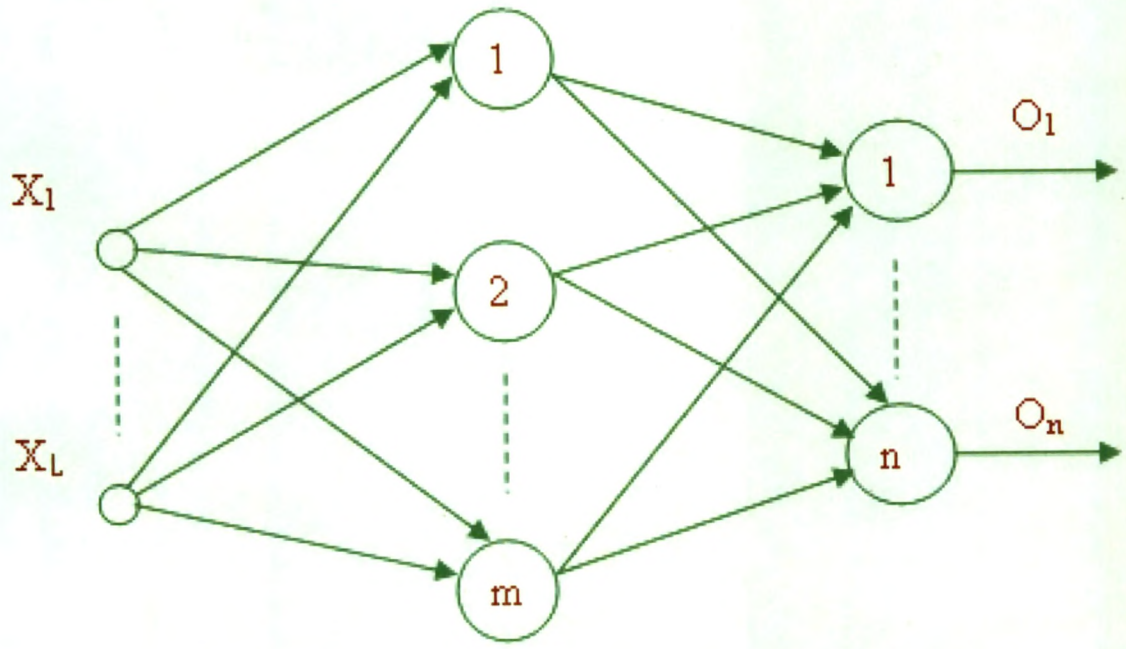
Chapter 2

Preliminaries



Chapter 2

PRELIMINARIES

2.1 Introduction to Artificial Neural Networks

Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by examples. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well. They are extremely simplified model of brain. They act essentially as function approximator, which transforms inputs into outputs to the best of its ability.

The similarity between the two is as shown in the Figure 2.1. Basically, Neural Network is composed of many neurons that co-operate to perform the desired function. The architectures of ANN are inspired by the working of the brain structure which is made up of numerous neurons and connections between them.

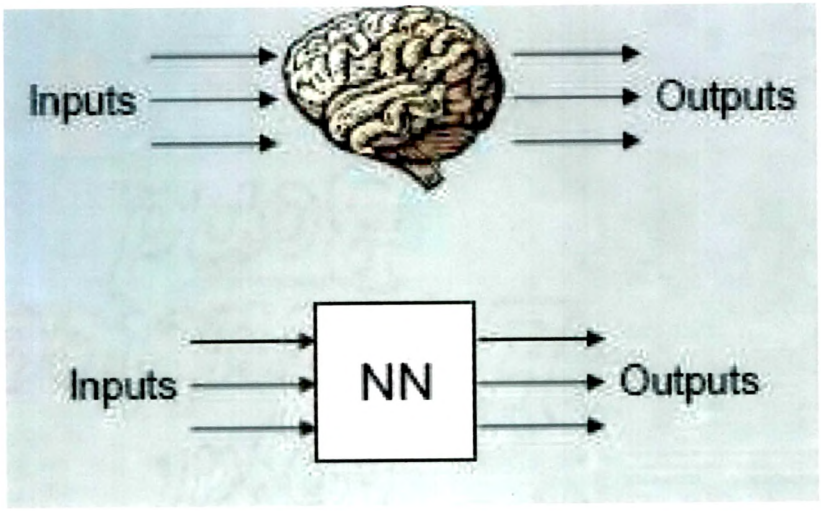


Figure 2.1: Similarity between brain and ANN working

Brief History of Neural Networks :

Neural network simulations appear to be a recent development. However, this field was established before the advent of computers, and has survived at least one major setback and several eras.

Many important advances have been boosted by the use of inexpensive computer emulations. Following an initial period of enthusiasm, the field survived a period of frustration and disrepute. During this period, when funding and professional support was minimal, important advances were made by relatively few researchers. These pioneers were able to develop convincing technology which surpassed the limitations identified by Minsky and Papert. Minsky and Papert, published a book (in 1969) in which they summed up a general feeling of frustration (against neural networks) among researchers and was thus accepted by most without further analysis. Currently, the neural network field enjoys a resurgence of interest and a corresponding increase in funding.

