INTRODUCTION:

What is intelligence?

Real intelligence is what determines the normal thought process of a human.

Artificial intelligence is a property of machines which gives it ability to mimic the human thought process. The intelligent machines are developed based on the intelligence of a subject, of a designer, of a person, of a human being.

Now two questions: can we construct a control system that hypothesizes its own control law? We encounter a plant and looking at the plant behavior, sometimes, we have to switch from one control system to another control system where the plant is operating. The plant is may be operating in a linear zone or non- linear zone; probably an operator can take a very nice intelligent decision about it, but can a machine do it? Can a machine actually hypothesize a control law, looking at the model? Can we design a method that can estimate any signal embedded in a noise without assuming any signal or noise behavior?

That is the first part; before we model a system, we need to observe. That is we collect certain data from the system and How do we actually do this? At the lowest level, we have to sense the environment, like if I want to do temperature control I must have temperature sensor. This data is polluted or corrupted by noise. How do we separate the actual data from the corrupted data? This is the second question. The first question is that can a control system be able to hypothesize its own control law? These are very important questions that we should think of actually. Similarly, also to represent knowledge in a world model, the way we manipulate the objects in this world and the advanced is a very high level of intelligence that we still do not understand; the capacity to perceive and understand.

What is AI ?

Artificial Intelligence is concerned with the design of intelligence in an artificial device.

The term was coined by McCarthy in 1956.

There are two ideas in the definition.

1. Intelligence

2. artificial device

What is intelligence?

- Is it that which characterize humans? Or is there an absolute standard of judgment?

- Accordingly there are two possibilities:

  – A system with intelligence is expected to behave as intelligently as a human

  – A system with intelligence is expected to behave in the best possible manner

- Secondly what type of behavior are we talking about?

  – Are we looking at the thought process or reasoning ability of the system?
  – Or are we only interested in the final manifestations of the system in terms of its actions?

Given this scenario different interpretations have been used by different researchers as defining the scope and view of Artificial Intelligence.

1. One view is that artificial intelligence is about designing systems that are as intelligent as humans. This view involves trying to understand human thought and an effort to build machines that emulate the human thought process. This view is the cognitive science approach to AI.

2. The second approach is best embodied by the concept of the Turing Test. Turing held that in future computers can be programmed to acquire abilities rivaling human intelligence. As part of his argument Turing put forward the idea of an 'imitation game', in which a human being and a computer would be interrogated under conditions where the interrogator would not know which was which, the communication being entirely by textual messages. Turing argued that if the interrogator could not distinguish them by questioning, then it would be unreasonable not to call the computer intelligent. Turing's 'imitation game' is now usually called 'the Turing test' for intelligence.

3. Logic and laws of thought deals with studies of ideal or rational thought process and inference. The emphasis in this case is on the inference mechanism, and its properties. That is how the system arrives at a conclusion, or the reasoning behind its selection of actions is very important in this point of view. The soundness and completeness of the inference mechanisms are important here.

4. The fourth view of AI is that it is the study of rational agents. This view deals with building machines that act rationally. The focus is on how the system acts and performs, and not so much on the reasoning process. A rational agent is one that acts rationally, that is, is in the best possible manner.

Typical AI problems

While studying the typical range of tasks that we might expect an "intelligent entity" to perform, we need to consider both "common-place" tasks as well as expert tasks.

Examples of common-place tasks include

- *Recognizing* people, objects.
- Communicating (through *natural language*).
- *Navigating* around obstacles on the streets

These tasks are done matter of factly and routinely by people and some other animals.

Expert tasks include:

- Medical diagnosis.
- Mathematical problem solving
- Playing games like chess

These tasks cannot be done by all people, and can only be performed by skilled specialists.

Now, which of these tasks are easy and which ones are hard? Clearly tasks of the first type are easy for humans to perform, and almost all are able to master them. However, when we look at what computer systems have been able to achieve to date, we see that their achievements include performing sophisticated tasks like medical diagnosis, performing symbolic integration, proving theorems and playing chess.

On the other hand it has proved to be very hard to make computer systems perform many routine tasks that all humans and a lot of animals can do. Examples of such tasks include navigating our way without running into things, catching prey and avoiding predators. Humans and animals are also capable of interpreting complex sensory information. We are able to recognize objects and people from the visual image that we receive. We are also able to perform complex social functions.

Intelligent behaviour

This discussion brings us back to the question of what constitutes intelligent behaviour. Some of these tasks and applications are:

1. Perception involving image recognition and computer vision
2. Reasoning
3. Learning
4. Understanding language involving natural language processing, speech processing
5. Solving problems
6. Robotics

Practical applications of AI

AI components are embedded in numerous devices e.g. in copy machines for automatic correction of operation for copy quality improvement. AI systems are in everyday use for identifying credit card fraud, for advising doctors, for recognizing speech and in helping complex planning tasks. Then there are intelligent tutoring systems that provide students with personalized attention.

Thus AI has increased understanding of the nature of intelligence and found many applications. It has helped in the understanding of human reasoning, and of the nature of intelligence. It has also helped us understand the complexity of modeling human reasoning.

Approaches to AI

Strong AI aims to build machines that can truly reason and solve problems. These machines should be self aware and their overall intellectual ability needs to be indistinguishable from that of a human being. Excessive optimism in the 1950s and 1960s concerning strong AI has given way to an appreciation of the extreme difficulty of the problem. Strong AI maintains that suitably programmed machines are capable of cognitive mental states.

Weak AI: deals with the creation of some form of computer-based artificial intelligence that cannot truly reason and solve problems, but can act as if it were intelligent. Weak AI holds that suitably programmed machines can simulate human cognition.

Applied AI: aims to produce commercially viable "smart" systems such as, for example, a security system that is able to recognise the faces of people who are permittedto enter a particular building. Applied AI has already enjoyed considerable success.

Cognitive AI: computers are used to test theories about how the human mind works--for example, theories about how we recognise faces and other objects, or about how we solve abstract problems.

Limits of AI Today

Today"s successful AI systems operate in well-defined domains and employ narrow, specialized knowledge. Common sense knowledge is needed to function in complex, open-ended worlds. Such a system also needs to understand unconstrained natural language. However these capabilities are not yet fully present in today"s intelligent systems.

What can AI systems do

Today"s AI systems have been able to achieve limited success in some of these tasks.

- In Computer vision, the systems are capable of face recognition
- In Robotics, we have been able to make vehicles that are mostly autonomous.
- In Natural language processing, we have systems that are capable of simple machine translation.
- Today"s Expert systems can carry out medical diagnosis in a narrowdomain
- Speech understanding systems are capable of recognizing several thousand words continuous speech
- Planning and scheduling systems had been employed in scheduling experiments with the Hubble Telescope.
- The Learning systems are capable of doing text categorization into about a 1000 topics
- In Games, AI systems can play at the Grand Master level in chess (world champion), checkers, etc.

What can AI systems NOT do yet?

- Understand natural language robustly (e.g., read and understand articles in a newspaper)
- Surf the web
- Interpret an arbitrary visual scene
- Learn a natural language
- Construct plans in dynamic real-time domains
- Exhibit true autonomy and intelligence

Applications:

We will now look at a few famous AI system that has been developed over the years.

1. **ALVINN:** Autonomous Land Vehicle In a Neural Network

In 1989, Dean Pomerleau at CMU created ALVINN. This is a system which learns to control vehicles by watching a person drive. It contains a neural network whose input is a 30x32 unit two dimensional camera image. The output layer is a representation of the direction the vehicle should travel.

