data as it is passed into the hidden layer. These are the weights that are modified while training the network using training algorithms. Finally, following the hidden layer there is the output layer which has just one output neuron. Again, there are weights acting as connections between the hidden and output layers.

Each neuron acts as a local receptive field. But, when layers of neurons are added, then these layers gain the ability to extract statistical features from the input data vectors. Each node in the input layer is connected to every other neuron in the hidden layer (the next forward layer). Similarly each hidden neuron is, in turn, connected to every output neuron. Thus the architecture is such that the output response from a neuron in one layer is applied as input to neurons in the next forward layer. Suppose that there are 5 input neurons, 4 hidden neurons and 3 output neurons in an ANN, this ANN would be known as 5-4-3 network.

3. Radial Basis Function (RBF) ANN

In statistics, the concept of an RBF was introduced to solve multivariate interpolation problems. RBFs are powerful tools in "curve fitting" and "function approximation" problems. This idea was extended to an ANN and gave rise to an RBF ANN. Basically the RBF ANN looks similar to the FF ANN, but the only difference is that only one hidden layer is used in the case of the RBF ANN whereas in a FF ANN, more than one layer can be used. This hidden layer transforms the input vector (from the input layer) to a higher dimension, by applying a nonlinear transformation from the input space to the hidden layer vector space. Then, from the hidden layer, there is a linear transformation to the output layer. This ANN would be suitable for application to a pattern classification problem, because at a higher dimension, data can be classified well than at a lower dimension (geometrically speaking). The dimension of the hidden layer is defined by the number of hidden neurons. This is the reason why, in an RBF ANN, the dimension of the hidden layer is generally much higher than that of the input layer (dimension of the input vector). [1-2]

4. 4. Self organizing maps (SOM)

The Self-Organizing Feature Map is a special class of artificial neural nets. It is loosely based on the way the brain functions. Different sensory inputs to the brain, such as visual,

auditory and the like are mapped into the brain in different zones or areas. SOM's are artificial mappings that learn through self-organization. The hidden neurons of a SOM sort-out or rather separate the input feature vectors into different domains based on the hidden neuron centers. Here, the Euclidean distances between an input pattern and the centers of the hidden neurons are calculated. The center with the minimum distance from the input vector is the "winner". Then, at the output layer, the input patterns are mapped onto a two dimensional topographic map.[1-2]

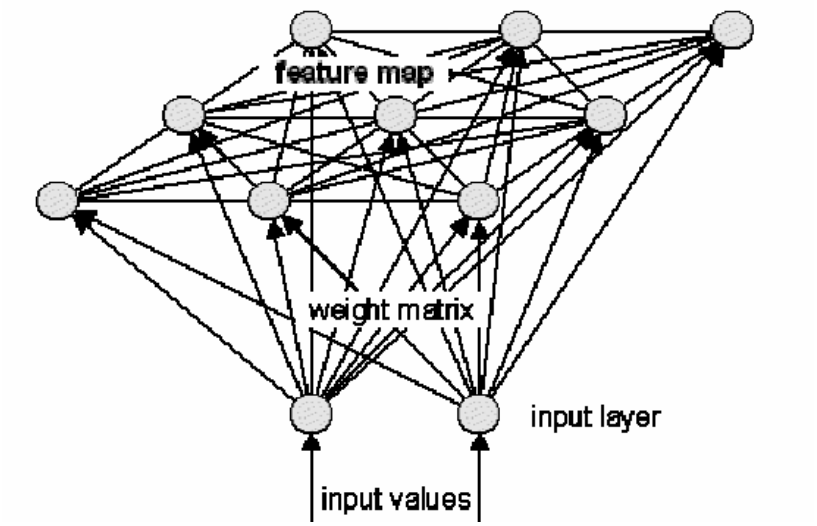

Fig 2.9 structure of self organizing map

3.14. Conclusions:

This chapter presents the brief view about neural networks, working principle of neural networks. Also this chapter explains the characteristics of artificial neural networks, electrical representation of neural network. Also this chapter explains the mathematical model representation of neural network, different types of activation functions, and different types of neural network models.

Chapter IV

4.1 Introduction

Lab view is an integral part of virtual Instrumentation. It provides an easy to use applications development environment specifically with the needs of engineers and scientists. Lab VIEW is a graphical programming language for developing diverse applications in a multitude of industries. The block diagram provides a unique form of source code expression that is dissimilar to most programming languages and development environments. The data flow paradigm represents the program as wires, terminals, structures, and nodes rich with functionality and innovation. Lab VIEW extends this innovation to the developer, providing tremendous freedom of expression and creativity. As such, there are many means to an end, or possible development styles, with Lab VIEW. [16-17]

4.2 Virtual Instrumentation

A virtual instrumentation system is software that is used by the user to develop a computerized test and measurement system, for controlling an external measurement hardware device from a desktop computer. Virtual instrumentation also extends to computerized systems for controlling processes based on the data collected and processed by a PC based instrumentation system. [16].

A virtual instrument is composed of the following blocks:

– Sensor module
– Sensor interface
– Information systems interface
– Processing module
– Database interface
– User interface

Figure 4.1 shows the general architecture of a virtual instrument. The sensor module detects physical signal and transforms it into electrical form, conditions the signal, and

transforms it into a digital form for further manipulation. Through a sensor interface, the sensor module communicates with a computer. Once the data are in a digital form on a computer, they can be processed, mixed, compared, and otherwise manipulated, or stored in a database. Then, the data may be displayed, or converted back to analog form for further process control. Virtual instruments are often integrated with some other information systems.

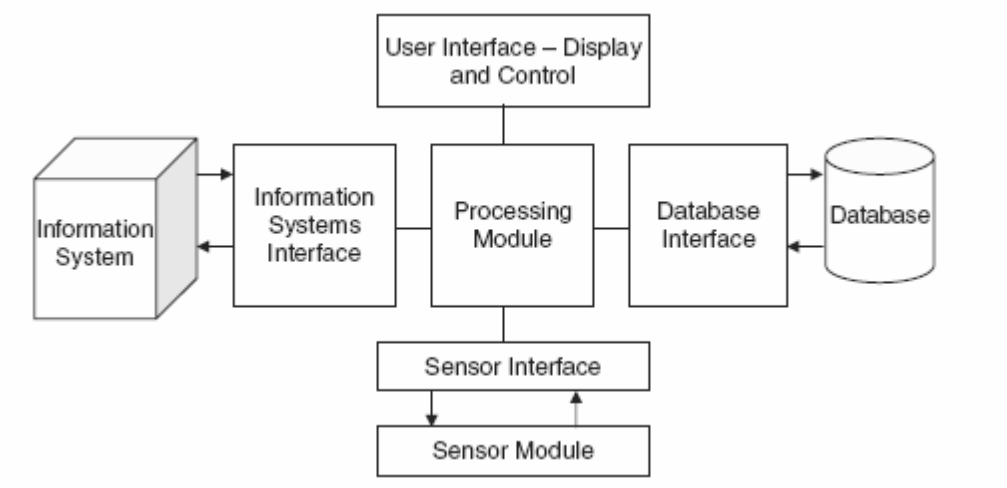

Fig 4.1 Architecture of a virtual instrument

4.3 Virtual Instruments versus Traditional Instruments

Lab VIEW is a virtual instrument (VI). Virtual instruments are software based functional instruments, whereas Traditional instruments are hardware functional specified.

Traditional instruments:

- Vendor defined, Function-specific, stand-alone hardware with limited connectivity
- Hardware is the key, Expensive
- Closed, fixed functionality, slow turn on technology (5–10 year life)
- Minimal economics of scale

Virtual Instruments:

- User-defined, Application- oriented system with connectivity to networks, peripherals, and applications.
- Software is the key, low cost, reusable
- Open, flexible functionality leveraging off familiar computer technology
- Maximum economics of scale
- Software minimizes development and maintenance costs

4.4 Lab VIEW

Lab VIEW is a graphical programming language for developing diverse applications in a multitude of industries. The block diagram provides a unique form of source code expression that is dissimilar to most programming languages and development environments. The data flow paradigm represents the program as wires, terminals, structures, and nodes rich with functionality and innovation. Lab VIEW extends this innovation to the developer, providing tremendous freedom of expression and creativity. As such, there are many means to an end, or possible development styles, with Lab VIEW. [16-20].

4.5 Features of Lab VIEW

Lab VIEW is a highly interactive environment platform for rapid prototype and incremental development of applications, such as measurement and automation to real time embedded and general purpose applications

The Lab VIEW graphical development environment offers the program codes simply by connecting icons. Consequently, Lab VIEW is a multitasking system capable of concurrently running multiple VI's .It helps the user to do more projects done in less time, by streamlining the process form inception to completion.

Easy to use: Lab view continuously improves based on usability research, customer feedback, and better technology. Lab VIEW simplifies even the most challenging tasks, such as multi threaded parallel execution, through Lab VIEW patented dataflow technology.

Complete functionality: With Lab VIEW the user can easily scale to meet current and feature requirements, whether it is required to create a small and easy to use instrument control application or a solution that monitors and controls the entire factory. [16-17].

4.6 Present Draw backs of Lab VIEW:

Some features of popular Languages are missing in Lab VIEW, but some elaborate general purpose complex applications can be built on its platform even without data acquisition or analysis. The application builder is not truly standalone and requires the Lab VIEW run-time engine on target computer on which users run the application. The use of standard controls require a runtime library for a language and all major operating system suppliers supply the required libraries, however , the runtime required for Lab VIEW is not supplied and is required to be specifically installed by the administrator or user. This requirement can cause problem if an application is distributed to a user who may be prepared to run the application but does not have the inclination or permission to install additional files on the host system prior to running the executable. [16-17].

4.7 Main parts of Lab view

Lab VIEW has many interesting features that make it a very useful tool for building neural nets. Lab VIEW programs are called Virtual Instruments (VI), because they appear very similar to actual laboratory instruments. There are three parts to a VI –the 'Front Panel', the 'Block Diagram' and Icon or connector plane. [20-21].

4.7.1Front panel window:

The front panel constitutes one part of the program in which the GUI is developed. Controls, constants and indicators form an integral part of the front panel. Examples for controls and indicators include knobs, horizontal and vertical sliders, buttons, input devices, graphs and charts. Lab VIEW utilizes a powerful Graphical User Interface (GUI). The Front panel acts as the user interface. The front panel, as the name suggests acts as the front-end of the virtual Instrument, while the block diagram is the background code. Fig4.1 shows an example of a front panel window. [20].

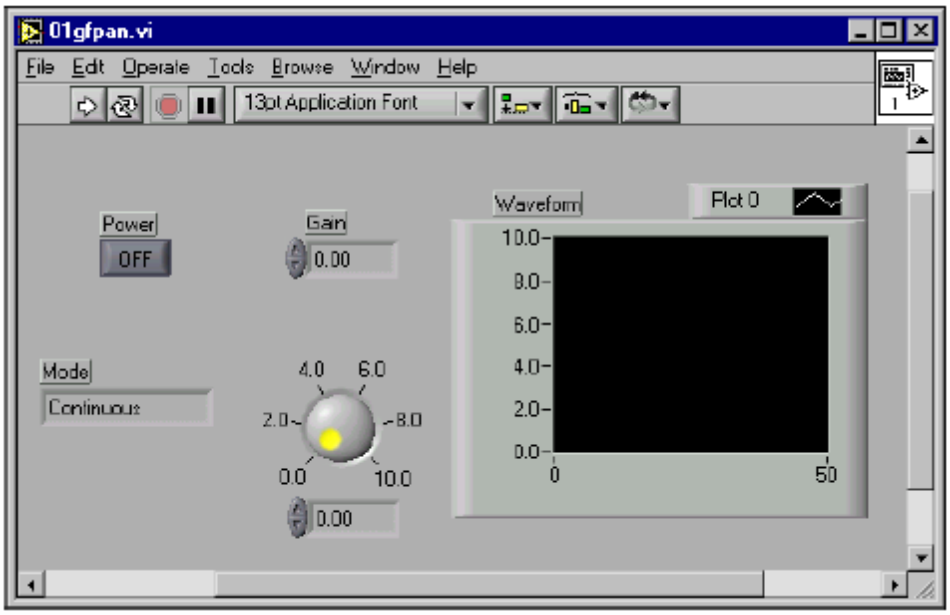

Fig 4.2 VI front panel

Main Parts in front panel:

Controls and indicators: you can create the front panel window with controls and indicators, which are the interactive input and output terminals of the VI, respectively. Different types of control and indicators are (i) Numeric controls and indicators: The numeric data type can represent numbers of various types, such as integer or real.

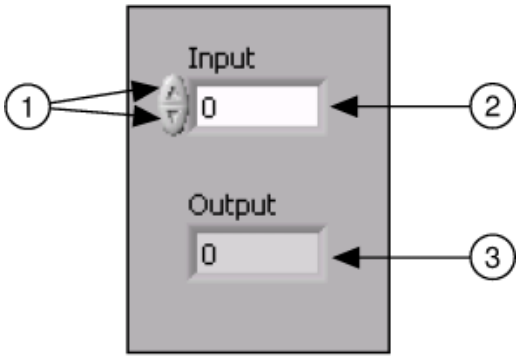

Fig4.3 Controls and indicators

In the above diagram 1 represents increment or decrement button,2 represents numeric control, 3 represents numeric indicator respectively.

(ii) Boolean controls and indicators: The Boolean data type represents data that only has two parts, such as TRUE and FALSE or ON and OFF. Generally Boolean controls and indicators are used to enter and display Boolean values. Boolean objects simulate switches, push buttons, and LEDs.

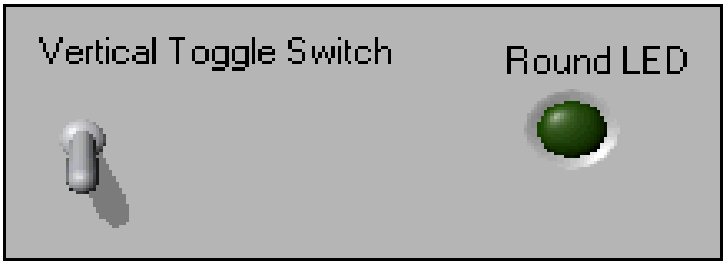

Fig 4.4 Boolean controls and indicators

(iii)String controls and indicators: The string data type is a sequence of ASCII characters. Use string controls to receive text from the user such as a password or user name.