The first artificial neuron was produced in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pitts. In order to describe how neurons in the brain might work, they modeled a simple neural network using electrical circuits. This simple neuron was able to emitate the simple boolean functions like AND, OR

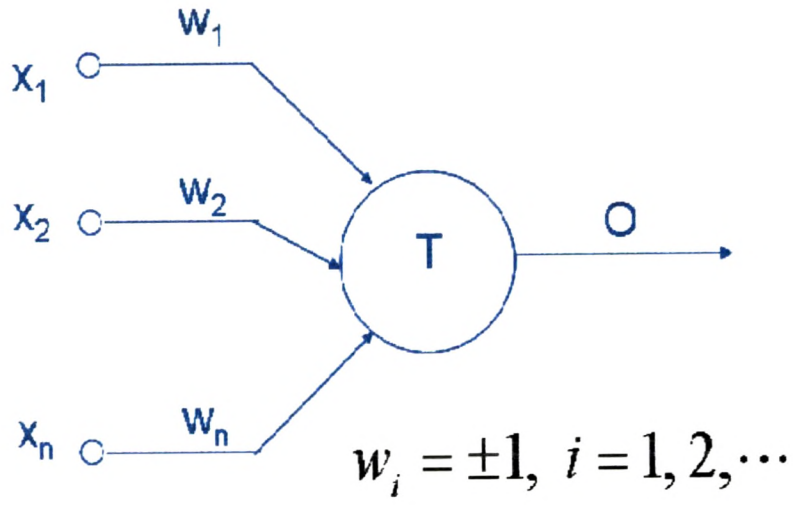


Figure 2.2: McCulloc-Pitts's Artificial Neuron

and NOT. The weights in this neuron can be ± 1.

In 1949, Donald Hebb published *"The Organization of Behavior"*, a work which pointed out the fact that neural pathways are strengthened each time they are used, a concept fundamentally essential to the ways in which humans learn. If two nerves fire at the same time, he argued, the connection between them is enhanced.

In 1959, Bernard Widrow and Marcian Hoff of Stanford developed models called "ADALINE" (ADAptive LINear Elements) and "MADALINE" (Multiple ADAptive LINear Elements). ADALINE was developed to recognize binary patterns so that if it was reading streaming bits from a phone line, it could predict the next bit. MADA-

LINE was the first neural network applied to a real world problem, using an adaptive filter that eliminates echoes on phone lines. While the system is as ancient as air traffic control systems, like air traffic control systems, it is still in commercial use.

In 1962, Widrow & Hoff developed a learning procedure that examines the value before the weight adjusts it (i.e. 0 or 1) according to the rule: Weight Change = (Pre-Weight line value) * (Error / (Number of Inputs)).

Despite the later success of the neural network, traditional Von Neumann architecture took over the computing scene, and neural research was left behind. Ironically, John Von Neumann himself suggested the limitation of neural functions by using telegraph relays or vacuum tubes.

In 1972, Kohonen and Anderson developed a similar network independently of one another, in which they were creating an array of analog ADALINE circuits.

The first multilayered network was developed in 1975, an unsupervised network.

In 1982, John Hopfield of Caltech presented a paper to the National Academy of Sciences. His approach was to create more useful machines by using bidirectional lines. Previously, the connections between neurons was only one way. That same year, Reilly and Cooper used a "Hybrid network" with multiple layers, each layer using a different problem-solving strategy.

Also in 1982, there was a joint US-Japan conference on Cooperative/Competitive Neural Networks. Japan announced a new Fifth Generation effort on neural networks, and US papers generated worry that the US could be left behind in the field. (Fifth generation computing involves artificial intelligence. First generation used switches and wires, second generation used the transistor, third state used solid-state technology like integrated circuits and higher level programming languages, and the fourth generation is code generators.) As a result, there was more funding and thus more research in the field.

In 1986, with multiple layered neural networks in the news, the problem was how to extend the Widrow-Hoff rule to multiple layers. Three independent groups of researchers, one of which included David Rumelhart, a former member of Stanford's psychology department, came up with similar ideas which are now called back-propagation networks because it distributes pattern recognition errors throughout the network. Hybrid networks used just two layers, these back-propagation networks use many. The result is that, back-propagation networks are "slow learners," needing possibly thousands of iterations (epochs) to learn.

Now, neural networks are used in several applications. The fundamental idea behind the nature of neural networks is that, if it works in nature, it must be able to work in computers. The future of neural networks, though, lies in the development of hardware. Much like the advanced chess-playing machines like Deep Blue, fast and efficient neural networks depend on hardware being specified for its eventual use.

Research that concentrates on developing neural networks is relatively slow. Due to the limitations of processors, neural networks take weeks to learn. Some companies are trying to create what is called a "silicon compiler" to generate a specific type of integrated circuit that is optimized for the application of neural networks. Digital, analog, and optical chips are the different types of chips being developed. One might immediately discount analog signals as a thing of the past. However neurons in the brain actually work more like analog signals than digital signals. While digital signals have two distinct states (1 or 0, on or off), analog signals vary between minimum and maximum values. It may be a while, though, before optical chips can be used in commercial applications.

Similarities - Human and Artificial Neurons :

Biological Neurons :

The brain is principally composed of about 10 billion neurons, each connected to about 10,000 other neurons. Each of the blobs in the Figure 2.3 are neuronal cell bodies (soma), and the lines are the input and output channels (dendrites and axons) which connect them.

Each neuron receives electrochemical inputs from other neurons at the dendrites. If the summation of these electrical inputs is sufficiently powerful to activate the neuron, it transmits an electrochemical signal along the axon, and passes this signal to the other neurons whose dendrites are attached at any of the axon terminals. These attached neurons may then fire. It is important to note that a neuron fires, only if the total signal received at the cell body exceeds a certain level. The neuron either fires or it doesn't, there aren't different grades of firing. That is, our entire brain is composed of these interconnected electro-chemical transmitting neurons. From a very large number of extremely simple processing units (each performing a weighted sum of its inputs, and then firing a binary signal if the total input exceeds a certain level) the brain manages to perform extremely complex tasks.