The system drove a car from the East Coast of USA to the west coast, a total of about 2850 miles. Out of this about 50 miles were driven by a human, and the rest solely by the system.

2. Deep Blue

In 1997, the Deep Blue chess program created by IBM, beat the current world chess champion, Gary Kasparov.

3. Machine translation

A system capable of translations between people speaking different languages will be a remarkable achievement of enormous economic and cultural benefit. Machine translation is one of the important fields of endeavour in AI. While some translating systems have been developed, there is a lot of scope for improvement in translation quality.

4. Autonomous agents

In space exploration, robotic space probes autonomously monitor their surroundings, make decisions and act to achieve their goals.

NASA's Mars rovers successfully completed their primary three-month missions in April, 2004. The Spirit rover had been exploring a range of Martian hills that took two months to reach. It is finding curiously eroded rocks that may be new pieces to the puzzle of the region's past. Spirit's twin, Opportunity, had been examining exposed rock layers inside a crater.

5. Internet agents

The explosive growth of the internet has also led to growing interest in internet agents to monitor users' tasks, seek needed information, and to learn which information is most useful

Q2

What is soft computing?

An approach to computing which parallels the remarkable ability of the human mind to reason and learn in an environment of uncertainty and imprecision.

It is characterized by the use of inexact solutions to computationally hard tasks such as the solution of nonparametric complex problems for which an exact solution can"t be derived in polynomial of time.

Why soft computing approach?

Mathematical model & analysis can be done for relatively simple systems. More complex systems arising in biology, medicine and management systems remain intractable to conventional mathematical and analytical methods. Soft computing deals with imprecision, uncertainty, partial truth and approximation to achieve tractability, robustness and low solution cost. It extends its application to various disciplines of Engineering and science. Typically human can:

1. Take decisions
2. Inference from previous situations experienced
3. Expertise in an area
4. Adapt to changing environment
5. Learn to do better
6. Social behaviour of collective intelligence

Intelligent control strategies have emerged from the above mentioned characteristics of human/ animals. The first two characteristics have given rise to Fuzzy logic;2^{nd} , 3^{rd} and 4^{th} have led to Neural Networks; 4^{th} , 5^{th} and 6^{th} have been used in evolutionary algorithms.

Characteristics of Neuro-Fuzzy & Soft Computing:

1. Human Expertise
2. Biologically inspired computing models
3. New Optimization Techniques
4. Numerical Computation
5. New Application domains
6. Model-free learning
7. Intensive computation
8. Fault tolerance
9. Goal driven characteristics
10. Real world applications

Intelligent Control Strategies (Components of Soft Computing): The popular soft computing components in designing intelligent control theory are:

1. Fuzzy Logic
2. Neural Networks
3. Evolutionary Algorithms

Fuzzy logic:

Most of the time, people are fascinated about fuzzy logic controller. At some point of time in Japan, the scientists designed fuzzy logic controller even for household appliances like a room heater or a washing machine. Its popularity is such that it has been applied to various engineering products.

Fuzzy number or fuzzy variable:

We are discussing the concept of a fuzzy number. Let us take three statements: zero, almost zero, near zero. Zero is exactly zero with truth value assigned 1. If it is almost 0, then I can think that between minus 1 to 1, the values around 0 is 0, because this is almost 0. I am not

very precise, but that is the way I use my day to day language in interpreting the real world. When I say near 0, maybe the bandwidth of the membership which represents actually the truth value. You can see that it is more, bandwidth increases near 0. This is the concept of fuzzy number. Without talking about membership now, but a notion is that I allow some small bandwidth when I say almost 0. When I say near 0 my bandwidth still further increases. In the case minus 2 to 2, when I encounter any data between minus 2 to 2, still I will consider them to be near 0. As I go away from 0 towards minus 2, the confidence level how near they are to 0 reduces; like if it is very near to 0, I am very certain. As I progressively go away from 0, the level of confidence also goes down, but still there is a tolerance limit. So when zero I am precise, I become imprecise when almost and I further become more imprecise in the third case.

When we say fuzzy logic, that is the variables that we encounter in physical devices, fuzzy numbers are used to describe these variables and using this methodology when a controller is designed, it is a fuzzy logic controller.

Neural networks :

Neural networks are basically inspired by various way of observing the biological organism. Most of the time, it is motivated from human way of learning. It is a learning theory. This is an artificial network that learns from example and because it is distributed in nature, fault tolerant, parallel processing of data and distributed structure.

The basic elements of artificial Neural Network are: input nodes, weights, activation function and output node. Inputs are associated with synaptic weights. They are all summed and passed through an activation function giving output y. In a way, output is summation of the signal multiplied with synaptic weight over many input channels.

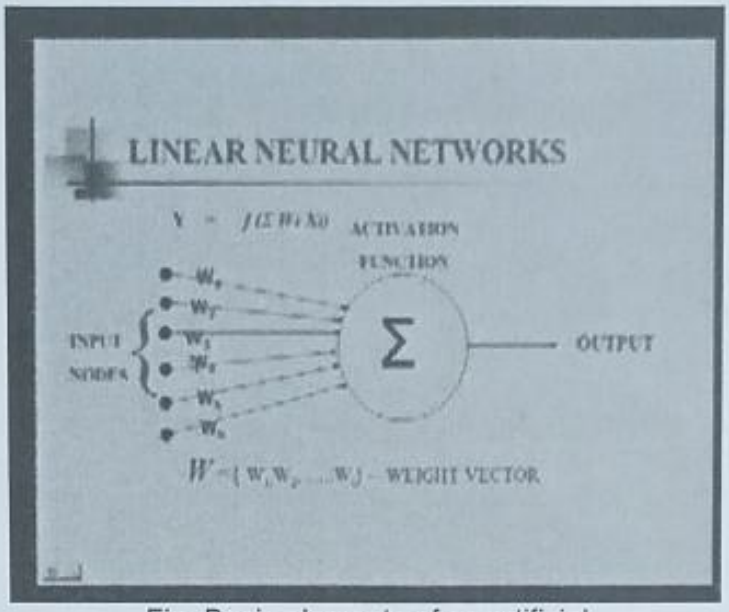


Fig. Basic elements of an artificial neuro

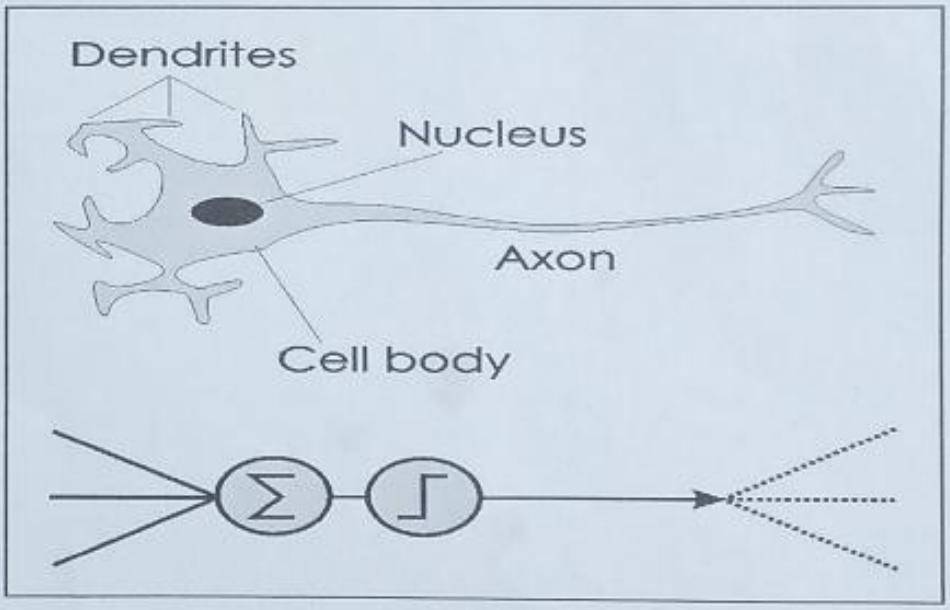


Fig. Analogy of biological neuron and artificial neuron

1.1 Introduction to Soft Computing

Two major problem solving techniques are:

- **Hard computing**

  It deals with precise model where accurate solutions are achieved.