4.7.2 Block diagram window:

The block diagram constitutes the actual program part of Lab VIEW. It is highly graphical, and no text codes are involved. The block diagram follows a similar idea as the 'data flow diagram' concept, in which the logic is presented as a diagram rather than as text code. [20].

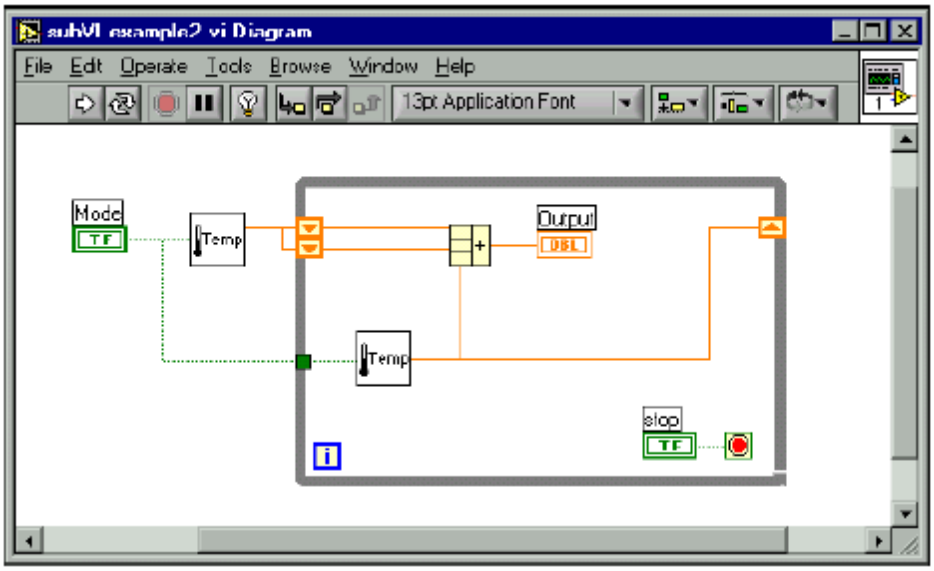

Fig4.5 VI Block diagram

After building the front panel, codes could be added by using graphical representations of functions to control the front panel objects. The block diagram contains this graphical source code. Front panel objects appear as terminals on the block diagram. Block diagram objects include terminals; sub vis, functions, structures, and wires, which transfer data among other block diagram objects.

TERMINALS:

The terminals represent the data type of the control (input) or indicator (output). The front panel controls or indicators can be configured to appear as icons or data type terminals on the block diagram. By default, front panel objects appear as icon terminals. The terminals in the block diagram represent the data type as double precision, floating point or any other format of that type.

Terminals are entry and exit ports that exchange information between the front panel and the block diagram. The data that a user enters into the front panel controls' enter the block diagram through the control terminals. After the functions complete their 5 calculations in the block diagram, the data flow to the indicator terminals, where they exit the block diagram, reenter the front panel, and appear as front panel indicators. Nodes are objects on the block diagram that have inputs and/or outputs and perform operations when a VI runs. They are analogous to statements, operators, functions, and subroutines in text-based programming languages. [16], [20].

NODES

Nodes are objects on the block diagram that have inputs and/or outputs and perform operations when a VI runs. A node is just a fancy word for a program execution element. Nodes are analogous to statements, operators, functions, and subroutines in standard programming languages. A structure is another type of node. Structures can execute code repeatedly or conditionally, similar to loops and Case statements in traditional programming languages. Lab VIEW also has special nodes, called Formula Nodes, which are useful for evaluating mathematical formulas or expressions. [16], [20].

WIRES

Data transfer between block diagram objects is done through wires. In Figure 1.1, wires connect the control and indicator terminals to the multiply node. Each wire has a single data source, but one can wire it to many VI's and functions that read the data. Wires are different colors, styles, and thicknesses, depending on their data types. A broken wire appears as a dashed black line with a red X in the middle. [20].

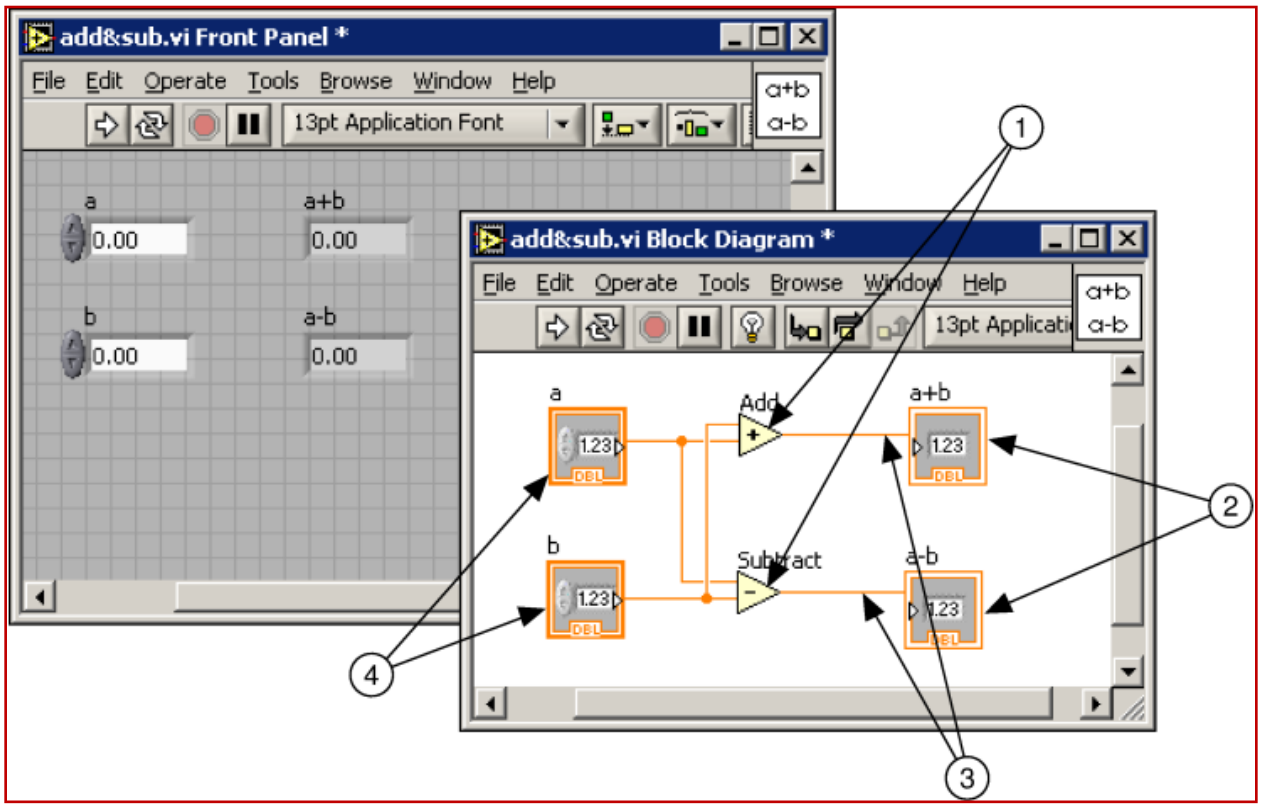

Fig 4.6 1. Nodes, 2. Indicator Terminals, 3.Wires, 4. Control Terminals

4.7.3 Icon and connector pane

After building a VI front panel and block diagram, one needs to build the icon and the connector pane so that this VI could be used as a sub-VI. Every VI displays an icon, such as the one shown in figure in the upper right corner of the front panel and block diagram windows. An icon is a graphical representation of a VI. It can contain text, images, or a combination of both. If one uses a VI as a sub-VI, the icon identifies the sub- VI on the block diagram of the VI. One can double-click the icon to customize or edit it. One also needs to build a connector pane, to use

the VI as a sub-VI. The connector pane is a set of terminals that correspond to the controls and indicators of that VI, similar to the parameter list of a function-call in text-based programming languages. The connector pane defines the inputs and outputs one can wire to the VI so that it could be used as a sub-VI. A connector pane receives data at its input terminals and passes the data to the block diagram code through the front panel controls and receives the results at its output terminals from the front panel indicators. When the connector pane is viewed for the first Time, one can see a connector pattern. A different pattern could be selected if necessary. The connector pane generally has one terminal for each control or indicator on the front panel. Up to 28 terminals could be assigned to a connector pane. If any changes are anticipated to the VI that would require a new input or output, one can leave extra terminals unassigned. [16], [20].

Fig4.7 Icon and connector pane

4.8 Controls, functions, and tools palette:

Controls palette: The control palette is available only on the front panel. The Controls palette contains the controls and indicators used to create the front panel. The controls and indicators are located on sub-palettes based on the types of controls and indicators

Functions palette: The functions palette is available only on the block diagram. The Functions palette contains the VI's and functions one can use to build the block diagram. The VI's and functions are located on sub-palettes based on the types of VI's and functions.

Tools palette: The tools palette is available on the front panel and the block diagram. A tool is a special operating mode of the cursor. The cursor selects an icon corresponding to a tool in the palette. One can use the tools to operate and modify front panel and block diagram objects.
The above three palettes shown in below fig 4.7 [19], [20].

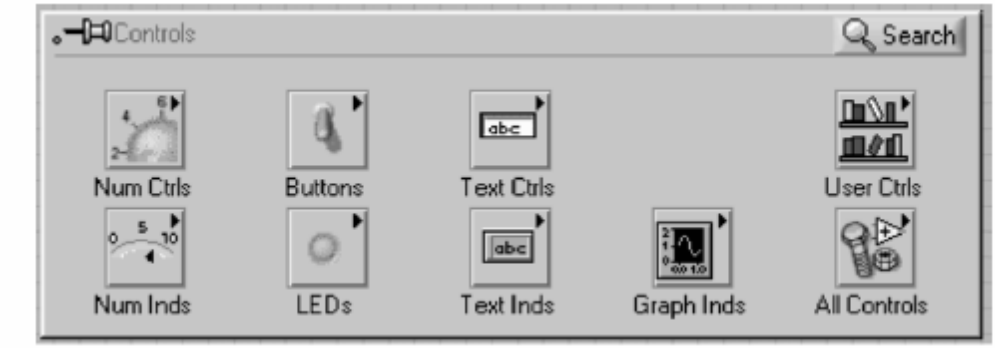

Fig 4.8a Controls palette

Fig 4.8b Functions palette

Fig 4.8c Tools palette

4.9 STRUCTURES:

Structures are graphical representations of the loops and case statements of text-based programming languages. Structures located on the block diagram window are used to repeat blocks of code and to execute code conditionally or in a specific order. Lab VIEW includes the following structures – the While Loop, For Loop, Case structure, Stacked Sequence structure, Flat Sequence structure, Event structure, and Formula Node which are described in this section. This section also describes on arrays which are used to group data elements of the same type and clusters which group data of different type. In the later sections, the section focuses on waveform charts, waveform graphs, XY graphs, and intensity plots along with strings and files I/Os. [20].

4.9.1 Different types of structures

(i)Control structures

In any programming language the user probably requires repetition of a section of code. Lab VIEW offers two loop structures namely, the For Loop and While Loop to control repetitive operation in a VI. A For Loop executes a specific number of times; a While Loop executes until a specified condition is no longer true.

(a) FOR LOOP

A *For Loop* executes the code inside its borders, called its *sub diagram*, for total of *count* times, where the count equals the value contained in the *count terminal*. The count can be set by wiring a value from outside the loop of the count terminal. If '0' is wired to the count terminal, the loop does not execute. The For Loop is shown in Fig.4.1 the *iteration terminal* contains the current number of completed loop iterations; 0 during the first iteration, 1 during the second, and so on, up to N-1 (where N is the number of times the loop executes).

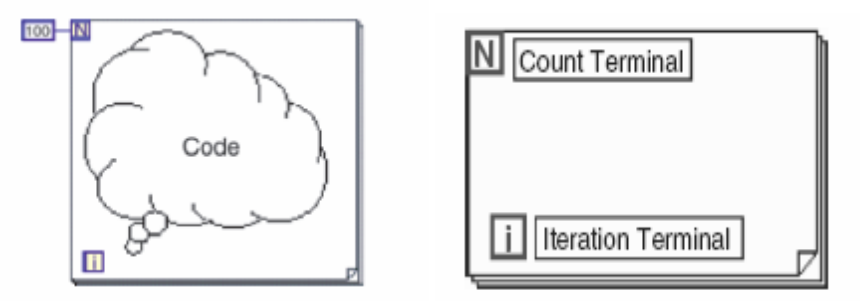

Fig 4.9 for loop

Count Terminal:

The value in the count terminal (an input terminal), shown in Fig. 4.8 indicates how many times to repeat the sub diagram.

Iteration Terminal:

The iteration terminal (an output terminal), shown in Fig. 4.8 contains the number of iterations completed. The iteration count always starts at zero. During the first iteration, the iteration terminal returns 0.

Example of for loop:

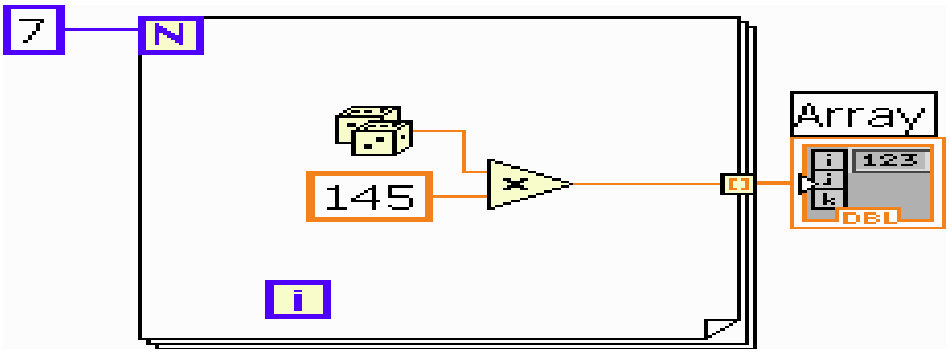

Fig 4.10 block diagram example to illustrate using "for" loop with indexing

Figure shows a simple "for loop" structure that repeats 7 times. Each time, the output value is stored in memory. After the loop repeats 7 times, the stored values form an output array.