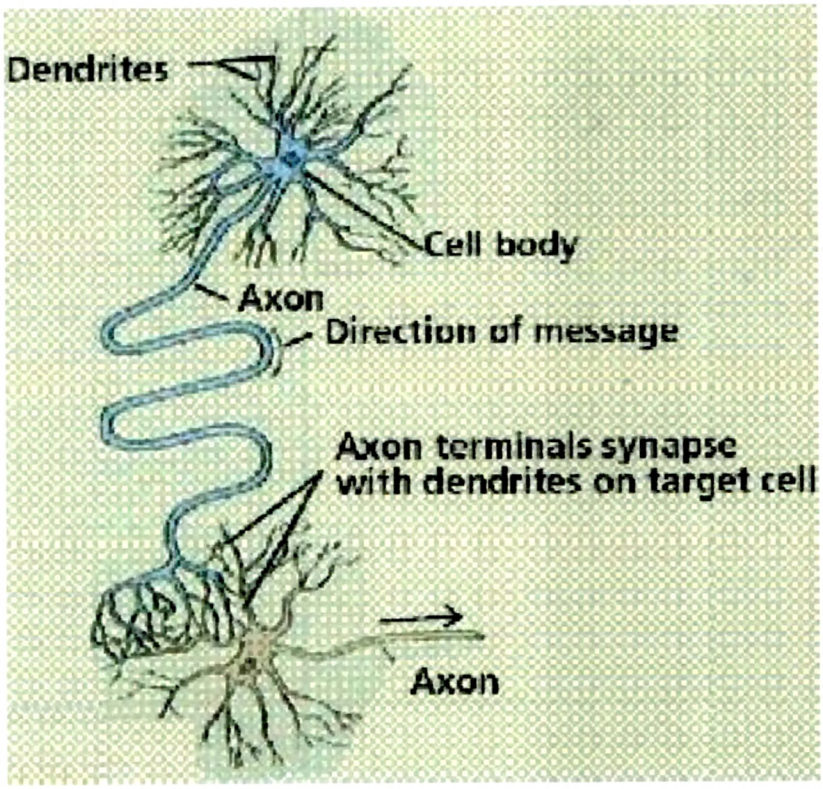


Figure 2.3: Components of a biological neuron

Artificial Neurons :

The simplest Artificial neuron was formulated by McCulloch and Pitts (1943) , shown in Figure 2.2, as described before.

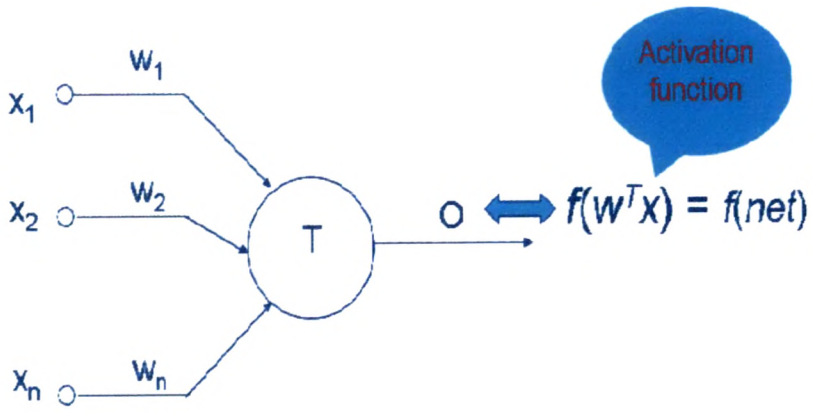


Figure 2.4: Perceptron

A simple neuron - Perceptron :

The perceptron, introduced by Rosenblatt(1958) is an advanced mathematical model of a biological neuron. While in actual neurons the dendrite receives electrical signals from the axons of other neurons, in the perceptron these electrical signals are represented as numerical values. At the synapses between the dendrite and axons, electrical signals are modulated in various amounts. This is also modeled in the perceptron by multiplying each input value by a value called the weight. An actual neuron fires an output signal only when the total strength of the input signals exceed a certain threshold. This phenomenon in a perceptron is modeled by calculating the weighted sum of the inputs to represent the total strength of the input signals, and applying a step function on the sum to determine its output, as shown in the Figure 2.4. As in biological neural networks, this output is fed to other perceptrons. Thus, an artificial neuron is a device with many inputs and one output.

This neuron has two modes of operation; the training mode and the using mode. In the training mode, the neuron can be trained to fire (or not), for particular input patterns. In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong

in the taught list of input patterns, the firing rule is used to determine whether to fire or not. In mathematical terms, the neuron fires if and only if

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

where, X_1, X_2,... are inputs, W_1, W_2,... are weights and T is the threshold. The inclusion of input weights and the threshold makes this neuron a very flexible and powerful one. This neuron has the ability to adapt to a particular situation by changing its weights and/or threshold. Various algorithms exist that cause the neuron to 'adapt'; the most used ones are the Delta rule and the Back-propagation. The former is used in feed-forward single layer networks and the latter in feed-forward multi layered networks. Hence forth we will interchangeably use the words perceptron, neuron or node they all mean the processing units in ANNs.

Transfer Function :

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

- linear (or ramp)
- threshold
- sigmoid

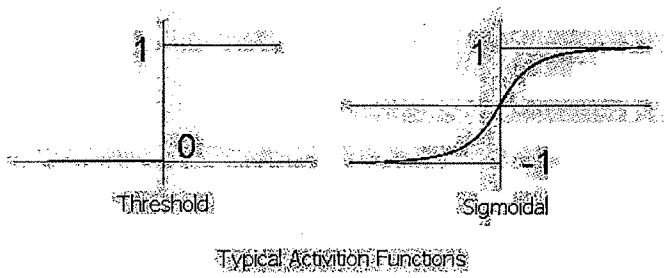


Figure 2.5: Activation Functions.

For linear units, the output activity is proportional to the total weighted output.

For threshold units, the output is set at one of two levels, depending on whether the total input is greater than or less than some threshold value.

For sigmoid units, the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurones than do linear or threshold units, but all three must be considered rough approximations.

Utility 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. Other advantages include:

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 are 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.

Applications of Neural Networks :

Character Recognition -

The idea of character recognition has become very important, as handheld devices like the Palm Pilot are becoming increasingly popular. Neural networks can be used to recognize handwritten characters.

Image Compression -

Neural networks can receive and process vast amounts of information at once, making them useful in image compression. With the Internet explosion and more sites using more images on their sites, using neural networks for image compression is worth a look.

Stock Market Prediction -

The day-to-day business of the stock market is extremely complicated. Many factors weigh in whether a given stock will go up or down on any given day. Since neural networks can examine a lot of information quickly and sort it all out, they can be used to predict stock prices.