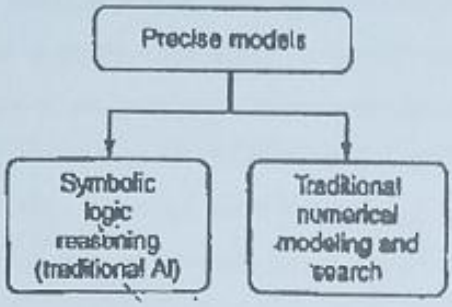


Figure 1.1: Hard Computing

- **Soft computing**

  It deals with approximate model to give solution for complex problems. The term "soft computing" was introduced by Professor Lorfi Zadeh with the objective of exploiting the tolerance for imprecision, uncertainty and partial truth to achieve tractability, robustness, low solution cost and better rapport with reality. The ultimate goal is to be able to emulate the human mind as closely as possible. It is a combination of Genetic Algorithm, Neural Network and Fuzzy Logic.

ANN
Fuzzy
Genetic

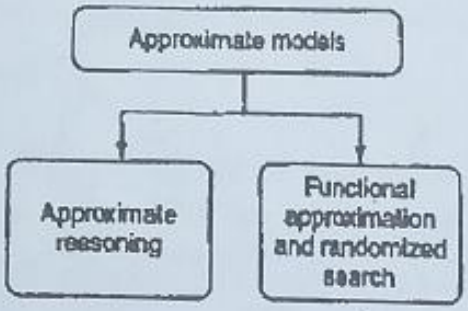


Figure 1.2: Soft Computing

1.2 Biological Neurons

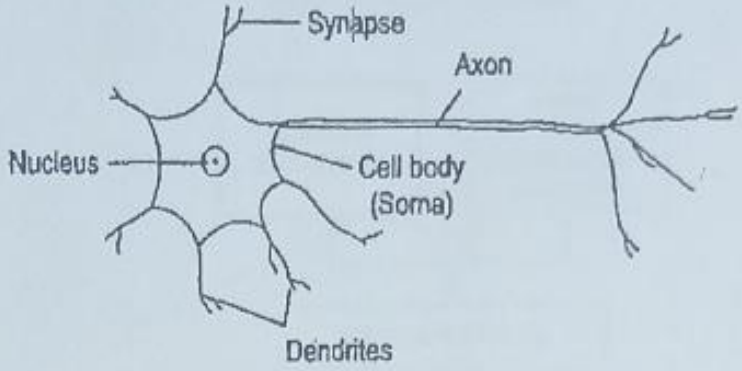


Figure 1.3: Schematic diagram of a biological neuron

The biological neuron consists of main three parts:

- **Soma or cell body**-where cell nucleus is located
- **Dendrites**-where the nerve is connected to the cell body
- **Axon**-which carries the impulses of the neuron

Dendrites are tree like networks made of nerve fiber connected to the cell body. An Axon is a single, long connection extending from the cell body and carrying signals from the neuron. The end of axon splits into fine strands. It is found that each strand terminated into small bulb like organs called as synapse. It is through synapse that the neuron introduces its signals to other nearby neurons. The receiving ends of these synapses on the nearby neurons can be found both on the dendrites and on the cell body. There are approximately 10^4 synapses per neuron in the human body. Electric impulse is passed between synapse and dendrites. It is a chemical process which results in increase/decrease in the electric potential inside the body of the receiving cell. If the electric potential reaches a thresh hold value, receiving cell fires & pulse / action potential of fixed strength and duration is send through the axon to synaptic junction of the cell. After that, cell has to wait for a period called refractory period.

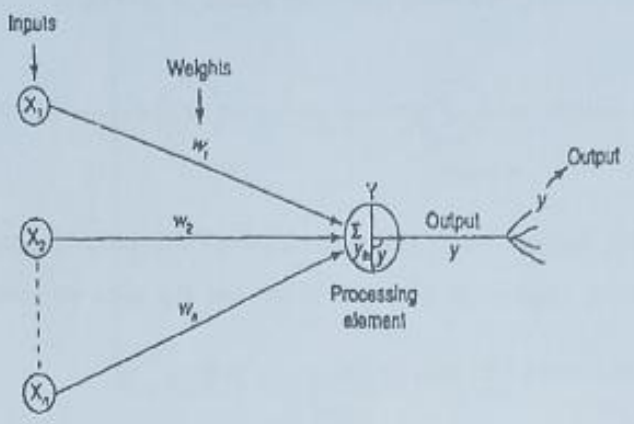


Figure 1.4: Mathematical model of artificial neuron

Biological neuron	Artificial neuron
Cell	Neuron
Dendrites	Weights or interconnections
Soma	Net input
Axon	Output

Table 1.1: Terminology relationships between biological and artificial neurons

In this model net input is calculated as

$$y_{in} = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n = \sum_{i=1}^{n} x_i w_i$$

Where, i representes i^{th} processing element. The activation function applied over it to calculate the output. The weight represents the strength of synapses connecting the input and output.

1.3 Artificial neural networks

An artificial neural network (ANN) is an efficient information processing system which resembles the characteristics of biological neural network. ANNs contain large number of highly interconnected processing elements called nodes or neurons or units. Each neuron is connected with other by connection link and each connection link is associated with weights which contain information about the input signal. This information is used by neuron net to solve a particular problem. ANNs have ability to learn, recall and generalize training pattern or

data similar to that of human brain. The ANN processing elements called neurons or artificial neurons.

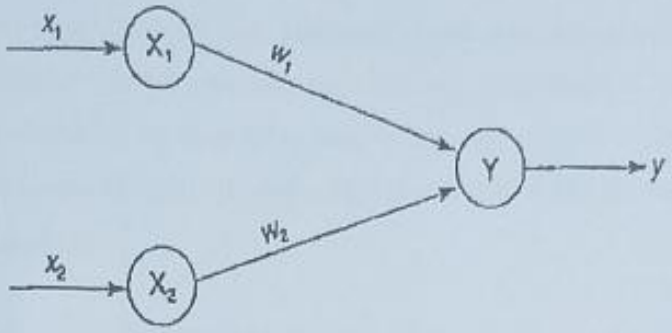


Figure 1.5: Architecture of a simple artificial neuron net

Each neuron has an internal state of its own, called activation or activity level of neuron which is the function of the inputs the neuron receives. The activation signal of a neuron is transmitted to other neurons. A neuron can send only one signal at a time which can be transmitted to several neurons.