(b) While loop

The While Loop executes the sub diagram inside its borders until the Boolean value wired to its conditional terminal is FALSE. Lab VIEW checks the conditional terminal value at the *end* of iteration. If the value is TRUE, the iteration repeats. The default value of the conditional terminal is FALSE, so if left unwired, the loop iterates only once. The While Loop's *iteration terminal* shown in Fig. 3.14 behaves exactly like the one in the For Loop. The While Loop is equivalent to the following pseudo code:

Do

Execute sub diagram

While condition is TRUE

We can also change the state that the conditional terminal of the While Loop checks, so that instead looping *while true*, we can have it loop *unless it's true*. To do this, we pop-up on the conditional terminal, and select **"Stop if True."**

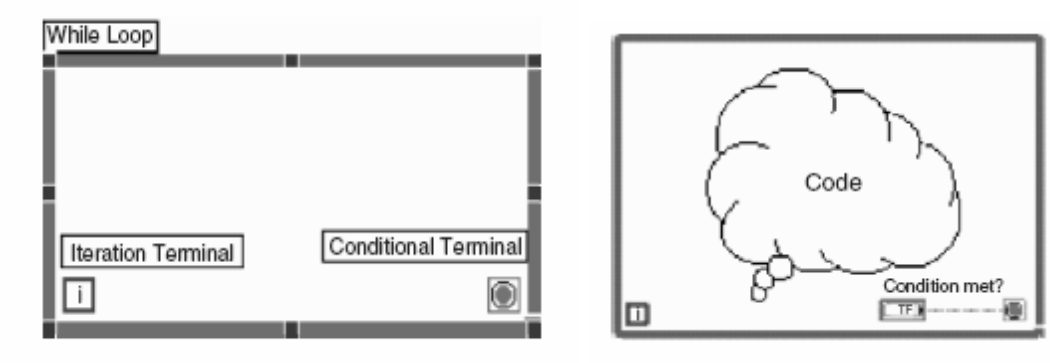

Fig 4.11 While loop

Condition Terminal: The While Loop executes the sub diagram until the conditional terminal, an input terminal, receives a specific Boolean value. The default behavior and appearance of the conditional terminal is **Stop If True**. When a conditional terminal is **Stop If True**, the While Loop executes its sub diagram until the conditional terminal receives a True value.

Iteration Terminal: The iteration terminal, an output terminal, contains the number of completed

Iterations. The iteration count always starts at zero. During the first iteration, the iteration terminal returns 0.

4.10. SHIFT REGISTERS:

Shift registers, available for while Loops and For Loops, are a special type of variable used to transfer values from one iteration of a loop to the next. They are unique to and necessary for Lab VIEW's graphical structure. Popping-up on the left or right loop border and selecting Add Shift Register from the Pop-up menu can create a shift register as in Fig. 4.11

A shift register comprises a pair of terminals directly opposite to each other on the vertical sides of the loop border. The right terminal stores the data upon the completion of iteration. These data are "shifted" at the end of the iteration and appear in the left terminal at the beginning of the next iteration. A shift register can hold any data type – numeric, Boolean, string, array, and so on. The shift register automatically adapts to the data type of the first object that is

wired to it. It appears black when first created, but the shift register assumes the color of the data type wired to it. [16], [20].

(ii) SELECTION STRUCTURES

Depending on the flow of data there are cases when a decision must be made in a program. For example, if *a happens*, do *b*; else if *c* happens, do *d*. In text-based programs, this can be accomplished with if–else statements, case statements, switch statements, and so on. Lab VIEW includes many different ways of making decisions such as select function, case structure, and formula node.

CASE STRUCTURE:

The case structure is Lab VIEW's method of executing conditional text, sort of like an "if–then–else" statement. It is located in the Structures sub palette of the Functions palette. The Case Structure has two or more sub diagrams, or cases; only one of them executes, depending on the value of the Boolean, numeric, or string value wired to the selector terminal. If a Boolean value is wired to the selector terminal, the structure has two cases, FALSE and TRUE. If a numeric or string data type is wired to the selector, the structure can have from zero to almost unlimited cases. Initially only two cases are available, but number of cases can be easily added. More than one value can be specified for a case, separated by commas. In addition, the user can always select a "Default" case that will execute if the value wired to the selector terminal doesn't match any of the other cases. When a case structure is first placed on the panel, the Case Structure appears in its Boolean form; it assumes numeric values as soon as a numeric data type is wired to its selector terminal. Case Structures can have multiple sub diagrams, but the user can see only one case at a time, sort of like a stacked deck of cards. Clicking on the decrement (left) or increment (right) arrow at the top of the structure displays the previous or next sub diagram, respectively. The user can also click on the display at the top of the structure for a pull-down menu listing all cases, and then pop-up on the structure border and select **Show Case**. If a floating-point number is wired to the selector, Lab VIEW rounds that number to the nearest integer value. Lab VIEW coerces negative numbers to 0 and reduces any value higher than the highest-numbered case to equal the number of that case. The selector terminal can be positioned anywhere along the left border. If the data type wired to the selector is changed from a numeric

to a Boolean, cases 0 to 1 change to FALSE and TRUE. If other cases exist (2 to n), Lab-VIEW does not discard them, in case the change in data type is accidental. However, these extra cases must be deleted before the structure can execute. For string data types wired to case selectors, the user should always specify the case values as strings between quotes. The only exception is the keyword **Default**, which should never be in quotes. [16-18],[20].

4.11 ARRAYS

An array consists of elements and dimensions. Elements are the data that make up the array. A dimension is the length, height, or depth of an array. An array can have one or more dimensions and as many as 230 elements per dimension, memory permitting. One can build arrays of numeric, Boolean, string, waveform, and cluster data types. One should consider using arrays when working with a collection of similar data or when one performs repetitive computations. Arrays are ideal for storing data collected from waveforms or data generated in loops, where each iteration of a loop produces one element of the array. One cannot create arrays of arrays. However, one can use a multidimensional array or create an array of clusters, where each cluster contains one or more arrays. To locate a particular element in an array requires one index per dimension. In Lab VIEW, indices let one navigate through an array and retrieve elements, rows,

Columns and pages from an array on the block diagram. The user create an array control or indicator on the front panel by placing an array shell on the front panel, and dragging a data object or element into the array shell, which can be a numeric, Boolean, string, or cluster control or indicator. The array shell automatically resizes to accommodate the new object, whether a Boolean control or a 3D graph. To create a multidimensional array on the front panel, one would right-click the index display and select *Add Dimension* from the shortcut menu. One also can resize the index display until one has as many dimensions as needed. To delete dimensions one at a time, the user would right-click the index display and select *Remove Dimension* from the shortcut menu. One also can resize the index display to delete dimensions. To display a particular element on the front panel, the user can either type the index number in the index display or use the arrows on the index display to navigate to that number.

Array functions can be used to create and manipulate arrays, such as in the following tasks:

• Extract individual data elements from an array.

• Insert, delete, or replace data elements in an array.

• Split arrays.

The *Index Array*, *Replace Array Subset*, *Insert into Array*, *Delete from Array*, and *Array Subset* functions automatically resize to match the dimensions of the input array that are wired. For example, if one wires a 1D array to one of these functions, the function shows a single index input. If one wires a 2D array to the same function, it shows two indexed inputs—one for the row and one for the column. One can access more than one element, or sub-array (row, column, or page) with these functions by using the positioning tool to manually resize the function. When one expands one of these functions, the function expands in increments determined by the dimension of the array wired to that function. If one wired a 1D array to one of these functions, the function expands by a single indexed input. If one wired a 2D array to the same function, the function expands by two indexed inputs—one for the row and one for the column. The indexed inputs that are wired determine the shape of the sub-array one wants to access or modify. For example, if the input to an *Index Array* function is a 2D array, and if only the *row* input is wired, a complete 1D row of the array is extracted. If only the *column* input is wired, a complete 1D column of the array is extracted. If both the *row* input and the *column* input are wired, a single element of the array is extracted. Each input group is independent and can access any portion of any dimension of the array.[19-20].

4.12. Clusters

Clusters group data elements of mixed types, such as a bundle of wires, as in a telephone cable, where each wire in the cable represents a different element of the cluster. A cluster is similar to a record or a structure in text-based programming languages. Bundling several data elements into a cluster eliminates wire clutter on the block diagram and reduces the number of connector pane terminals required by the corresponding sub-vi. The connector pane can have, at most, 28 terminals. If the front panel contains more than 28 controls and indicators required by a program, then some of them could be grouped into a cluster, which in turn can be assigned to a terminal on the connector pane. Although cluster and array elements are both ordered, one must

unbundle all cluster elements at once rather than index one element at a time. One also can use the *Unbundle by Name* function to access specific cluster elements. Clusters also differ from arrays in that they have a fixed size. Similar to an array, a cluster is either a control or an indicator. A cluster cannot contain a mixture of controls and indicators. Most clusters on the block diagram use a pink wire pattern and a data type terminal. Clusters of the numeric type, sometimes referred to as points, use a brown wire pattern and a data type terminal. One can wire brown numeric clusters to 'Numeric functions', such as 'Add' or 'Square Root', to perform the same operation simultaneously on all elements of the cluster. Cluster elements have a logical order unrelated to their position in the shell. The first object placed in the cluster is element 0; the second will be element 1, and so on. If an element is deleted, the order adjusts automatically. The cluster order determines the order in which the elements appear as terminals in the *Bundle* and *Unbundle* functions placed on the block diagram. One can view and modify the cluster order by right-clicking the cluster border and selecting *Reorder Controls in Cluster* from the shortcut menu. To connect two clusters, both clusters must have the same number of elements. Corresponding elements, determined by the cluster order, must have compatible data types. For example, if double-precision floating-point numeric data in one cluster corresponds in cluster order to a string in another cluster, the connecting wire on the block diagram will appear broken, and the VI will not run. If the numeric data are of different representations, Lab VIEW forces them to have the same representation. In addition, one can perform the following tasks:

• Extract individual data elements from a cluster.

• Add individual data elements to a cluster.

• Break a cluster out into its individual data elements.[16-20].

4.13 conclusions:

This chapter presents the importance of virtual instrumentation, features of Lab VIEW and present drawback of lab view. This chapter also presents the main parts of lab view, different types of structures viz. for loop, while loop, case structure, shift registers in detail. Finally this chapter presents the different types of arrays, and sclusters.

CHAPTER V

5.1 LAB VIEW implementation of a neuron:

5.1.1. Using sigmoid function:

The below figure shows the basic neuron model implementation in lab view environment. In this block diagram input array and weigh array is a 1-dimentional array. These two are combined with the dot product which gives the net input of the neuron. The net input of the neuron is then applied to sigmoid function, which gives the neuron response. Here the sigmoid function is a sub VI

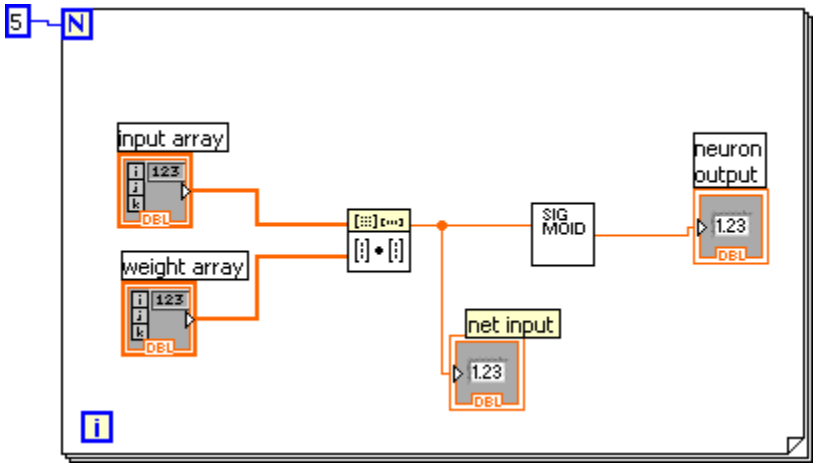

Fig5.1 block diagram implementation of a neuron

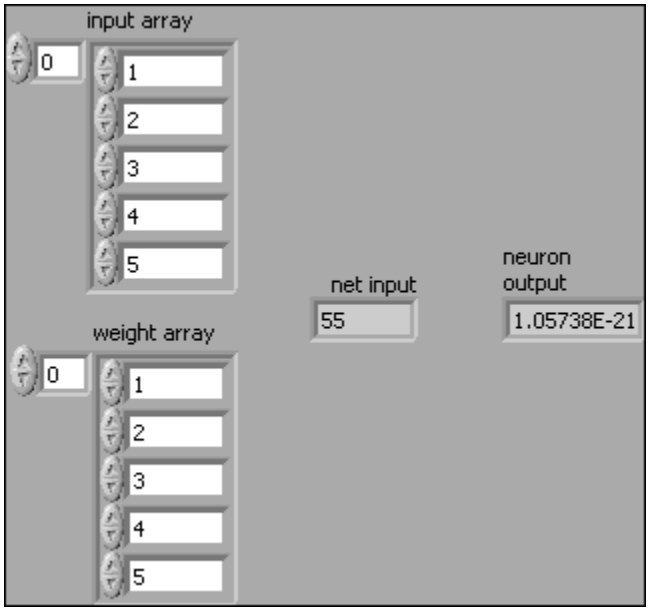

Fig5.2 front panel implementation of a neuron

The above figure5.2 shows the front panel implementation of a neuron using sigmoid function. Here, considering x_1,x_2,x_3,x_4,x_5 as inputs,w_1,w_2,w_3,w_4,w_5 as weight array, by using the following equation we find out the net input of the neuron.

$$X_i*W_{ij} = x_1w_1+x_2w_2+x_3w_3+x_4w_4+x_5w_5 \quad (1)$$

$$=1*1+2*2+3*3+4*4+5*5=55.$$

This is the net input of the neuron which is then applied to sigmoid function which gives the neuron output.