Traveling Salesman's Problem -

Interestingly enough, neural networks can solve the traveling salesman problem, but only to a certain degree of approximation.

Medicine, Electronic Nose, Security, and Loan Applications -

These are some applications that are in their proof-of-concept stage, with the acception of a neural network that will decide whether or not to grant a loan, something that has already been used more successfully than many humans.

Architecture of neural networks :

Single-Layer Feedforward networks :

In a layered neural network the neurons are organized in the form of layers. In the simplest form of a layered network, we have an input layer of source nodes that projects onto an output layer of neurons (computation neurons). In other words, the network is strictly a feedforward or acyclic type. It is shown in Figure 2.6. The name 'single-layer' refers to the output layer of computation nodes. Hence, such a network is called *single-layer* network.

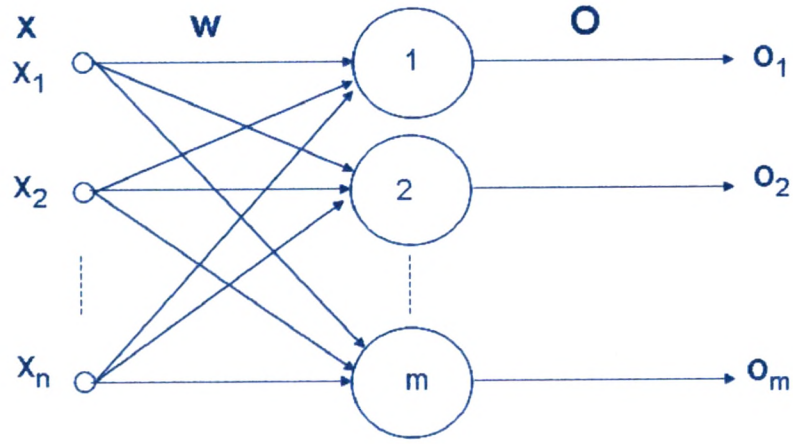


Figure 2.6: Single layered Architecture

Multilayer Feed-forward networks :

The common type of artificial neural network consists of three layers, of units: a layer of "input" units is connected to a layer of "hidden" units, which is connected to a layer of "output" units. Feed-forward networks, as shown in Figure 2.7, have the following characteristics:

1. Nodes are arranged in layers, with the first layer taking in inputs and the last layer producing outputs. The middle layers have no connection with the external world, and hence are called hidden layers.

2. Each node in one layer is connected to every node on the next layer. Hence information is constantly "fed forward" from one layer to the next., and this explains why these networks are called feed-forward networks.

3. There is no connection among nodes in the same layer.

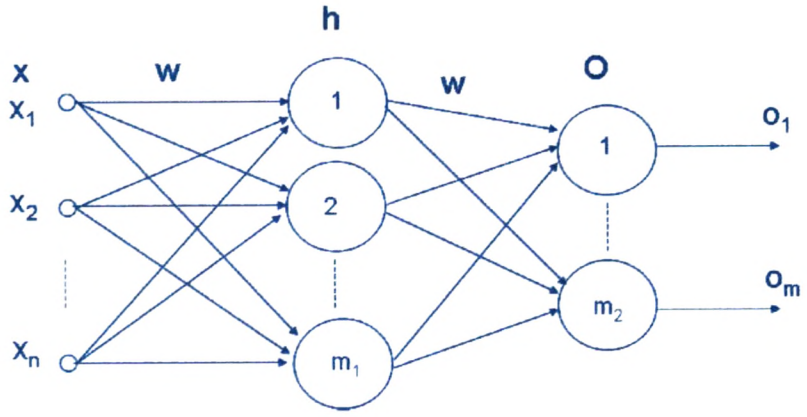


Figure 2.7: A multilayer feedforward network

Feed-forward ANNs, Figure 2.7 allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer. Feed-forward ANNs tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organization is also referred to as bottom-up or top-down. Working of such networks can be summed up as follows:

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

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

Feedback networks :

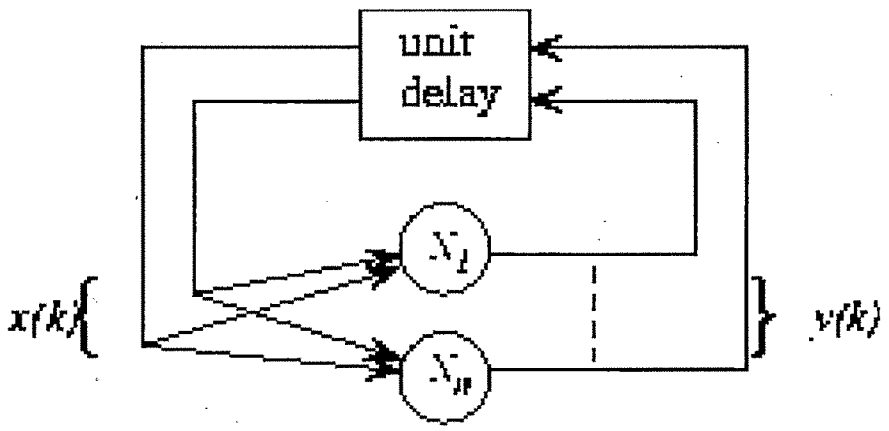


Figure 2.8: A recurrent network

Feedback networks, as shown in Figure 2.8, can have signals traveling in both directions by introducing loops in the network. Feedback networks are very powerful and can get extremely complicated. They are dynamic, that is, their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent, although the latter term is often used to denote feedback connections in single-layer organizations.

The Learning Process

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

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

- Auto-association: An input pattern is associated with itself and the states of input and output units coincide. This is used to provide pattern completion, ie to produce a pattern whenever a portion of it or a distorted pattern is presented.
- Hetero-association: In this case, the network actually stores pairs of patterns building an association between two sets of patterns.