Consider the figure 1.5, here X_1 and X_2 are input neurons, Y is the output neuron W_1 and W_2 are the weights net input is calculated as

$$y_{in} = x_1 w_1 + x_2 w_2$$

where x_1 and x_2 are the activation of the input neurons X_1 and X_2, i.e., is the output of the input signals. The output y of the output neuron Y can be obtained by applying activations over the net input.

$$y = f(y_{in})$$

Output = Function (net input calculated)

The function to be applied over the net input is called activation function. The net input calculation is similar to the calculation of output of a pure linear straight line equation y=mx

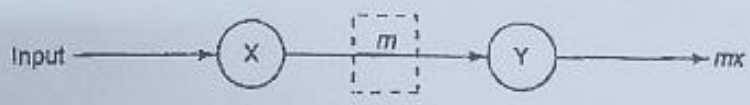


Figure 1.6: Neural net of pure linear equation

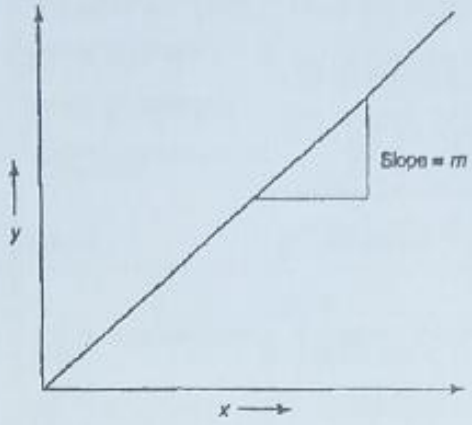


Figure 1.7: Graph for $y = mx$

The weight involve in the ANN is equivalent to the slope of the straight line.

1.4 Comparison between Biological neuron and Artificial neuron

Term	Brain	Computer
Speed	Execution time is few milliseconds	Execution time is few nano seconds
Processing	Perform massive parallel operations simultaneously	Perform several parallel operations simultaneously. It is faster the biological neuron
Size and complexity	Number of Neuron is 10^{11} and number of interconnections is 10^{15}. So complexity of brain is higher than computer	It depends on the chosen application and network designer.
Storage capacity	• Information is stored in interconnections or in synapse strength. • New information is stored without destroying old one. • Sometimes fails to recollect information	• Stored in continuous memory location. • Overloading may destroy older locations. • Can be easily retrieved

Tolerance	• Fault tolerant • Store and retrieve information even interconnections fails • Accept redundancies	• No fault tolerance • Information corrupted if the network connections disconnected. • No redundancies
Control mechanism	Depends on active chemicals and neuron connections are strong or weak	CPU Control mechanism is very simple

Table 1.2: Comparison between Biological neuron and Artificial neuron

Characteristics of ANN:

- It is a neurally implemented mathematical model
- Large number of processing elements called neurons exists here.
- Interconnections with weighted linkage hold informative knowledge.
- Input signals arrive at processing elements through connections and connecting weights.
- Processing elements can learn, recall and generalize from the given data.
- Computational power is determined by the collective behavior of neurons.
    - ANN is a connection models, parallel distributed processing models, self-organizing systems, neuro-computing systems and neuro - morphic system.

1.5 Evolution of neural networks

Year	Neural network	Designer	Description
1943	McCulloch and Pitts neuron	McCulloch and Pitts	Arrangement of neurons is combination of logic gate. Unique feature is thresh hold
1949	Hebb network	Hebb	If two neurons are active, then their connection strengths should be increased.
1958, 1959, 1962, 1988,	Perceptron	Frank Rosenblatt, Block, Minsky and Papert	Here the weights on the connection path can be adjusted.

1960	Adaline	Widrow and Hoff	Here the weights are adjusted to reduce the difference between the net input to the output unit and the desired output.
1972	Kohonen self-organizing feature map	Kohonen	Inputs are clustered to obtain a fired output neuron.
1982, 1984, 1985, 1986, 1987	Hopfield network	John Hopfield and Tank	Based on fixed weights. Can act as associative memory nets
1986	Back propagation network	Rumelhart, Hinton and Williams	• Multilayered • Error propagated backward from output to the hidden units
1988	Counter propagation network	Grossberg	Similar to kohonen network.
1987-1990	Adaptive resonance Theory(ART)	Carpenter and Grossberg	Designed for binary and analog inputs.
1988	Radial basis function network	Broomhead and Lowe	Resemble back propagation network, but activation function used is Gaussian function.
1988	Neo cognitron	Fukushima	For character recognition.

Table 1.3: Evolution of neural networks

1.6 Basic models of artificial neural networks

Models are based on three entities

- The model's synaptic interconnections.
- The training or learning rules adopted for updating and adjusting the connection weights.
- Their activation functions

1.6.1 Connections

The arrangement of neurons to form layers and the connection pattern formed within and between layers is called the network architecture. There exist five basic types of connection architecture.

They are:

1. Single layer feed forward network
2. Multilayer feed-forward network
3. Single node with its own feedback
4. Single-layer recurrent network
5. Multilayer recurrent network

Feed forward network: If no neuron in the output layer is an input to a node in the same layer / proceeding layer.

Feedback network: If outputs are directed back as input to the processing elements in the same layer/proceeding layer.

Lateral feedback: If the output is directed back to the input of the same layer.

Recurrent networks: Are networks with feedback networks with closed loop.

1. **Single layer feed forward network**

   Layer is formed by taking processing elements and combining it with other processing elements. Input and output are linked with each other Inputs are connected to the processing nodes with various weights, resulting in series of outputs one per node.

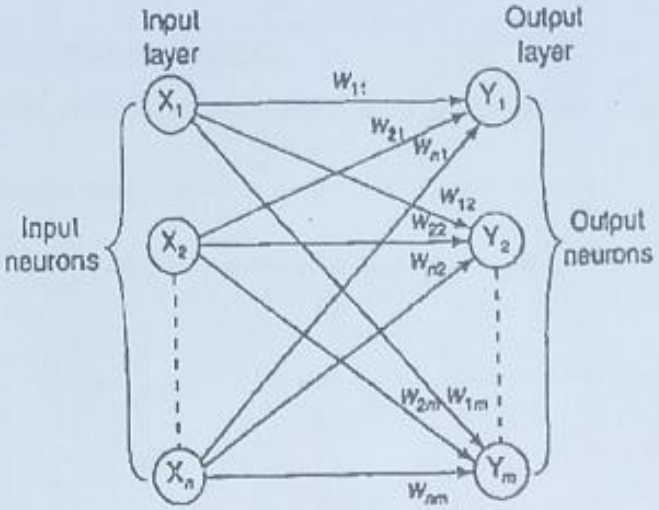


Figure 1.8: Single-layer feed-forward network

When a layer of processing nodes is formed the inputs can be connected to these nodes with various weights, resulting in a serious of outputs, one per node. This is called single layer feedforward network.

2. Multilayer feed-forward network

This network is formed by the interconnection of several layers. Input layer receives input and buffers input signal. Output layer generated output. Layer between input and output is called hidden layer. Hidden layer is internal to the network. There are Zero to several hidden layers in a network. More the hidden layer more is the complexity of network, but efficient output is produced.