5.1.2. SIGMIOD FUNCTION:

It is defined as strictly increasing function that exhibits grateful balance between linear and non linear behavior. It is defined by eqn. **(**2)

$$F(u_i) = 1/\ 1+\exp(-u_i/\sigma) \quad (2)$$

Where σ is a slope parameter of the sigmoid function.[1],[2].

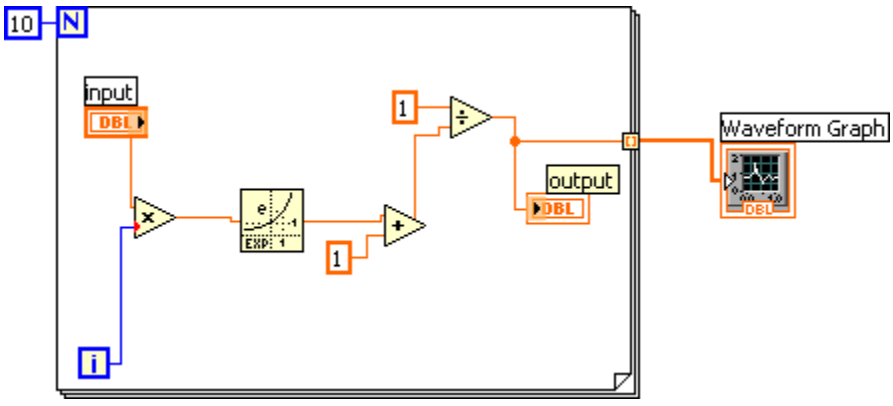

Fig 5.3 BD implementation of sigmoid function

The above block diagram shows the sigmoid function. The widely used activation function is sigmoid function because it is continuous and differentiable which is useful in back propagation algorithms.

The below diagram shows the front panel of sigmoid function, here, taking input values as 5 then the output becomes 2.8625.

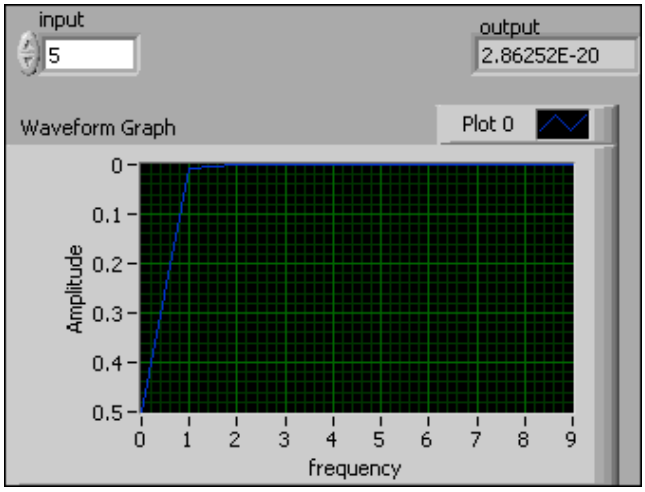

Fig 5.4 FF implementation of sigmoid function

5.1.3. Using Gaussian functions:

The below diagram shows the block diagram implementation of the neuron model using Gaussian function. Here the input nodes and weigh vector has taken as numeric values, these two multiplied with dot product which gives the net weight of the neuron. The net weight of the neuron is applied to Gaussian function which gives the neuron output.

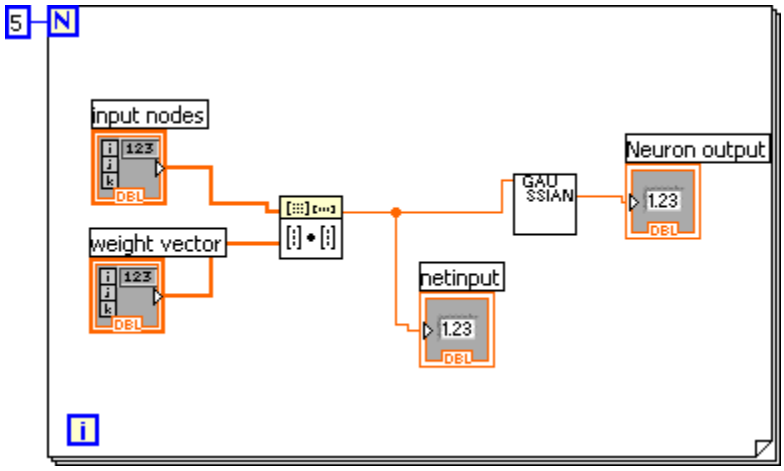

Fig5.5 BD implementation of a neuron using Gaussian function

The below diagram shows the front panel of the neuron model ,here considering input nodes as 1,2,3 and weight vector as 1 ,2,3.

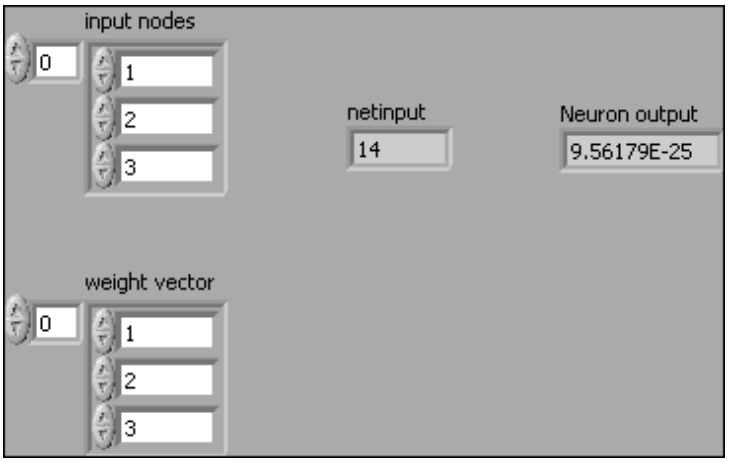

Fig 5.6 FF implementation of a neuron using Gaussian function.

5.1.4. Gaussian function

The Gaussian function can be represented by the following equation

$$F(u_i) = \exp(-u_i/\sigma^2) \qquad (3)$$

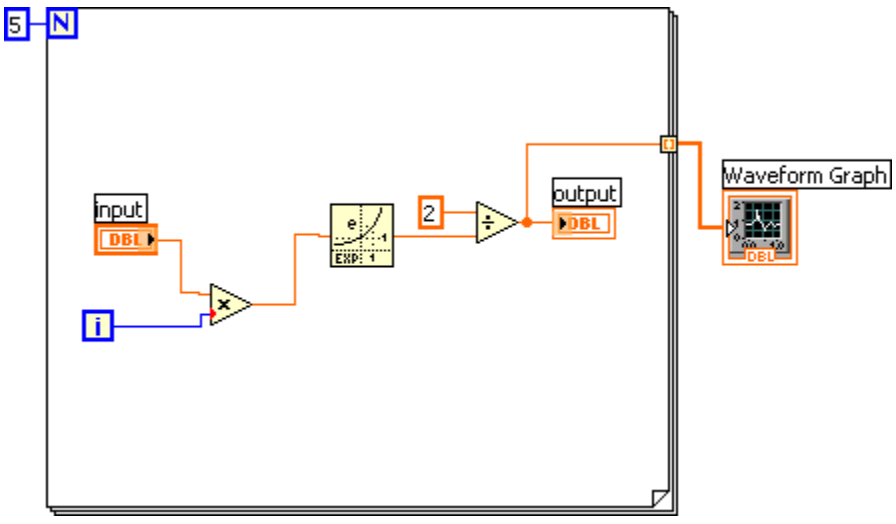

Fig 5.7 BD implementation of a Gaussian function

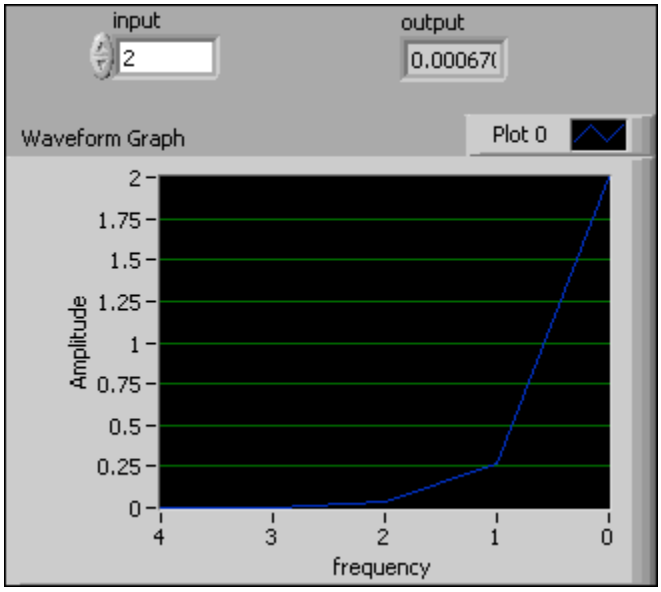

Fig 5.8 FF implementation of a Gaussian function

5.1.5. Inputs to an ANN

The inputs to any ANN should be assigned numeric values. Inputs to an ANN are also called patterns. Each pattern could be a single numeric value or vector. The number of features of input data is determined by the number of components in this input vector.

5.1.6. Weight Vector of an ANN

The weights are the connection strengths between neurons in adjacent layers. in the fig 1 Wi1,wi2,wi3,wi4,wi5 are the weights associated with each of the five connections between the inputs to a neuron and the neuron.

5.1.7. Net Input to a neuron

The net input to a neuron is the dot product of the weight vector and the input data vector.

5.2 Multi input –single output Neuron response:

5.2.1. Using Sigmoid function

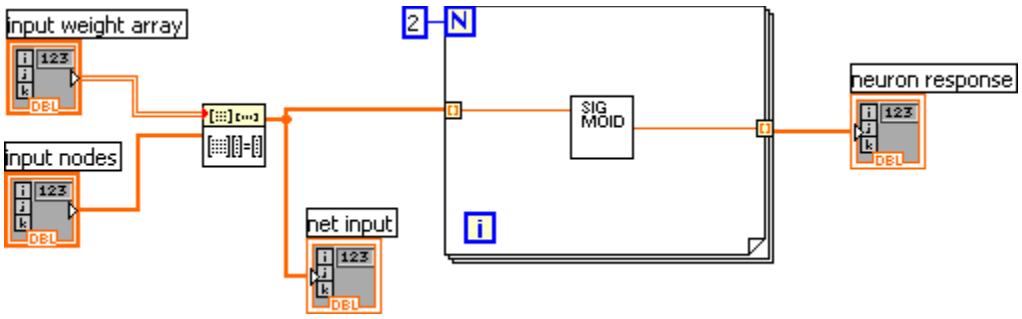

Fig 5.9 BD implementation of a neuron for multi input-single output

In the above block diagram the input node and the weight vector is applied to a linear basis input which gives the net input of the neuron. The net input of the neuron is applied to sigmoid function which gives the response of the neuron. Here the main advantage of using linear basis input is it is useful for finding the response of multi input neuron.

Assuming U(X, W) as linear basis having two inputs (1, 1) and Weights (0.6, 0.6), (0.8, 0.8) respectively then the output of summation of signal is given by the following equation

$U_i(W,X)=\Sigma_{j=1}(W_{ij}x_j)$,where j=1,2…….m; i=1,2……..n

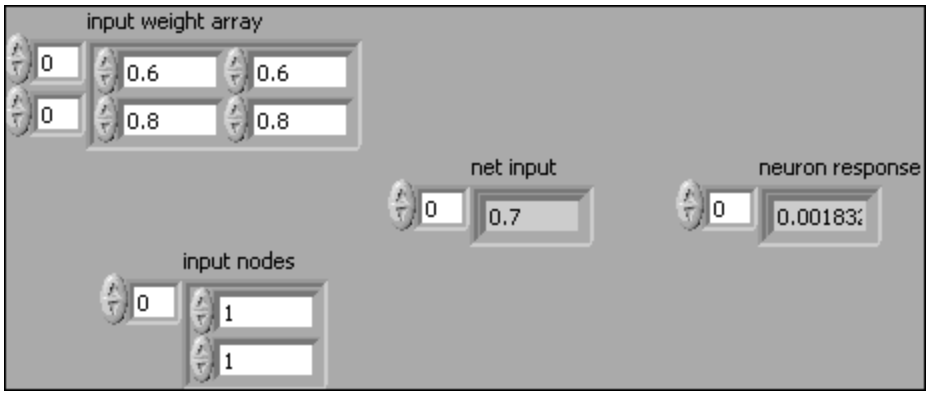

Fig 5.10 FF implementation of a multi input single output of a neuron

In the earlier study of neuron we are taken dot product function that is only for single input single output. Suppose if we want to increase inputs, better to use linear basis input as a net input to the neuron. For that purpose using linear equation function instead of dot product function which gives multi input response of a neuron.

5.2.3. Using Gaussian function

The below block diagram shows the block diagram implementation of a neuron for multi input-single output using Gaussian function.

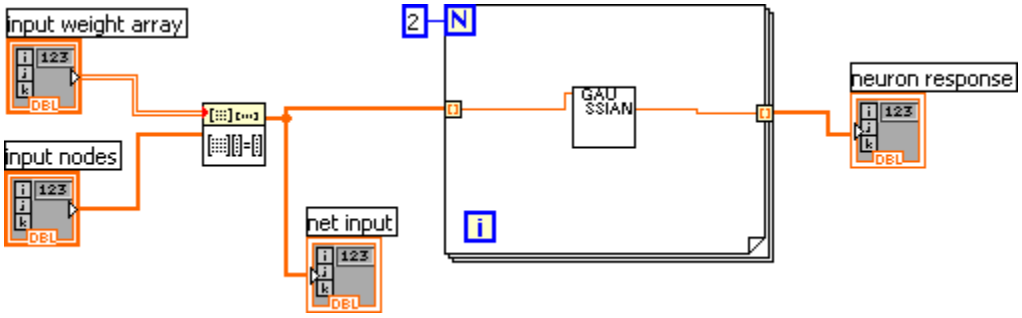

Fig 5.11BD implementation of multi input- single output of a neuron

The below front fanel shows the block diagram implementation of a neuron for multi input-single output using Gaussian function. Here I am taking the values of input node as (1, 1) and input weight array as (0.6, 0.6), (0.8, 0.8) respectively.