  It is related to two recall mechanisms:

  1. Nearest-neighbor recall, where the output pattern produced corresponds to the input pattern stored, which is closest to the pattern presented, and
  2. Interpolative recall, where the output pattern is a similarity dependent interpolation of the patterns stored corresponding to the pattern presented. Yet another paradigm, which is a variant associative mapping is classification, ie when there is a fixed set of categories into which the input patterns are to be classified.

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

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

Information is stored in the weight matrix W of a neural network. Learning is the determination of the weights. The learning is performed in the following way, based on it one can distinguish two major categories of neural networks:

- *Fixed networks:* in which the weights cannot be changed, That is, $\frac{dW}{dt} = 0$. In such networks, the weights are fixed a priori according to the problem to solve.
- *Adaptive networks:* which are able to change their weights, That is, $\frac{dW}{dt} \neq 0$.

All learning methods used for adaptive neural networks can be classified into two major categories:

- *Supervised:* learning which incorporates an external teacher, so that each output unit is told what its desired response to input signals ought to be. During the learning process global information may be required. Paradigms of supervised learning include error-correction learning, reinforcement learning and stochastic learning.
  An important issue concerning supervised learning is the problem of error convergence, that is, the minimization of error between the desired and computed unit values. The aim is to determine a set of weights which minimizes the error. One well-known method, which is common to many learning paradigms is the least mean square (LMS) convergence.
- *Unsupervised:* learning uses no external teacher and is based upon only local information. It is also referred to as self-organization, in the sense that it self-organizes data presented to the network and detects their emergent collective properties. Paradigms of unsupervised learning are Hebbian learning and competitive learning.

From Human Neurons to Artificial Neurons the aspect of learning concerns the distinction or not of a separate phase, during which the network is trained, and a subsequent operation phase. We say that a neural network learns off-line if the learning phase and the operation phase are distinct. A neural network learns on-line if it learns and operates at the same time. Usually, supervised learning is performed off-line, whereas unsupervised learning is performed on-line.

To design a neural network that performs some specific task, we must choose how the units are connected to one another, and we must set the weights on the connections appropriately. The connections determine whether it is possible for one unit to influence another. The weights specify the strength of the influence.

We can teach a three-layer network to perform a particular task by using the following procedure:

1. We present the network with training examples, which consist of a pattern of activities for the input units together with the desired pattern of activities for the output units.
2. We determine how closely the actual output of the network matches the desired output.
3. We change the weight of each connection so that the network produces a better approximation of the desired output.

The Back-Propagation Algorithm :

In order to train a neural network to perform some task, we must adjust the weights of each unit in such a way that the error between the desired output and the actual output is reduced. One such form of supervised training is done using an algorithm known as Backpropagation algorithm.

The basic Backpropagation algorithm is based on minimizing the error of the network using the derivatives of the error function. Most common measure of error is the mean square error: $E = (target - output)^2$.

Let f be the activation function then partial derivatives of the error with respect to the weights for the output as well hidden neurons are:

Output Neurons:

$$\delta_j = f'(net_j)(target_j - Output_j)$$
$$\partial E/\partial w_{ji} = -output_i \delta_j$$

where, j = output neuron and i = neuron in last hidden layer.

Hidden Neurons:

$$\delta_j = f'(net_j) \sum(\delta_k w_{kj})$$
$$\partial E/\partial w_{ji} = -output_i \delta_j$$

where, j = hidden neuron and i = neuron in previous layer and k = neuron in next layer.

Since, the calculation of the derivatives flows backwards through the network, hence the name, Backpropagation. These derivatives point in the direction of maximum increase of the error function.

The correction Δw_{ji} applied to the synaptic weight connecting neuron i to neuron j is defined by the delta rule.

Weight correction = learning rate parameter * local gradient * input to neuron j

That is,

$$\Delta w_{ji}(n) = \alpha.\delta_j(n).y_i(n)$$

Hence,

$$w_{new} = w_{old} - \alpha \partial E / \partial w_{old}$$

The learning rate is very important, if it is too small the convergence will be extremely slow and if the α is too large there may not be any convergence.

A small step (learning rate) in the opposite direction will result in the maximum decrease of the (local) error function.

There are issues of local minimum with Backpropagation algorithm to overcome it we use Backpropagation with momentum as it tends to aid convergence and applies smoothed averaging to the change in weights as follows

$$\Delta_{new} = \beta \Delta_{old} - \alpha \partial E / \partial w_{old}$$

where, β is the momentum coefficient. And we get new weight change rule as

$$w_{new} = w_{old} + \Delta_{new}$$

Such a modification in a Backpropagation algorithm acts as a low-pass filter by reducing rapid fluctuations.

Basically, training in the algorithm is essentially minimizing the mean square error function keeping in mind to avoid local minima.

Traditional techniques for avoiding local minima are:

Simulated annealing: Perturb the weights in progressively smaller amounts

Genetic algorithms: Use the weights as chromosomes, Apply natural selection, mating, and mutations to these chromosomes.

An incident to share :

(By Neil Fraser, September 1998)

In the 1980s, the Pentagon wanted to harness computer technology to make their tanks harder to attack.

The Plan: The preliminary plan was to fit each tank with a digital camera hooked up to a computer. The computer would continually scan the environment outside for possible threats (such as an enemy tank hiding behind a tree), and alert the tank crew to anything suspicious. Computers are really good at doing repetitive tasks without taking a break, but they are generally bad at interpreting images. The only possible way to solve the problem was to employ a neural network.

The Implementation: The research team went out and took 100 photographs of tanks hiding behind trees, and then took 100 photographs of trees - with no tanks. They took half the photos from each group and put them in a vault for safe-keeping, then scanned the other half into their mainframe computer. The huge neural network was fed each photo one at a time and asked if there was a tank hiding behind the trees. Of course at the beginning its answers were completely random since the network didn't know what was going on or what it was supposed to do. But each time it was fed a photo and it generated an answer, the scientists told it if it was right or wrong. If it was wrong it would randomly change the weights in its network until it gave the correct answer. Over time it got better and better until eventually it was getting each photo correct. It could correctly determine if there was a tank hiding behind the trees in any one of the photos.