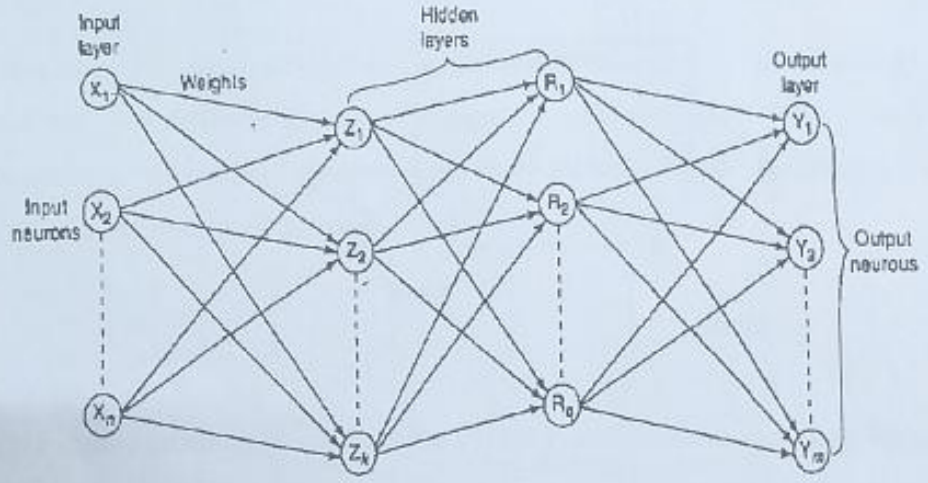


Figure 1.9: Multilayer feed-forward network

3. Single node with its own feedback

It is a simple recurrent neural network having a single neuron with feedback to itself.

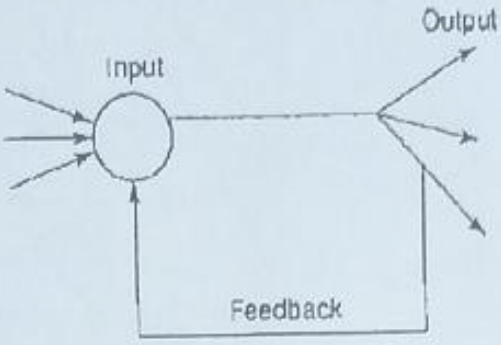


Figure 1.10: Single node with own feedback

4. Single layer recurrent network

A single layer network with feedback from output can be directed to processing element itself or to other processing element/both.

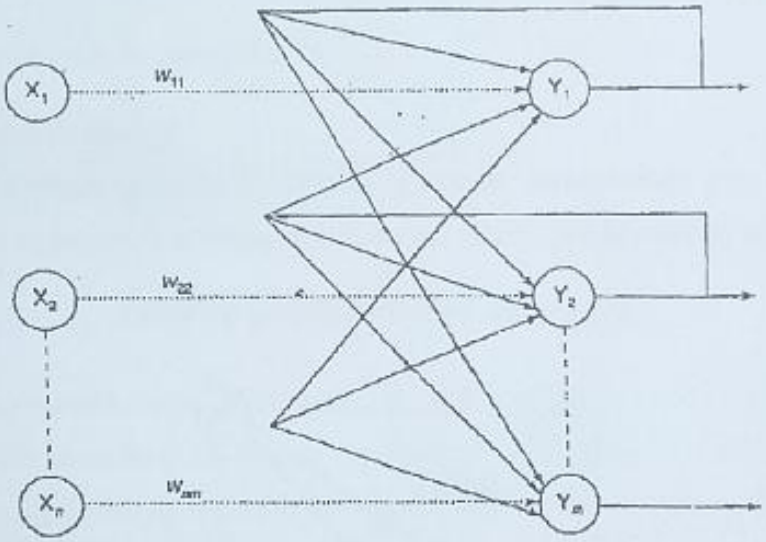


Figure 1.11: Single-layer recurrent network

5. Multilayer recurrent network

Processing element output can be directed back to the nodes in the preceding layer, forming a multilayer recurrent network.

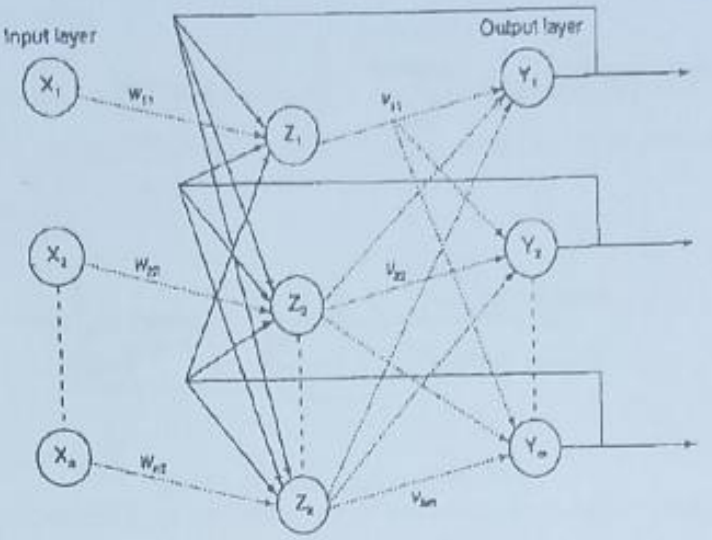


Figure 1.12: Multilayer recurrent network

- **Maxnet** –competitive interconnections having fixed weights.

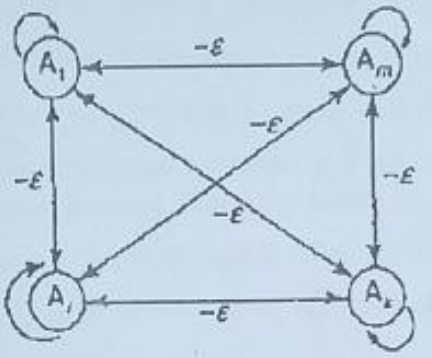


Figure 1.13: Competitive nets

- **On-center-off-surround/lateral inhibition structure** – each processing neuron receives two different classes of inputs- "excitatory" input from nearby processing elements & "inhibitory" elements from more distantly located processing elements. This type of interconnection is shown below

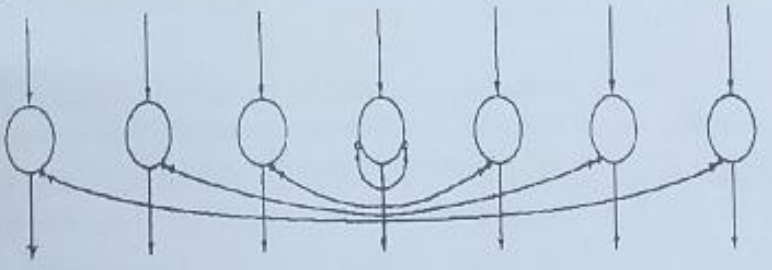


Figure 1.14: Lateral inhibition structure

1.6.2 Learning

Learning or Training is the process by means of which a neural network adapts itself to a stimulus by making proper parameter adjustments, resulting in the production of desired response.

Two broad kinds of learning in ANNs is:

i) Parameter learning – updates connecting weights in a neural net.

ii) Structure learning – focus on change in the network.