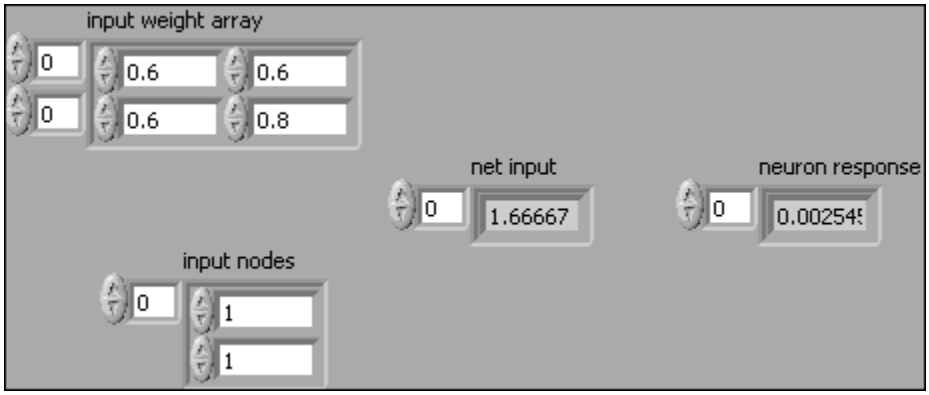

Fig 5.12 FF implementation of a multi input-single output of a neuron

5.3New weight of the neuron using different types of inputs:

(i) Sigmoid function

The below block diagram represents the determination of new weight of the neuron using different types of activation functions

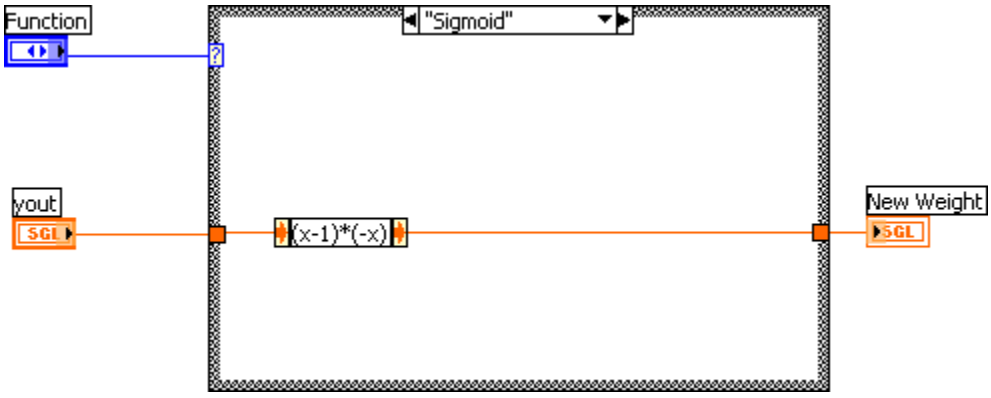

Fig 5.13 BD implementation for finding new weight of the neuron using sigmoid function

In this block diagram, take case structure for selecting different types of activation functions viz. sigmoid, linear, tangential hyperbola etc. In case structure it will takes conditions through condition terminal. In this conditional terminal connect tab control. By using tab control it is possible to add different types of functions. Define each activation function separately by using expression node .this will give the new weight of the neuron.

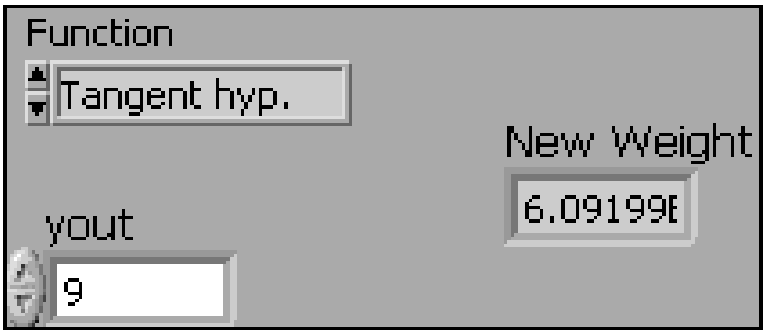

Fig 5.14FF implementation for finding new weight of the neuron using sigmoid function

The above diagram shows the front panel implementation of the new weight of the neuron using sigmoid function.

(ii) Tangential hyperbola

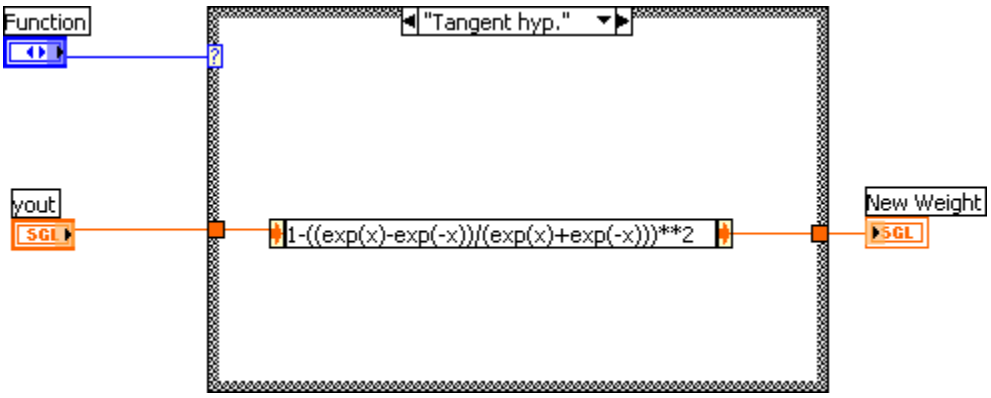

Fig 5.15. BD implementation for finding new weight of the neuron using sigmoid function

Fig 5.16 BD implementation for finding new weight of the neuron using sigmoid function

(iii) Linear input

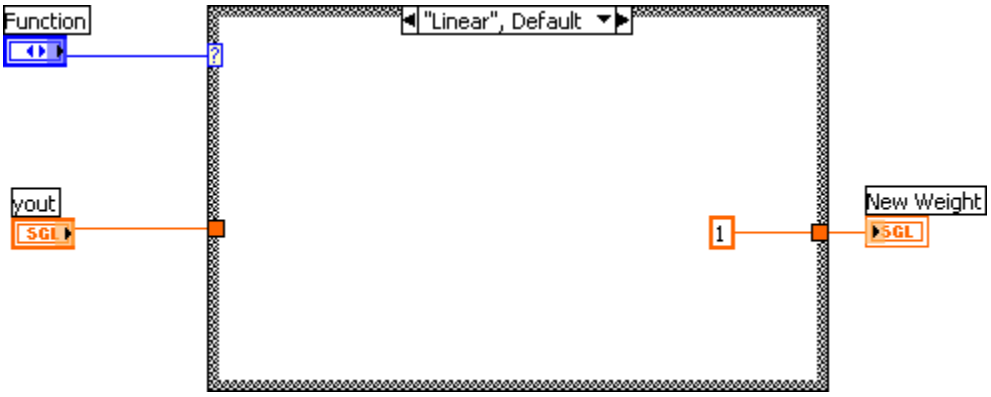

Fig 5.17 BD implementation for finding new weight of the neuron using sigmoid function

Fig 5.18 BD implementation for finding new weight of the neuron using sigmoid function

5.4 Neuron response using different types of inputs:

1. Sigmoid function

The below block diagram shows the neuron output response using sigmoid function. In this block diagram I am taking case structure, the main advantage of using case structure is it is possible to select different types of inputs through selector line. By using expression node, different types of functions will be defined like sigmoid, tangential hyperbola, linear input etc…

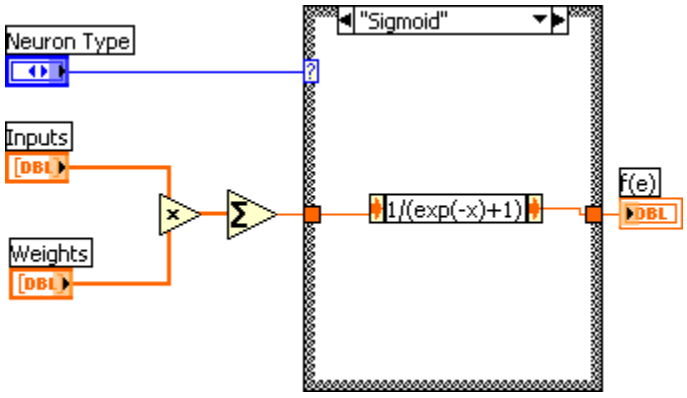

Fig 5.19 BD implementation of neuron output response using sigmoid function

The below diagram shows the front panel implementation of a neuron output using sigmoid function. Here considering input values as (0.6,0.6,0.8,0.8,1),weights (1,2,3,4,5) respectively. By using these values the net output of the neuron has been calculated.

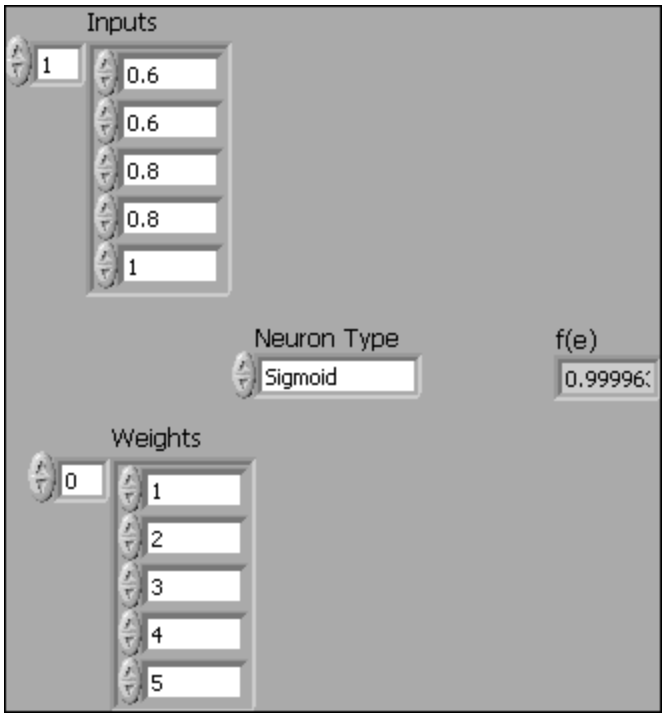

Fig 5.20 FF implementation of neuron output response using sigmoid function

(ii)Tangential hyperbola

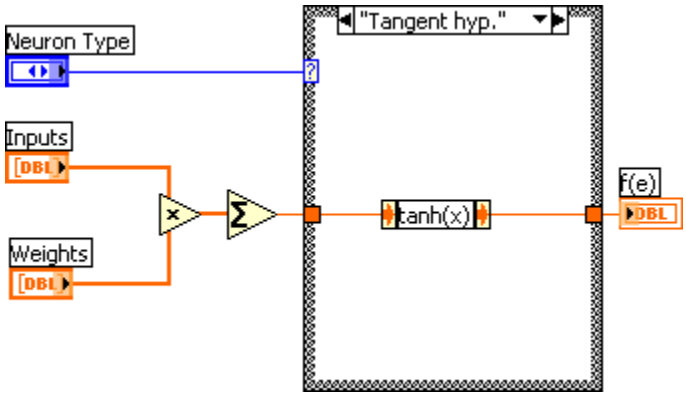

Fig 5.21 BD implementation of neuron output response using tangential function

The below front panel shows the neuron output response using Tangential hyperbola function as activation function. Here I am considering inputs as (0.1, 0.2, 0.3, 0.4, 0.5) weights as (0.2, 0.3, 0.4, 0.5) respectively which gives the neuron output as 0.3799.

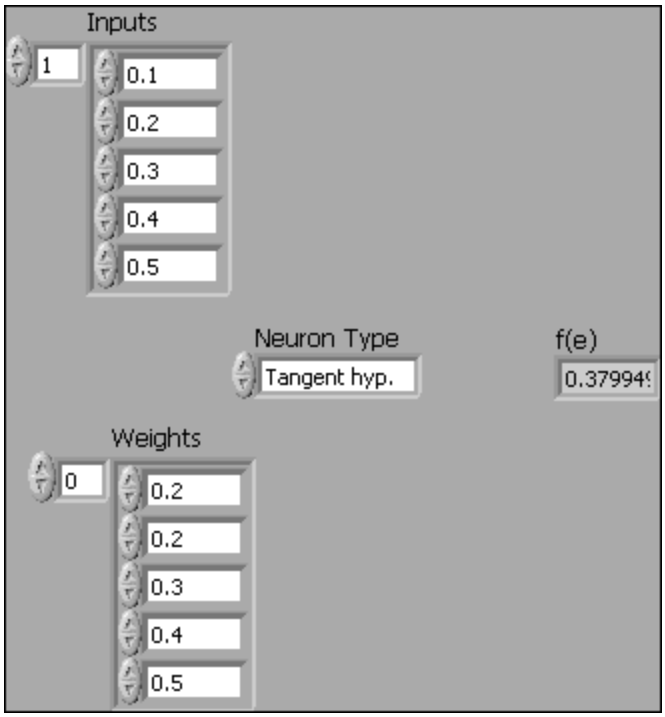

Fig 5.22. FF implementation of neuron output response using tangential function

(iii)Linear input

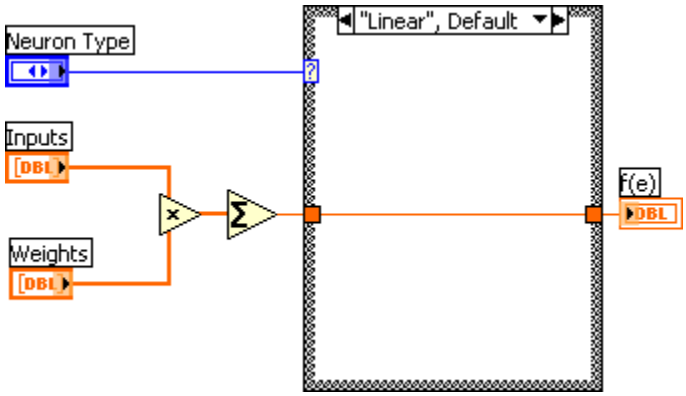

Fig 5.23 BD implementation of neuron output response using Linear input

The below front panel shows the neuron output response using linear input as activation function. Here I am considering input values as (0.1, 0.2, 0.3, 0.4, 0.5), weights as (0.2, 0.3, 0.4, 0.5) respectively which gives the neuron output as 0.4.