Verification: But the scientists were worried: had it actually found a way to recognize if there was a tank in the photo, or had it merely memorized which photos had tanks and which did not? This is a big problem with neural networks, after they have been trained you have no idea how they arrive at their answers, they just do. The question was did it understand the concept of tanks vs. no tanks, or had it merely memorized the answers? So the scientists took out the photos they had been keeping in the vault and fed them through the computer. The computer had never seen these photos before – this would be the big test. To their immense relief the neural net correctly identified each photo as either having a tank or not having one.

Independent testing: The Pentagon was very pleased with this, but a little bit suspicious. They commissioned another set of photos (half with tanks and half without) and scanned them into the computer and through the neural network. The

results were completely random. For a long time nobody could figure out why. After all nobody understood how the neural had trained itself.

Grey skies for the US military: Eventually someone noticed that in the original set of 200 photos, all the images with tanks had been taken on a cloudy day while all the images without tanks had been taken on a sunny day. The neural network had been asked to separate the two groups of photos and it had chosen the most obvious way to do it - not by looking for a camouflaged tank hiding behind a tree, but merely by looking at the color of the sky. The military was now the proud owner of a multi-million dollar mainframe computer that could tell you if it was sunny or not.

The Lesson: Training and testing are the most vital phases of using neural networks for any application.

Neural Network Control :

A lot of research has been done on using feed-forward neural networks as the adaptive component in a learning controller [76]. The network weights can be adjusted using the Backpropagation algorithm, genetic algorithms [59], or various stochastic search algorithms (for example, Alopex [77] and statistical gradient following [63]). Supervised training is usually performed using error signals derived from the systems performance error, although other approaches which transfer expert information from a rule base are common. Such an approach is implemented in [53], and other author have used a fuzzy rule base. Several control approaches have been developed which perform training on the system with its controlling neural network unfolded over discrete time. Backpropagation through time [9] propagates error information backwards through time, and forward propagation algorithms [63] do the reverse. Such algorithms can also train recurrent neural network controllers that have their own dynamical properties. These algorithms have been generalized to continuous systems [39] and made more efficient in various ways. Although theoretically elegant, forward and backward propagation approaches are ill suited to practical on-line control. Others have used a more successful analytical control-theory approach to train a neural network so that it becomes an inverse (in some sense) of the system being controlled ([25], [79], [45],).

Neural networks can be used for approximating a controller because of two facts:

- Firstly, Multilayer feed-forward Neural Networks are universal approximator. (refer [22] and [33])

- Secondly, due to the Backpropagation Algorithm.

The Backpropagation Algorithm performs gradient descent. Recall that the change in weights after k iterations can be given as

$$w(k+1) = w(k) - \alpha(k)e(k).$$

For the dynamical system

$$x(k+1) = f(x(k), u(k))$$

$$x(0) = 0$$

neural network approximation for system with the controller being implemented using neural network can be given as:

$$\hat{x}(k+1) = NN_{f_i}[\hat{x}, NN_u(\hat{x}(k))] \tag{2.1.1}$$

with

$$NN_f[0,0] = 0.$$

2.2 Basics of Control Theory

In Control Theory one is interested in the behavior of model of the physical system working under the influence of physical laws and external inputs. Once the system is modeled it analyzed for the existence and uniqueness of solution, stability, controllability, observability etc. The dynamical systems can be classified into being Linear, Nonlinear; Time variant, Time-invariant; Continuous, Discrete; Autonomous, Non-Autonomous etc.

In this work we will be mainly dealing with the semilinear systems both in the discrete as well as the continuous form. The semilinear system is the one which has the linear part as well as the nonlinear part. In this section we present required results of the Linear Systems.

Continuous Linear System

The continuous time variant linear system is given by:

$$\dot{x(t)} = A(t)x(t) + B(t)u(t)$$

$$x(t_0) = x_0 \tag{2.2.2}$$

where, $A(t)_{n\times n}$ is the system matrix, which is assumed to be continuous for all t, $B(t)_{n\times m}$ is the control matrix. The state $x(t) \in X \subseteq R^n$, $u(t) \in U \subseteq R^m$ is the control input to the system.

DEFINITION 2.2.1 ***Controllability:*** *A dynamical system (2.2.2) is said to be completely controllable, if for any initial and final states x_i and x_T in the state space X, there exist control u that will steer the system from $x(t_0) = x_i$ to $x(T) = x_T$, during $[t_0, T]$.*

The *state transition matrix* for the homogeneous part of the system (2.2.2) is denoted by $\Phi(t, t_0)$ and is given by

$$\Phi(t, t_0) = \psi(t)\psi^{-1}(t) \tag{2.2.3}$$

where $\psi(t)$ is called *fundamental matrix* whose n columns consist of n linearly independent solutions of the homogeneous part of (2.2.2). The state transition matrix given by (2.2.3) satisfies the differential equation

$$\frac{d}{dt}\Phi(t, t_0) = A(t)\Phi(t, t_0) \tag{2.2.4}$$

and the boundary conditions

$$\Phi(t_0, t_0) = I$$

The *solution* for the system (2.2.2) is given by

$$x(t) = \Phi(t, t_0)x_0 + \int_{t_0}^{t} \Phi(t, \tau)B(\tau)u(\tau)d\tau \tag{2.2.5}$$

The controllability for the system (2.2.2) is guaranteed if the controllability grammian is invertible (refer Brockett citeBRW. The *controllability grammian* for the system (2.2.2) is given by

$$W(t_0, T) = \int_{t_0}^{T} \Phi(t_0, t)B(t)B^*(t)\Phi^*(t_0, t)dt$$

and the *minimum norm steering control* for the system (2.2.2) is given by

$$u(t) = -B^*(t)\Phi^*(t_0, t)W^{-1}(t_0, T)(x_0 - \Phi(t_0, T)x(T)) \tag{2.2.6}$$