Apart from these, learning in ANN is classified into three categories as

i) Supervised learning

ii) Unsupervised learning

iii) Reinforcement learning

i) Supervised learning

The Learning here is performed with the help of a teacher. Example: Consider the learning process of a small child. Child doesn't know how to read/write. Their each and every action is supervised by a teacher. Actually a child works on the basis of the output that he/she has to produce. In ANN, each input vector requires a corresponding target vector, which represents the desired output. The input vector along with target vector is called training pair. Input vector results in output vector. The actual output vector is compared with desired output vector. If there is a difference means an error signal is generated by the network. It is used for adjustment of weights until actual output matches desired output.

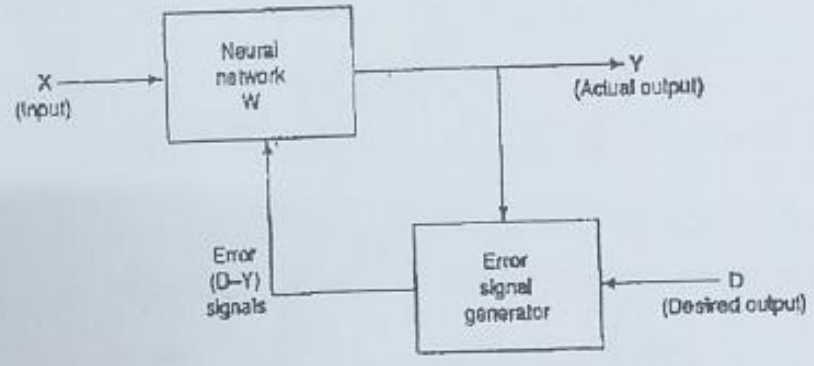


Figure 1.15: Supervised learning

ii) Unsupervised learning

Learning is performed without the help of a teacher. Example: tadpole – learn to swim by itself. In ANN, during training process, network receives input patterns and organize it to form clusters.

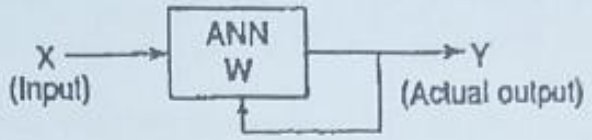


Figure 1.16: Unsupervised learning

From the above Fig.1.16 it is observed that no feedback is applied from environment to inform what output should be or whether they are correct. The network itself discover patterns, regularities, features/ categories from the input data and relations for the input data over the output. Exact clusters are formed by discovering similarities & dissimilarities so called as self – organizing.

iii) Reinforcement learning

It is similar to supervised learning. Learning based on critic information is called reinforcement learning & the feedback sent is called reinforcement signal. The network receives some feedback from the environment. Feedback is only evaluative.

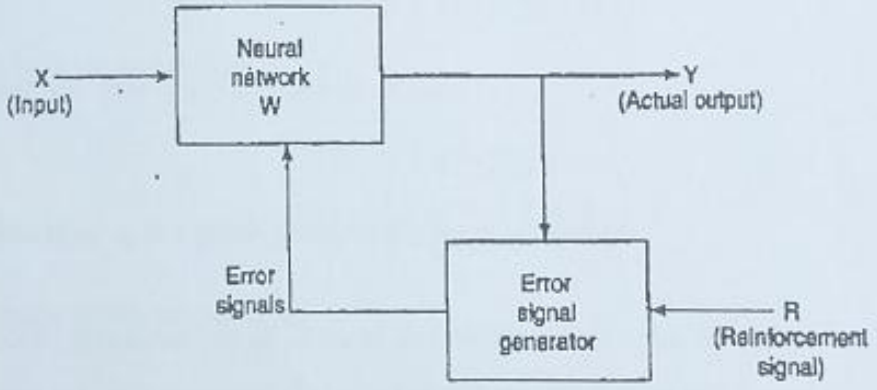


Figure 1.17: Reinforcement learning

The external reinforcement signals are processed in the critic signal generator, and the obtained critic signals are sent to the ANN for adjustment of weights properly to get critic feedback in future.

1.6.3 Activation Functions

To make work more efficient and for exact output, some force or activation is given. Like that, activation function is applied over the net input to calculate the output of an ANN. Information processing of processing element has two major parts: input and output. An integration function (f) is associated with input of processing element.

Several activation functions are there.

1. **Identity function:** It is a linear function which is defined as

$$f(x) = x \text{ for all } x$$

The output is same as the input.

2. **Binary step function:** This function can be defined as

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

Where, θ represents thresh hold value. It is used in single layer nets to convert the net input to an output that is binary (0 or 1).

3. **Bipolar step function:** This function can be defined as

$$f(x) = \begin{cases} 1 \text{ if } x \geq \theta \\ -1 \text{ if } x < \theta \end{cases}$$

Where, θ represents threshold value. It is used in single layer nets to convert the net input to an output that is bipolar (+1 or -1).

4. **Sigmoid function:** It is used in Back propagation nets.

Two types:

a) **Binary sigmoid function:** It is also termed as logistic sigmoid function or unipolar sigmoid function. It is defined as

$$f(x) = \frac{1}{1 + e^{-\lambda x}}$$

where, λ represents steepness parameter. The derivative of this function is

$$f'(x) = \lambda f(x)[1 - f(x)]$$

The range of sigmoid function is 0 to 1.

b) Bipolar sigmoid function: This function is defined as

$$f(x) = \frac{2}{1 + e^{-\lambda x}} - 1 = \frac{1 - e^{-\lambda x}}{1 + e^{-\lambda x}}$$

Where λ represents steepness parameter and the sigmoid range is between -1 and +1. The derivative of this function can be

$$f'(x) = \frac{\lambda}{2}[1 + f(x)][1 - f(x)]$$

It is closely related to hyperbolic tangent function, which is written as

$$h(x) = = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$h(x) = = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

The derivative of the hyperbolic tangent function is

$$h'(x) = [1 + h(x)][1 - h(x)]$$

5. Ramp function: The ramp function is defined as

$$f(x) = \begin{cases} 1 \; if x > 1 \\ x \; if \; 0 \leq x \leq 1 \\ 0 \; if \; x < 0 \end{cases}$$

The graphical representation of all these function is given in the upcoming figure 1.18