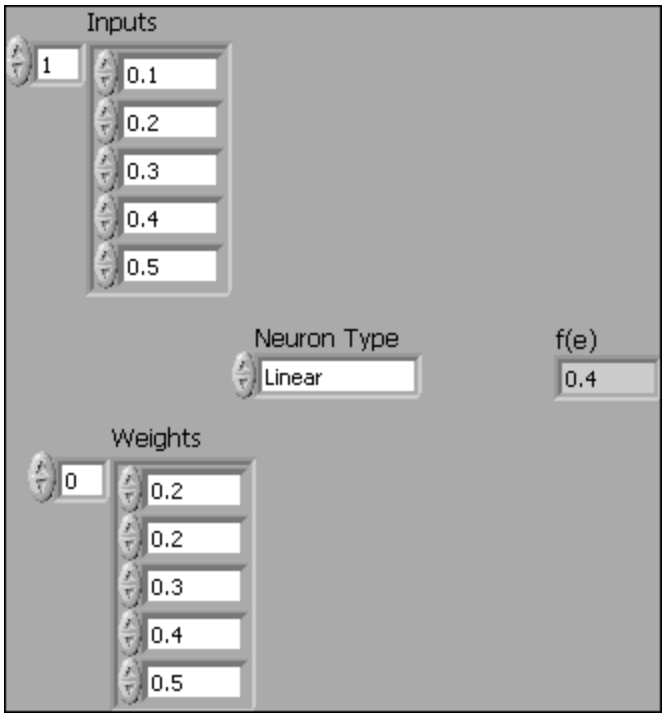

Fig 5.24 FF implementation of neuron output response using linear input

5.5. Gaussian function implementation using radial basis input

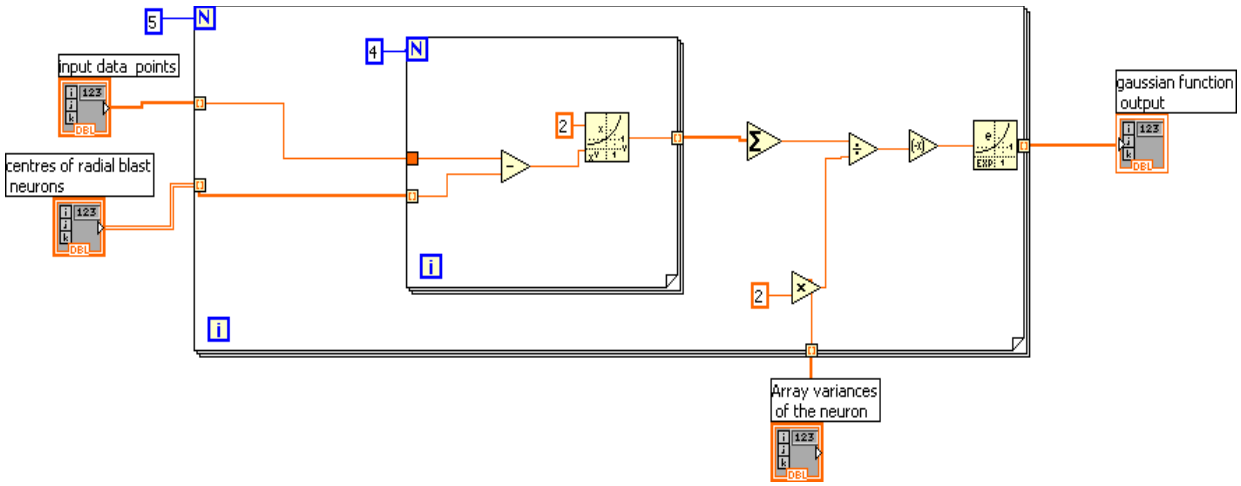

Fig 5.25 Gaussian function implementation using radial basis input

Radial Basis Input: This is a nonlinear basis function in which the net distance (Euclidean norm) between the input patterns and the weights is defined as

$$(W, X) = \sqrt{\sum_{j=1}^{n} (x_j - w_{ij})^2} \qquad j = 1,2,\ldots m;\ i = 1,2,\ldots n;$$

This is a hyper spherical function.

Where, x - input feature vector,

x_i– centers of radial basis activation functions of the neuron,

σ - Radius of the radial basis function

‖ X-xi ‖ This is the Euclidean norm,

Fig(6) and (7) show the Gaussian model in lab view representing mathematical model given by eqn. (3).

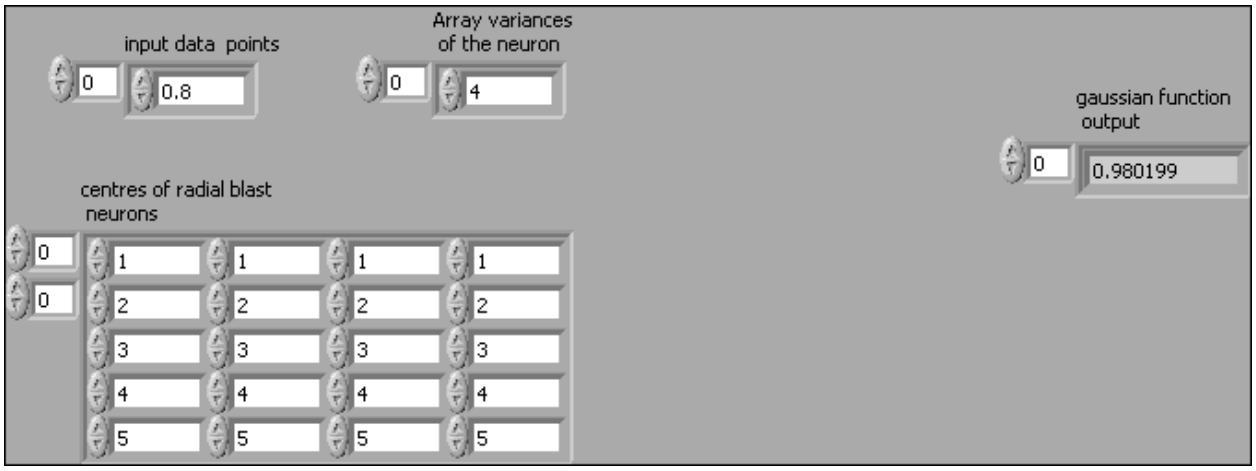

Fig 5.26 Gaussian function implementation using radial basis input

The above figure shows how the Gaussian function output is computed using neurons on Lab view platform. The input layer is a one-dimensional array whereas the weight matrix is a 5×4 matrix. There are 2 "for loops" involved here. The first one is to sequence the 5 centers, one after the other. It is noted that the indexing is enabled for the "centers of radial basis neurons whereas it is disabled for the "input data points". The second "for loop" is set-up for four iterations. '4' corresponds to the dimensions of the data points and the dimensions of the centers as mentioned earlier. Each time that this loop executes, a corresponding dimension of the data

point and the center are subtracted from each other, and resulting value squared. In this way one can obtain the difference between two vectors by adding up all of the squared distances and then applying the square root function over the sum. This loop thus calculates the Euclidean distance between these two vectors. This will give the Gaussian function output of the neuron.

5.6. Implementation of Learning Algorithms using Lab VIEW

5.6.1. Supervised Learning Algorithms:

Supervised ANN

This is the most popular type of ANN in which a "teacher" provides information to the network to drive it to emulate the desired function. Suppose a set of input patterns (input data vectors) are applied to the network, then the output response of the ANN is then compared with the desired response from the "teacher". The teacher would inform the network whether the output decision is correct or incorrect. Also, one could define an error criterion that would then be a basis to update the weights of the ANN so that the network will be trained with successive input patterns. This, in general is how a supervised network would be achieved.

Consider the example of pattern classification. Here, a sequence of input patterns is applied to the network. The desired output response is assigned as the "teacher". If the network gives a correct response to a particular input then no change is made. But, if the response is incorrect, then the teacher provides an error difference between the actual response and the desired response, which is used to adjust the weights.

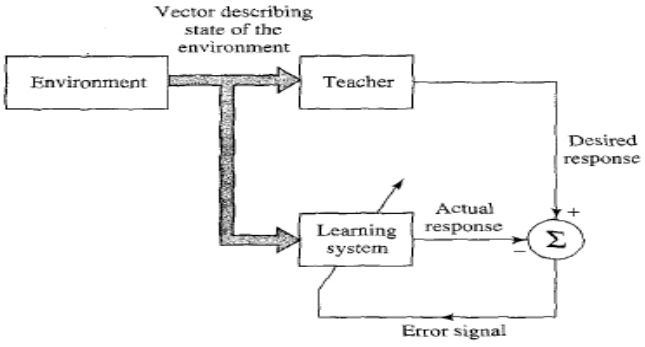

Fig 5.27. Block diagram of learning with a teacher

5.6.2. Back propagation Algorithm:

The back propagation algorithm is the most widely used algorithm for the supervised learning of an ANN. Basically in this algorithm there is a forward pass and a backward pass through the layers of the network. The product of the weights and the input data vector, followed by the sum and evaluation take place in the hidden layer. The dot product of the outputs of the hidden nodes and the output layer weights are applied as input to the output layer. The forward pass ends here.

The backward pass begins by calculating the error at the output layer, where h is the squared error difference between the outputs and the desired values. Let there be k output neurons.

O_K- output neuron k

Y_K- desired value of output neuron k

W_{ij} -Weight matrix between the i and j layers. The matrix dimensions depend on the number of features (dimensions) of the input vector and the number of hidden neurons

Wjk- Weight matrix between j and k layers.

Weight matrix $\varepsilon_k = O_K (1-O_k) (Y_k-O_K)$……. (1)

This error function is used to update weights between output layer and next to last layer.Wjk weight vector between j and k layers is updated by the equation

$$W^{new}_{jk}=W^{old}_{jk} +\eta O\varepsilon k \text{ ……. (2)}$$

η is the learning parameter whose value lies between (0, 1).Its value is assigned based on trial and error methods. The next step in the backward pass would be to calculate the error function of the hidden layer. The weight of the connection between I and j layers are given by

$$W^{new}_{ij}=W^{old}_{ij} +\eta O\varepsilon k \text{ ……. (3)}$$

Where ε_j is given by $\varepsilon_j = Oj\,(1-Oj)\,\sum_k W_{jk}\,\varepsilon_k$ where εj is the error function for each hidden neuron and εk is the error function of the output neuron.

5.6.3. Lab VIEW implementation of Back propagation algorithm

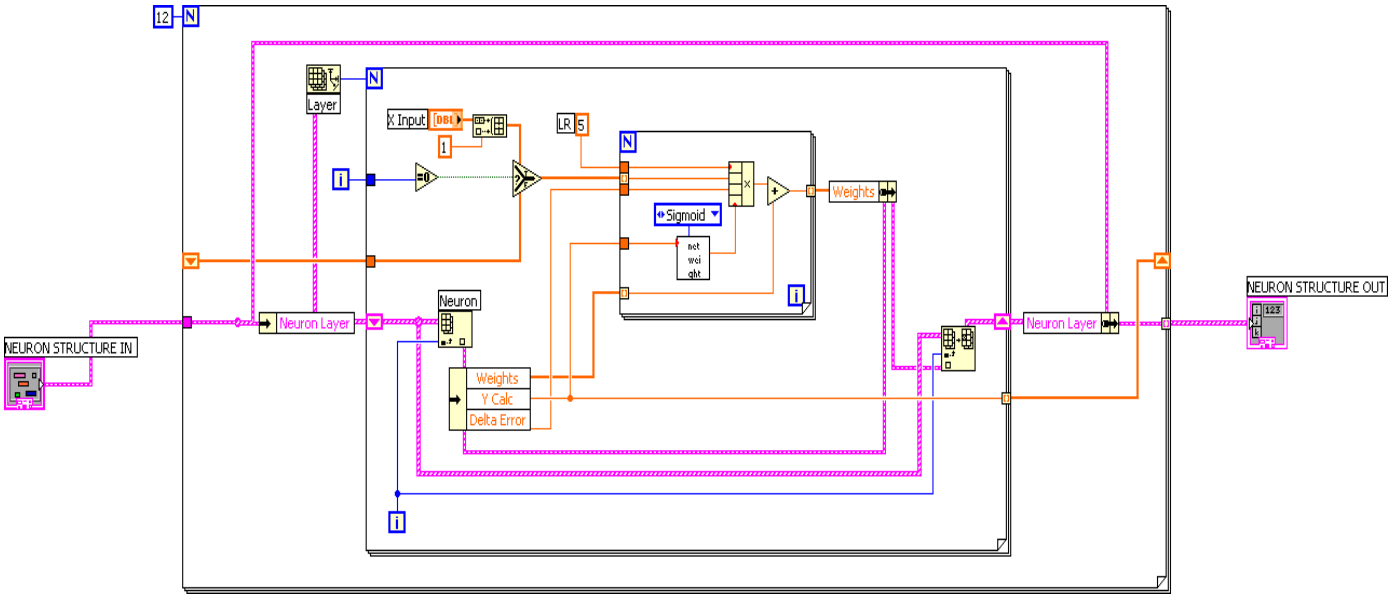

Fig 5.28.BD implementation of back propagation algorithm

The above fig shows the block diagram implementation of back propagation algorithm. Back propagation algorithm is under the class of multilayer perceptrons. Error back propagation learning consists of two passes through the different layers of the network. A forward pass and a backward pass. In the forward pass, an activity pattern (input vector) is applied to the sensory nodes of the network, and its effects propagate through the network layer by layer. Finally, a set of outputs is produced as the actual response of the network. During the forward pass the synaptic weights of the networks are all fixed. During the backward pass, the synaptic weights are all adjusted in accordance with an error – correction rule. Specifically, the actual response of the network is subtracted from a desired response to produce an error signal. This error signal is then propagated backward through the network against the direction of synaptic connections. The synaptic weights are adjusted to make the actual response of the network move closer to the

desired response in a statically sense. The error back propagation algorithm is also referred as back propagation algorithm.