In the system (2.2.2) if $A(t)$ and $B(t)$ are time invariant then the system can be written as

$$\dot{x}(t) = Ax(t) + Bu(t)$$

$$x(t_0) = x_0 \tag{2.2.7}$$

for which the state transition matrix is given by

$$\Phi(t, t_0) = e^{A(t-t_0)} = \Phi(t - t_0) \tag{2.2.8}$$

The solution for the system (2.2.7) is given by

$$x(t) = e^{A(t-t_0)}(x_0 + \int_{t_0}^{t} e^{A(t_0-\tau)}Bu(\tau)d\tau) \tag{2.2.9}$$

The controllability grammian is given by

$$W(t_0, T) = \int_{t_0}^{T} e^{A(t_0-t)}BB^*e^{A^*(t_0-t)}dt$$

and the minimum norm controller is given by

$$u(t) = -B^*e^{A^*(t_0-t)}W^{-1}(t_0, T)(x_0 - e^{A(t_0-T)}x(T)) \tag{2.2.10}$$

Discrete Linear System :

The discrete time variant linear system is given by:

$$x(k+1) = F(k)x(k) + G(k)u(k)$$

$$x(k_0) = x_0 \tag{2.2.11}$$

where, $F(k)_{n\times n}, G(k)_{n\times m}$ are time dependent matrices and $F(k)$ is non-singular for all k. The state $x(k) \in X \subseteq R^n$, $u(k) \in U \subseteq R^m$ is the control input to the system.

For the system (2.2.11) the state transition matrix is given by $\Phi(k, k_0) = \prod_{i=k_0}^{k-1} F(i)$. The solution for the system (2.2.11), for $k \geq k_0$ is given by

$$x(k) = \Phi(k, k_0)x_0 + \Phi(k, k_0) \sum_{i=k_0+1}^{k} \Phi^{-1}(i, k_0)G(i-1)u(i-1)$$

The controllability Grammian matrix is given by

$$W(k_0, k_1) = \sum_{j=k_0}^{k_1-1} \Phi(k_0, j+1)G(j)G^T(j)\Phi^T(k_0, j+1)$$

The system (2.2.11) is controllable if and only if the controllability Grammian $W(k_0, k_1)$ is invertible. Thus, System (2.2.11) will be controllable if and only if n rows of $n \times m$ matrix functions $\Phi(k_0, k+1)G(k)$ are linearly independent on $[k_0, k_1]$.

In this case, a control $u(k)$ which steers the system (2.2.11) from x_0 to x_1 is given by

$$u(k) = -G^T(k)\Phi^T(k_0, k+1)W^{-1}(k_0, k_1)[x_0 - \Phi(k_0, k_1)x_1] \tag{2.2.12}$$

In case when $F(k)$ and $G(k)$ are time invariant the system (2.2.11) becomes

$$x(k+1) = Fx(k) + Gu(k)$$
$$x(k_0) = x_0 \tag{2.2.13}$$

where, $F_{n\times n}, G_{n\times m}$ are time independent matrices and F is invertible. The state $x(k) \in X \subseteq R^n$, $u(k) \in U \subseteq R^m$ is the control input to the system. For the system (2.2.13) the state transition matrix is given by

$$\Phi(k, k_0) = \Phi(k - k_0) = F^{(k-k_0)}$$

The solution for the system (2.2.13), for $k \geq k_0$ is given by

$$x(k) = F^{(k-k_0)}x_0 + F^{(k-k_0)} \sum_{i=k_0+1}^{k} F^{-(i-k_0)}Gu(i-1)$$

If $k_0 = 0$, as is usually assumed in the time-invariant case, we get solution as

$$x(k) = F^k x_0 + \sum_{i=1}^{k} F^{k-i}Gu(i-1)$$

The discrete-time system (2.2.13) is controllable if and only if the rank of $(n \times nm)$ controllability matrix U where, $U \equiv [G|FG|...|F^{n-1}G]$ is n. That is, $\rho(U) = n$. In this case, a control u which steers the system (2.2.13) from x_0 to x_1 is given by

$$u = U^{-1}[x_1 - F^n x_0] \tag{2.2.14}$$

2.3 Definitions :

Consider the dynamical system

$$\dot{x}(t) = Ax(t) + Bu(t) + f(x(t), u(t))$$
$$x(t_0) = x_i \tag{2.3.15}$$

For a dynamical system state controllability usually mean that it is possible - by admissible inputs - to steer the states from any initial state to any final state within some finite time. Putting into mathematical perspective

DEFINITION 2.3.1 ***Controllability:*** *A dynamical system (2.3.15) is said to be completely controllable, if for any initial and final states x_i and x_T in the state space X, there exist control u that will steer the system from $x(t_0) = x_i$ to $x(T) = x_T$, during $[t_0, T]$.*

For linear systems (2.2.2) there is a test to check if a system is controllable or not (called Kalman's condition). For a system with n dimensional state vector, if the rank of the following controllability matrix

$$\left[\ B|AB|A^2B|\cdots|A^{n-1}B\ \right]$$

is equal to n, then the system is controllable.

DEFINITION 2.3.2 ***Local Controllability:*** *A dynamical system is locally controllable around an equilibrium state $x = 0$, if for every neighborhood W_x of the origin, there is some neighborhood V_x of the origin such that for any two states x_i, x_T in V_x, there exist a control that will transfer the system from x_i to x_f without leaving W_x.*

DEFINITION 2.3.3 ***Reachability:*** *Let $T > 0$. We define for any initial data $x_0 \in X$, the set of reachable state as*

$$R(T; x_0) = x(T) : x \text{ is a solution of (2.3.15) with } u \in L^2(0, T; X)$$

DEFINITION 2.3.4 ***Exactly Controllable:*** *The dynamical system (2.3.15) is said to be exactly controllable on $[0, T]$ if, for given x_0 and $\hat{x}$ in R^n, there exists a control $u \in L^2(0, T; R^m)$ such that the corresponding solution $x(t)$ of (2.3.15) satisfies $x(T) = \hat{x}$. In other words, if the set of reachable states $R(T, x_0)$ coincides with R^n, system (2.3.15) is exactly controllable.*

DEFINITION 2.3.5 ***Approximately Controllable:*** *Let X and U be Hilbert Spaces. The dynamical system (2.3.15) is said to be approximately controllable on $[0, T]$ if, for every initial state $x_0 \in X$, the set of reachable states $R(T, x_0)$ is dense in X. That is, the system (2.3.15) is approximately controllable, if for any given x_0 and $\hat{x}$ in X and any $\epsilon > 0$, there exists a control $u \in L^2(0, T; U)$ such that $\| x(T) - \hat{x} \| < \epsilon$.*

In control theory, Observability is a measure for how well internal states of a system can be inferred by knowledge of its external outputs. The observability and controllability of a system are mathematical duals.