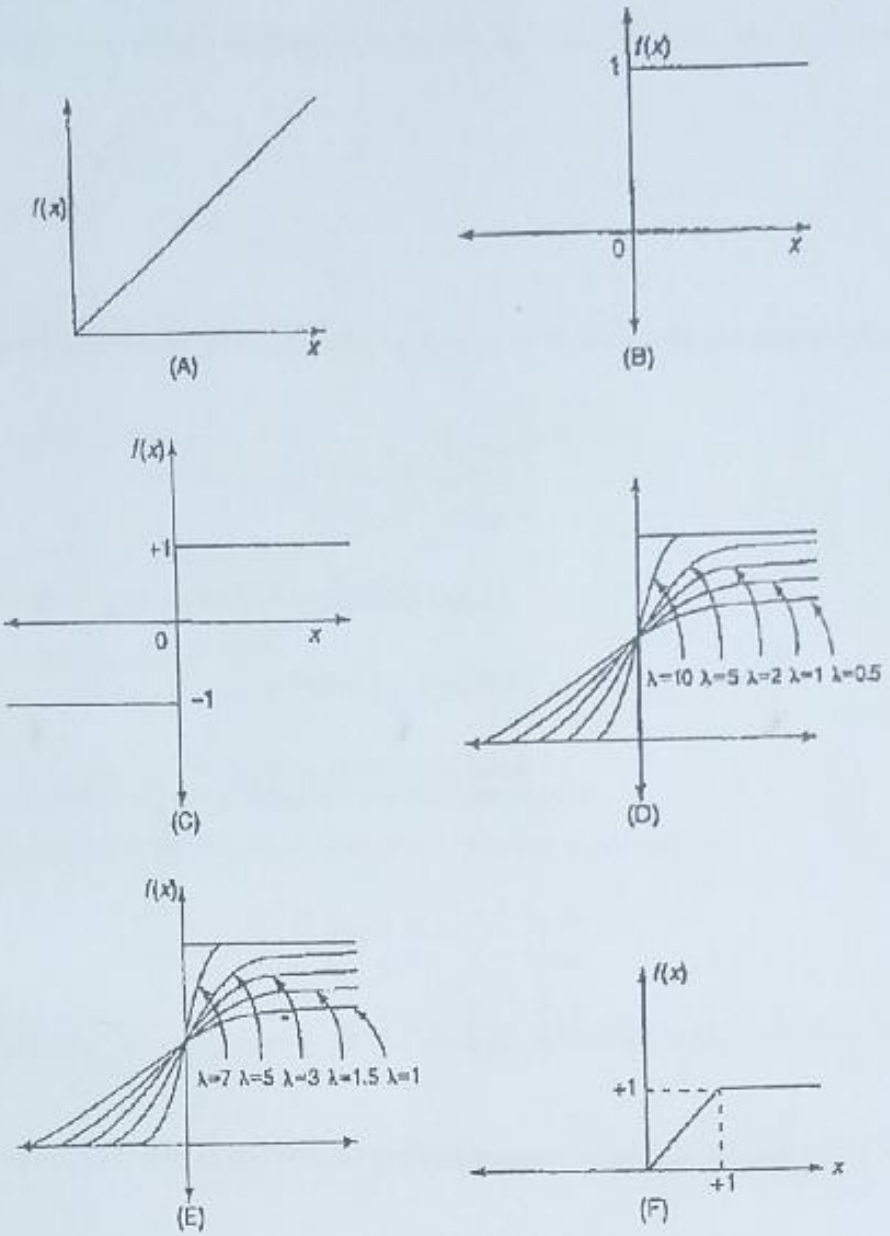


Figure 1.18: Depiction of activation functions: (A) identity function; (B) binary step function; (C) bipolar step function; (D) binary sigmoidal function; (E) bipolar sigmoidal function; (F) ramp function.

1.7 McCulloch and Pitts Neuron

It is discovered in 1943 and usually called as M-P neuron. M-P neurons are connected by directed weighted paths. Activation of M-P neurons is binary (i.e) at any time step the neuron may fire or may not fire. Weights associated with communication links may be excitatory (wgts are positive)/inhibitory (wgts are negative). Threshold plays major role here. There is a fixed threshold for each neuron and if the net input to the neuron is greater than the threshold then the neuron fires. They are widely used in logic functions. A simple M-P neuron is shown in the figure. It is excitatory with weight w (w>0) / inhibitory with weight –p (p<0). In the Fig.,

inputs from x_1 to x_n possess excitatory weighted connection and X_{n+1} to x_{n+m} has inhibitory weighted interconnections.

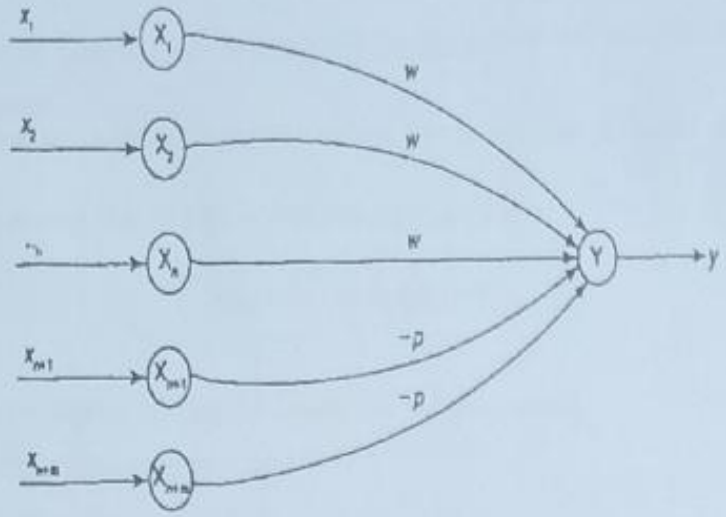


Figure 1.19: McCulloch-Pins neuron model

Since the firing of neuron is based on threshold, activation function is defined as

$$f(x) = \begin{cases} 1 & if\ y_{in} \geq \theta \\ 0 & if\ y_{in} < \theta \end{cases}$$

For inhibition to be absolute, the threshold with the activation function should satisfy the following condition:

$$\theta > nw - p$$

Output will fire if it receives "k" or more excitatory inputs but no inhibitory inputs where

$$kw \geq \theta > (k-1)\ w$$

The M-P neuron has no particular training algorithm. An analysis is performed to determine the weights and the threshold. It is used as a building block where any function or phenomenon is modeled based on a logic function.

1.8 Hebb network

Donald Hebb stated in 1949 that "In brain, the learning is performed by the change in the synaptic gap". When an axon of cell A is near enough to excite cell B, and repeatedly or permanently takes place in firing it, some growth process or metabolic change takes place in

one or both the cells such that A's efficiency, as one of the cells firing B, is increased. According to Hebb rule, the weight vector is found to increase proportionately to the product of the input and the learning signal. In Hebb learning, two interconnected neurons are 'on' simultaneously. The weight update in Hebb rule is given by

$$w_i(new) = w_i(old) + x_i y$$

Hebbs network is suited more for bipolar data. If binary data is used, the weight updation formula cannot distinguish two conditions namely:

1. A training pair in which an input unit is "on" and the target value is "off".
2. A training pair in which both the input unit and the target value is "off".

Training algorithm

The training algorithm is used for the calculation and adjustment of weights. The flowchart for the training algorithm of Hebb network is given below

Step 0: First initialize the weights. Basically in this network they may be set to zero, i.e., $w_i = 0$, for i= 1 to n where "n" may be the total number of input neurons.

Step 1: Steps 2-4 have to be performed for each input training vector and target output pair, s: t.

Step 2: Input units activations are set. Generally, the activation function of input layer is identity function: $x_i = s_i$ *for* $i=1$ *to* n

Step 3: Output units activations are set: $y = t$.

Step 4: Weight adjustments and bias adjustments are performed:

$$w_i(new) = w_i(old) + x_i y$$

$$b(new) = b(old) + y$$

In step 4, the weight updation formula can be written in vector form as

$$w(new) = w(old) + y$$

Hence, Change in weight is expressed as

$$\Delta w = xy$$

As a result,

$$w(new)=w(old)+\Delta w$$

Hebb rule is used for pattern association, pattern categorization, pattern classification and over a range of other areas.