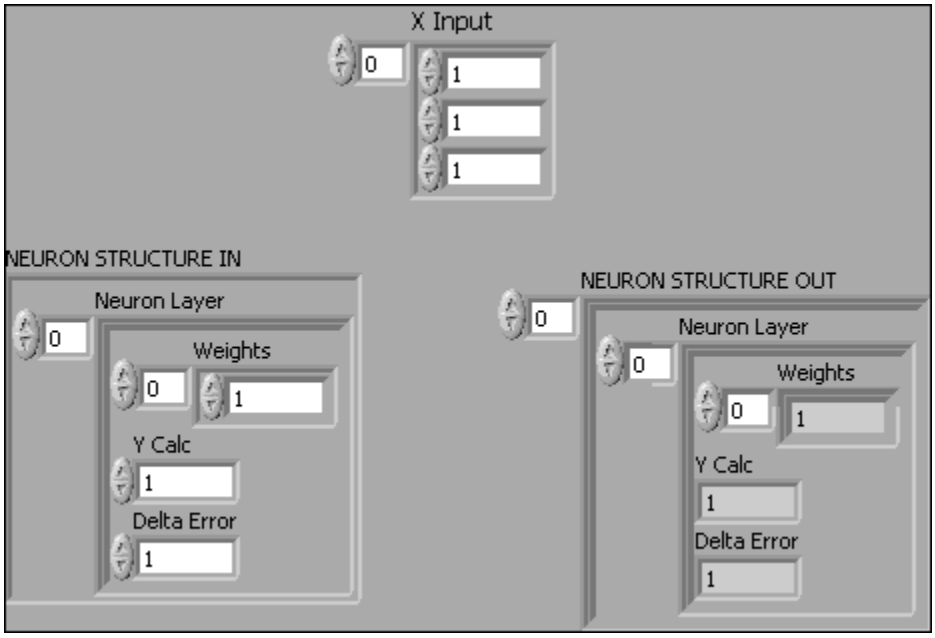

Fig 5.29. FF implementation of back propagation algorithm

5.7. Unsupervised Learning Algorithms:

5.7.1. Unsupervised ANN:

In Unsupervised learning, there is no teacher to provide any feedback information .There is no feedback from the environment to say what the outputs should be or whether they are correct. The network must discover for itself patterns, features, regularities, correlations, or categories in the input data and code for them in the output.

The unsupervised networks are also called self-organizing networks or rather, *competitive learning* networks. These competitive learning networks do not have a "teacher's" guidance. The basis of unsupervised networks is clustering techniques. They help in grouping similar patterns, where each cluster has patterns closer together. Some basic unsupervised models are the Self-Organized Feature Mapping ANN (SOM), the Vector Quantization ANN (VQ), and the RBF ANN. The basic idea in all of these networks is that the hidden layer of neurons should capture the statistical features of the input data. The hidden neurons have the ability to extract the

features of the data set. In most cases, the hidden neurons compete with each other to determine which one of them is closest to the input data, and the "winning neuron" is updated, which means the winning neuron is moved closer to the input data point which is a vector in multidimensional space. Hence, this type of network is also called "winner-takes-all" network

5.7.2. Unsupervised clustering algorithms:

Clustering techniques come under the concept of competitive learning. Clustering is one of the most important unsupervised training techniques. Basically it is a process by which objects that are closely related to each other are grouped. Similar objects form groups and they are dissimilar from other groups. For example, consider a radial basis network; it uses the Euclidean distance technique to find out the distance between the input m-dimensional vector and the 'prototype' radial basis centers. 'Prototype' center refers to the ideal center (mean value of a cluster) that needs to be found using learning algorithms. Each input point is an m-dimensional vector that lies in an m-dimensional feature space. The radial basis 'centers' are also m-dimensional vectors. Suppose initially centers are assumed for k Radial Basis Functions. Then, the center with the minimum distance from the input vectors is considered the 'winner' and is updated (moved closer to the input vector). The idea is to move the 'prototype' center nearer to the input vectors so that they form a group. During successive iterations, the winning vector is updated and moved closer to the input data. After a large number of iterations (epochs) and after comparing the input data points many times with the centers, one can find that the cluster centers have moved to their approximate means. Suppose that one has assumed 10 clusters; each cluster has a corresponding center (mean) point; then, after many iterations of the learning algorithm, one would be able to obtain 10 updated cluster center (mean) values. Thus 10 different clouds or clusters (either overlapping or non-overlapping) are formed. Now, new input feature vectors can be applied to the "learned network" and then one can find the closest cluster center to each input vector. Thus, one can classify future data points entering the network.

5.7.3. K-Means Clustering Algorithm

This is a well-known algorithm to determine the appropriate centers of the prototype hidden neurons in radial basis function networks. Suppose there are m centers, then

Step 1: Pick ‘m’ random points from the input data

Step 2: Select vector randomly from the input space. Let it be ‘x’

Step 3: Suppose k varies from 1 to m then let $t_k(n)$ be the center of the K^{th} RBF at time step ‘n’. The minimum Euclidean distance between and $t_k(n)$ is evaluated for all k varying from 1 to m. Let K(x) denotes the index of the winning center.

$$K(x) = \arg \{\min (\| x(n) - t_k(n) \|)\}$$

Where k=1, 2… m and $\| x(n) - t_k(n) \|$ is the Euclidean norm

Step 4: The winning neuron center is updated as follows:

$t_k(n+1) = t_k(n)+\eta[x(n)- t_k(n)]$ k is the index of the winning center

$t_k(n+1)= t_k(n)$, otherwise

Step 5: The time step is incremented to the next iteration and then step 1 to step 4 are repeated again until the appropriate centers for the m clusters have been obtained.

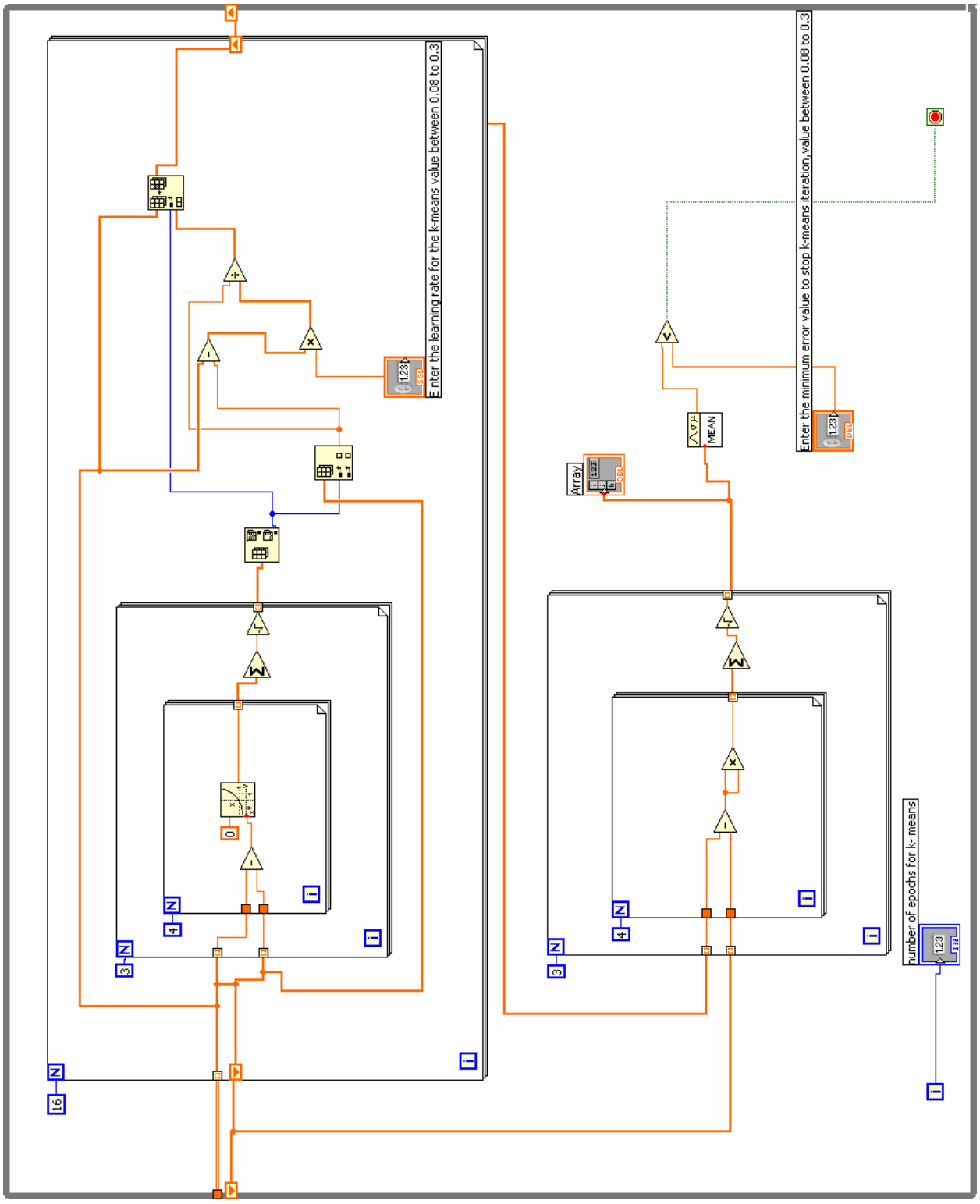

Fig 5.30 BD implementation of k-means clustering algorithm

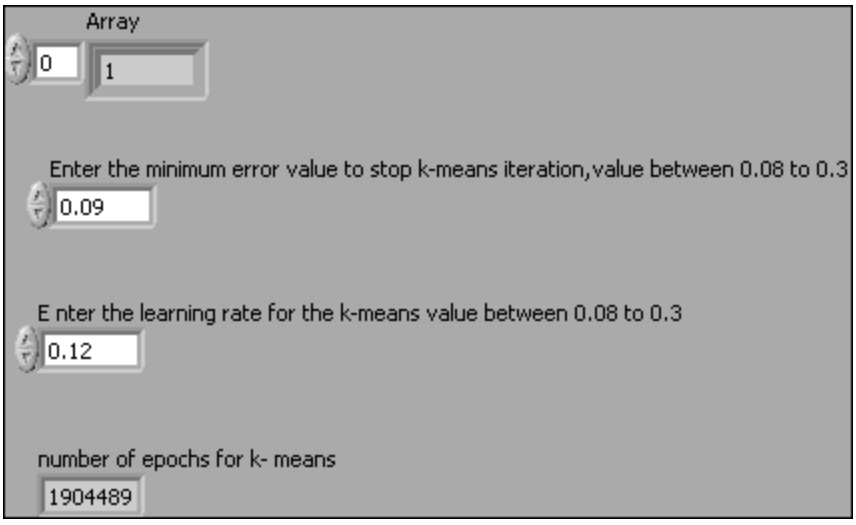

Fig 5.31 FF implementation

The block diagram shown in figure executes the K-means clustering algorithm the program has two parts. The first part implements the SOM algorithm. A "while loop" structure with a stopping criterion has been used here. The loop executes until the stopping criterion is achieved. The stopping criterion would generally be a pre-set level for the error. Once the error reaches the pre-set value, training stops. The 'for loop' with 16 iterations correspond to 16 centers obtained from the Self-organized mapping technique. The idea here is to pick 3 ideal prototype centers from those 16 possible cluster centers using the K-Means algorithm.

5.7.4 SOM ALGORITHM

The self-organizing feature mapping algorithm is an extension of the K-Means algorithm. The major difference between the two is that, in K-Means, as mentioned earlier, only the winning neuron is updated, and the center of the winning neuron is moved nearer to the input data point presented each time. But in the case of SOM, the winning neuron along with its neighbors are all updated and moved closer to the input data point during successive iterations. The main idea of SOM's is to convert an n-dimensional input space into 1 or 2 dimensional output space with each output neuron having a distinct center from the other. The SOM algorithm [1] follows.

Step 1: Randomly choose m centers from the input vectors. Assume that are m neurons in the hidden layer.

Step 2: Select input vectors, one at each time step 'n' from the input space.

Step 3: Calculate the minimum Euclidean distance between each input vector and the center for each of the hidden centers and the winner is found as before using the k-means algorithm.

$K(x) = \arg \{\min (\| x(n) - w_j \|)\}$

Where j = 1, 2…….m and k(x) refers to the index of the winning neuron

Step 4: Update the weights as follows:

$$w_j(n+1) = w_j(n) + \eta * h_{j,k(x)}(n) * [x(n) - w_j(n)]$$

Where k(x) refers to the index of the winning neuron; $h_{j, k(x)}$ is the neighborhood function in the output space defined by

$$h_{j,k(x)} = \exp\left(\frac{-\| r_k - r_j \|}{2 * \sigma^2(n)}\right)$$

Where r_k is the coordinates of the winning neuron in the 2 dimensional output feature space and refer to the coordinates of all the neurons in the 2 dimensional feature output space where j = 1, 2…….m. Also $\sigma^2(n)$ refers to the variance of the Euclidean distances between the winning neuron and all the other neurons in the output space

Step 5: Repeat steps 1 to 4 until it is judged that the output space has appropriate weight vectors, which are statistically independent of each other and that the weight vectors have extracted exclusive statistical features of the input data (or input space).