DEFINITION 2.3.6 ***Observability:*** *A dynamical system is said to be observable if from current state its previous state can be observed, or, from observations it is possible to get the complete state or initial state.*

For linear systems in the state space representation, with the associated output equation $y(t) = Cx(t)$, there is a convenient test to check if a system is observable. For a system with n states, if the rank of the following observability matrix

$$\begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

is equal to n, then the system is observable. The rationale for this test is that if n rows are linearly independent, then each of the n states is viewable through linear combinations of the output variables $y(t)$.

DEFINITION 2.3.7 ***BIBO Stable:*** *BIBO stands for Bounded-Input Bounded-Output. A dynamical system is said to be BIBO stable if the the output will be bounded for every input to the system that is bounded.*

DEFINITION 2.3.8 ***Asymptotic Stable:*** *A dynamical system is said to be asymptotic stable if there is no change in the system for $t > T$.*

DEFINITION 2.3.9 ***Moore Penrose Inverse:*** *Given an $m \times n$ matrix B, the Moore-Penrose inverse of B is a unique $n \times m$ generalized inverse of B and is denoted by B^+. This matrix is variously known as the generalized inverse, pseudoinverse, or Moore-Penrose inverse.*
The Moore-Penrose inverse satisfies

$$BB^+B = B$$
$$B^+BB^+ = B^+$$
$$(BB^+)^T = BB^+$$
$$(B^+B)^T = B^+B$$

DEFINITION 2.3.10 ***C_0-Semigroup:*** *Let X be a Banach space and A be linear operator on X. A strongly continuous family $\{T(t)\}_{t\geq 0}$ of bounded operators in X is called a* ***C_0-semigroup*** *generated by A if*

(i) $T(t+s)x = T(t)T(s)x,\ x \in X$ *and* $t, s \geq 0$,

(ii) $T(0)x = x,\ x \in X$

(iii) $t \mapsto T(t)x$ *is continuous for* $t \geq 0, x \in X$,

(iv) $Ax = \lim_{t\to 0^+} \frac{T(t)x-x}{t} = \frac{d^+T(t)x}{dt}|_{t=0}; \qquad x \in D(A)$,

where $D(A) = \{x \in X : lim_{t\to 0^+} \frac{T(t)x-x}{t}\}$ *exists.*

Here $D(A)$ *is the domain of* A.

We have following property of C_0-semigroup. Let $S(t)$ be a strongly continuous semigroup and let A be its infinitesimal generator. Then, for $x \in D(A)$ we have that $S(t)x \in D(A)$ and

$$\frac{d}{dt}S(t)x = AS(t)x = S(t)Ax, \forall t \geq 0.$$

2.4 Theorems form functional Analysis :

THEOREM 2.4.1 ***Inverse Function Theorem:*** *Let E and F be normed vector spaces. Let U be open in E, let $a \in U$ and let $f : U \to F$ be C^p (i.e. continuous p derivative) map. Assume that the Jacobian $Df(a) : E \to F$ is invertible. Then f is locally C^p-invertible at a.*

THEOREM 2.4.2 ***Implicit Function Theorem:*** *Let E, F and G be normed vector spaces. Let U be open in $E \times F$, let $f : U \to G$ be C^p map. Let (a, b) be a point of U with $a \in E$ and $b \in F$. Let $f(a, b) = 0$. Assume that the Jacobian of f with respect to y denoted by $D_2f(a, b) : F \to G$ is invertible. Then there is an open ball V centered at a in E and a continuous map $g : V \to F$ such that $g(a) = b$ and $f(x, g(x)) = 0$ for all $x \in V$. If V is a sufficiently small ball, then g is uniquely determined and is of class C^p.*

THEOREM 2.4.3 ***Banach Contraction Mapping Theorem:*** *Let T be a contraction mapping of a Banach space X onto itself, i.e.*

$$\| T(x) - T(y) \| \leq \alpha \| x - y \|, \forall x, y \in X, 0 \leq \alpha < 1.$$

Then, Tf has a fixed point in X. That is, there exists $x^* \in X$ such that

$$Tx^* = x^*$$

and

$$x^{n+1} = Tx^n$$

converges to this fixed point for arbitrary x^0.

THEOREM 2.4.4 ***Generalized Contraction Principle:*** *Let $T : X \to X$ be an operator on a Banach space and if T^n is a contraction for some $n \geq 1$ then T has a fixed point*

THEOREM 2.4.5 ***Schauder's Fixed Point Theorem:*** *The Schauder fixed point theorem asserts that if K is a compact, convex subset of a Banach space and T is a mapping of K into itself, then T has a fixed point.*

THEOREM 2.4.6 ***Gronwall's Inequality:*** *Let f be a positive continuous function satisfying the following integral inequality*

$$f(t) \leq C + \int_0^t g(s)f(s)ds,$$

where, C is a positive constant and $g(t) \geq 0 (\forall t \in [0,T])$ is a continuous function. Then

$$f(t) \leq Ce^{\int_0^t g(s)ds}, t \in [0,T].$$

This chapter has the essentials for developing the work. In the next chapter we explore the properties of the Hopfield type Neural Networks.