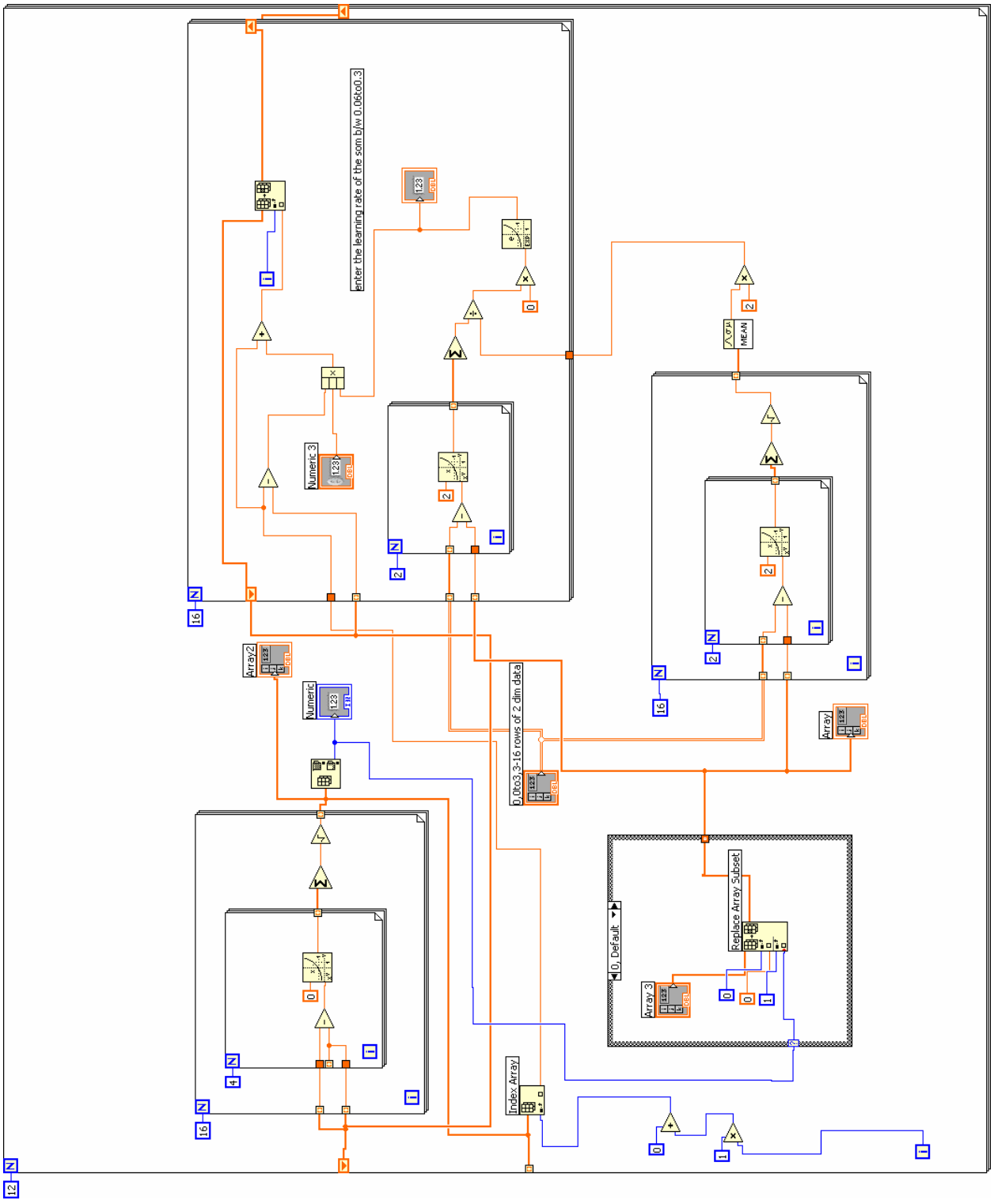

Fig5.32 Block diagram implementation of SOM algorithm

Figure 5.32 shows the block diagram which implements the SOM algorithm. By using the SOM algorithm the centers are updated during successive iterations. In order to map the 4 dimensional data into the 2 dimensional maps, a "case structure" is used as shown in figure 5.32. Case structures require input conditions, based on which a case is selected. In this problem the condition for case structure is the value of the index of the winning neuron. Since there are 16 centers the index of the winning neuron may lie between (0, 15). So four cases are chosen based on the range in which the index of the winning neuron falls. If the index of the winning neuron is in the range between (0, 4) then first case is selected. If the range is between (5, 7) then the second case is selected. The range of values for the other two cases are (8, 11) and (12, 15).

5.7.8. Implementation of Feed forward neural network using Logic gates:

A combined logic gate will be constructed to develop a feed forward neural network to analyze the input output truth-values. National Instrument Lab VIEW is employed to define the logic gate. Figure 7 depicts the front panel for interaction. And the graphical programming diagram block is shown in Figure 8. As A, B and C buttons are pressed; their status will be changed. That is, truth value will be toggled from 1 to 0 or vise visa. Each status will bring different byte-value for button Q. With this system, eight patterns truth-value can be used as the dataset for training, validating and testing the feed forward neural network. Table 1 is the truth-value is generated by the system.

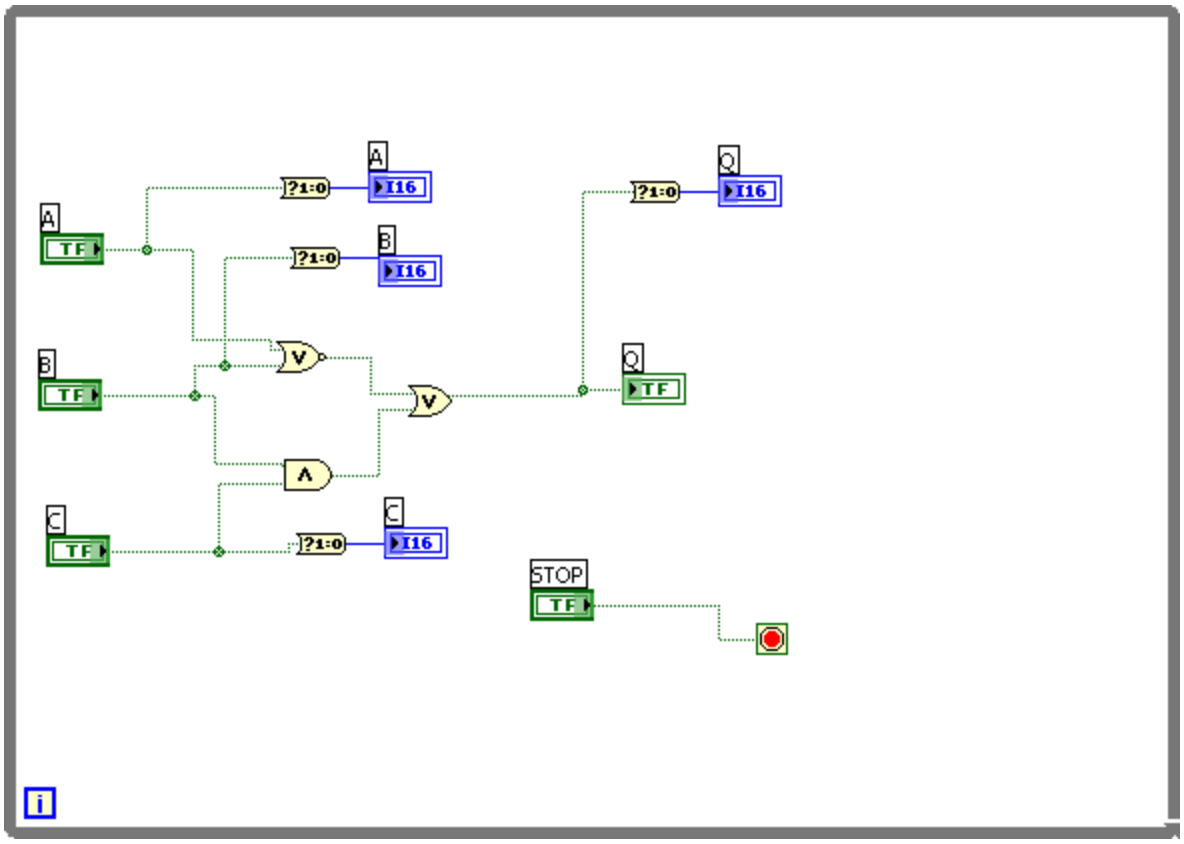

Fig5.33 Block diagram implementation of Feed forward NN

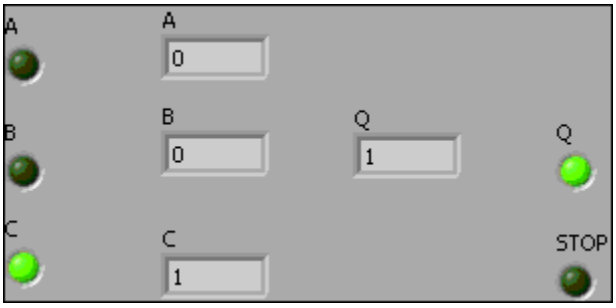

Fig5.34 FF implementation of Feed forward NN

Truth table of combined logic

A	0	0	0	1	1	0	1
B	0	1	0	0	1	1	0
C	0	0	1	0	0	1	1
Q	1	0	1	0	0	1	0

The above truth table shows the different types of input how the output of feed forward ANN is changing.

Conclusions:

In this chapter fusion of neuron with lab VIEW is explained in detail. Neuron response for single input –single output, multi input-single out function with different types of activation functions like sigmoid and Gaussian function is explained. Also this chapter explains new weight of the neuron using different activation functions like sigmoid, tangential hyperbola, linear function is implemented in Lab VIEW environment.

In this chapter also explains different training algorithms viz.supervised and unsupervised algorithms are implemented in Lab VIEW. In supervised learning algorithms, back propagation algorithm is implemented. In unsupervised learning K-means clustering algorithm, SOM algorithm, feed forward ANN using logic gates implemented

Conclusions

Neural Networks are the approximate emulation of biological neurons. It has the ability to extract the information quantitatively from the vague and uncertainty data. It has therefore been used to study the all the engineering problems. Lab view is a virtual graphical icon based protocol to study and analyze the process. Neural networks and lab view have some similar features of parallel processing. How ever the lab view does not have the neural network standard icon and analysis ability to process and analyze the neural net based model in its standard library. This thesis therefore have emphasize the development of neural network and stored in the standard library of lab view to enable the lab view protocol for further investigations.

Chapter III gives the mathematical model representation of neural networks and also electrical representation of neural networks. Chapter III also explains the characteristics of neural networks and different types of neural network architectures like single layer, multilayer networks etc. Chapter IV gives the analysis of Lab view graphical programming and also explains the need of different types clusters and arrays and the need of for loop and while loop. Chapter V gives the implementation of neuron model for different types of inputs viz. Single input- single-out functions, multi input- single output function using different types of activation functions. It also explains the different types of learning algorithms viz. supervised learning as well as unsupervised learning algorithms. In supervised learning algorithms error back propagation algorithm, in unsupervised learning, K-means clustering algorithm, SOM algorithm is implemented in lab view. Also this chapter gives the implementation of feed forward ANN using logic gates in lab view environment.

Future scope of work:

This thesis explains different types of architectures viz. single input- single output neuron, multi input- single out put neuron, different types of activation functions viz. sigmoid, tangential hyperbola, linear function, Gaussian function respectively. Also in this thesis explains different types of learning algorithms viz. supervised and unsupervised learning algorithms. By using the above developed learning algorithms, it is possible to develop plant system modeling, pattern recognition techniques, image compression tasks.

Bibliography

[1]. Haykin, S., 1999. "Neural networks*"* (2nd ed.), Prentice-Hall, Upper Saddle River, NJ.

[2]. Michael A. Arbib "The hand book of brain theory and neural networks", MIT press London, 2002.

[3]. N.P.Padhy, 2005.Artificial Intelligence and intelligent systems, oxford publications, India

[4]. Jeffheaton, 2005. Programming Neural networks in JAVA, McGraw-Hill publications, USA

[5]. Nikola K.Kasabov, 1998.Foundations of neural networks, Fuzzy systems, and knowledge engineering, MIT press

[6]. Alexander I.Galushkin 2007. Neural networks theory, Springer publications Germany.

[7]. Madan M.gupta, Dandinah.rao, "Neuro-Control systems", IEEE press, NY, 1994.

[8]. Chin-Teng Lin, C.S.Georhe LEE, Neural fuzzy systems: A neuro fuzzy synergism to intelligent systems, prentice hall ptr newjersy 1996.

[9]. Hagan, Demuth, Beale, Neural network design, Thomson Asia pvt ltd, Singapore, 2002.

[10]. Massimo Grattarola, Giuseppe Massobrio, "Bioelectronics handbook mosfets, biosensors, and neurons", McGraw-Hill publications, USA, 1998.

[11]. Michael A.Arbib "The hand book of brain theory and neural networks", MIT press London, 2002.

[12]. Anil K.Jain,K.M. Mohiudin, "Artificial Neural networks: A tutorial IEEE transaction on neural networks,1996.

[13]. K.krishna Kumar, "Back propagation algorithm for a generalized neural network structure", IEEE transaction on neural networks vol 2 no 4 July 2003

[14]. Lakshmi, C.Jain, N.M. martin, "Fusion of neural networks, Fuzzy systems, and genetic algorithms: Industrial applications, CRC press, 1998, USA

[15].Terrencel.Fine,"Feed forward neural network methodology", Springer publications, newyork, 1999.

[16]. The Self-Organizing map, Teuvo Kohonen .

[17].S. Sumathi, P.surekha, “Lab view based advanced instrumentation systems”, springer publications, India, 2007

[18].Rick bitter, Taqi Mohiuddin, Matt Nawrocki, “Lab view advanced programming techniques”, second edition, CRC press, Newyork, 2007.

[19]. John Conway and Steve watts, “A software engineering approach to lab view”, Prentice hall PTR, Newyork 2003.

[20]. Robert bishop, “Lab VIEW 7 express student edition”,, PTR pub,Newyork,dec,2003.

[22]. “Lab VIEW basics I: Introduction course manual”, National instruments India, 2008.

[23]. “Lab VIEW basics II: Introduction course manual”, National instruments India, 2008.

[24]. www.ni.com/india.

BIO- DATA

Name: RAMANJANEYULU GALI

Academic qualification:

- Completed 10 th from Z.P.H.S, kandlagunta, Guntur.
- Completed 12 th from Vivekananda junior college from Vijayawada
- Completed B.E from Andhra University

List of publications:

1. Presented IEEE paper on " Neuron modeling and simulation studies using Lab VIEW", Transaction id
2. Presented conference paper on "Neuron modeling and simulation studies using Lab VIEW', In WASET conference, Norway 2009.

Permanent address: s/o vengala rao,
Kandlagunta (v.g),
Nekarikallu (m.d),
Guntur (d.t), Andhra Pradesh.

Contact email: ram99@ymail.com,raamelectrical@gmail.com