Flowchart of Training algorithm

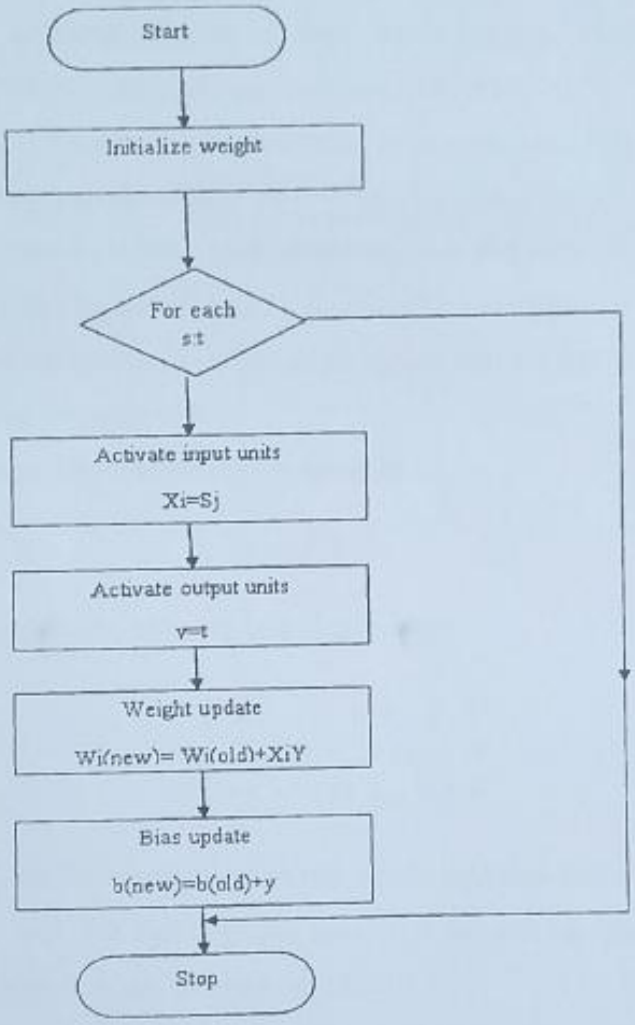


Figure 1.20: Flowchart of Hebb training algorithm

2.1 Perceptron networks

2.1.1 Theory

Perceptron networks come under single-layer feed-forward networks and are also called simple perceptrons. Various types of perceptrons were designed by Rosenblatt (1962) and Minsky-Papert (1969, 1988).

The key points to be noted in a perceptron network are:

1. The perceptron network consists of three units, namely, sensory unit (input unit), associator unit (hidden unit), and response unit (output unit).
2. The sensory units are connected to associator units with fixed weights having values 1, 0 or -l, which are assigned at random.
3. The binary activation function is used in sensory unit and associator unit.
4. The response unit has an activation of l, 0 or -1. The binary step with fixed threshold ɵ is used as activation for associator. The output signals that are sent from the associator unit to the response unit are only binary.
5. The output of the perceptron network is given by

$$y = f(y_{in})$$

where $f(y_{in})$ is activation function and is defined as

$$f(y_{in}) = \begin{cases} 1 & if\, y_{in} > \theta \\ 0 & if\, -\theta \le y_{in} \le \theta \\ -1 & if\, y_{in} < -\theta \end{cases}$$

6. The perceptron learning rule is used in the weight updation between the associator unit and the response unit. For each training input, the net will calculate the response and it will determine whether or not an error has occurred.
7. The error calculation is based on the comparison of the values of targets with those of the calculated outputs.
8. The weights on the connections from the units that send the nonzero signal will get adjusted suitably.
9. The weights will be adjusted on the basis of the learning rule an error has occurred for a particular training patterns .i.e..,

$$w_i(new) = w_i(old) + \alpha\, t x_i$$

$$b(new) = b(old) + \alpha\, t$$

If no error occurs, there is no weight updation and hence the training process may be stopped. In the above equations, the target value "t" is +1 or-l and α is the learning rate. In general, these learning rules begin with an initial guess at the weight values and then successive adjustments are made on the basis of the evaluation of an objective function. Eventually, the learning rules reach a near optimal or optimal solution in a finite number of steps.

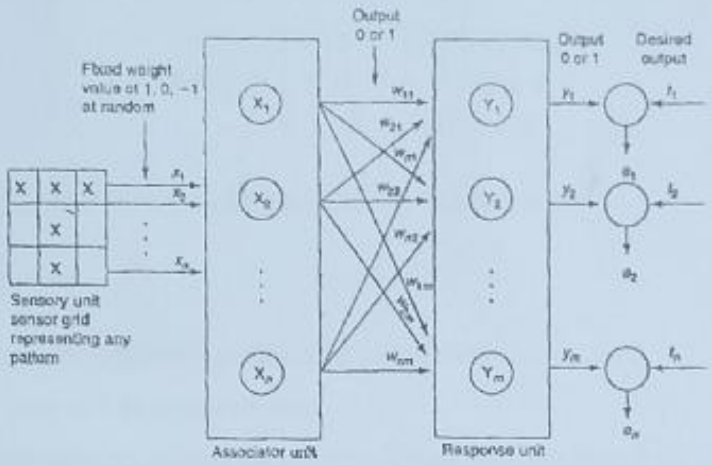


Figure 2.1: Original perceptron network

A Perceptron network with its three units is shown in above figure. The sensory unit can be a two-dimensional matrix of 400 photodetectors upon which a lighted picture with geometric black and white pattern impinges. These detectors provide a binary (0) electrical signal if the input signal is found to exceed a certain value of threshold. Also, these detectors are connected randomly with the associator unit. The associator unit is found to consist of a set of subcircuits called feature predicates. The feature predicates are hardwired to detect the specific feature of a pattern and are equivalent to the feature detectors. For a particular feature, each predicate is examined with a few or all of the responses of the sensory unit. It can be found that the results from the predicate units are also binary (0 or 1). The last unit, i.e. response unit, contains the pattern recognizers or perceptrons. The weights present in the input layers are all fixed, while the weights on the response unit are trainable.

2.1.2 Perceptron Learning Rule

Learning signal is the difference between desired and actual response of a neuron. The perceptron learning rule is explained as follows:

Consider a finite "n" number of input training vectors, with their associated target (desired) values x(n) and t(n), where "n" ranges from 1 to N. The target is either +1 or -1. The output "y" is obtained on the basis of the net input calculated and activation function being applied over the net input.

$$y = f(y_{in}) = \begin{cases} 1 & if\, y_{in} > \theta \\ 0 & if\, -\theta \le y_{in} \le \theta \\ -1 & if\, y_{in} < -\theta \end{cases}$$

The weight updation in case of perceptron learning is as shown.

If $y \neq t$, then

$$w(new) = w(old) + \alpha\, tx\ (\alpha - learning\ rate)$$

else,

$$w(new) = w(old)$$

The weights can be initialized at any values in this method. The perceptron rule convergence theorem states that " If there is a weight vector W such that $f(x(n)W) = t(n)$, for all n then for any starting vector w_1, the perceptron learning rule will convergence to a weight vector that gives the correct response for all training patterns, and this learning takes place within a finite number of steps provided that the solution exists".

2.1.3 Architecture

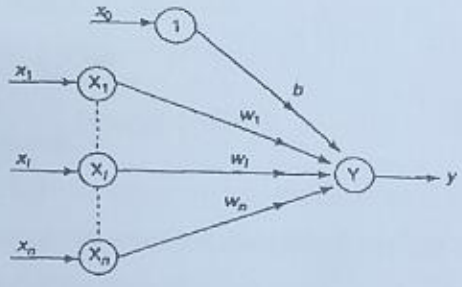


Figure 2.2: Single classification perceptron network

Here only the weights between the associator unit and the output unit can be adjusted, and the weights between the sensory and associator units are fixed.

2.1.4 Flowchart for Training Process

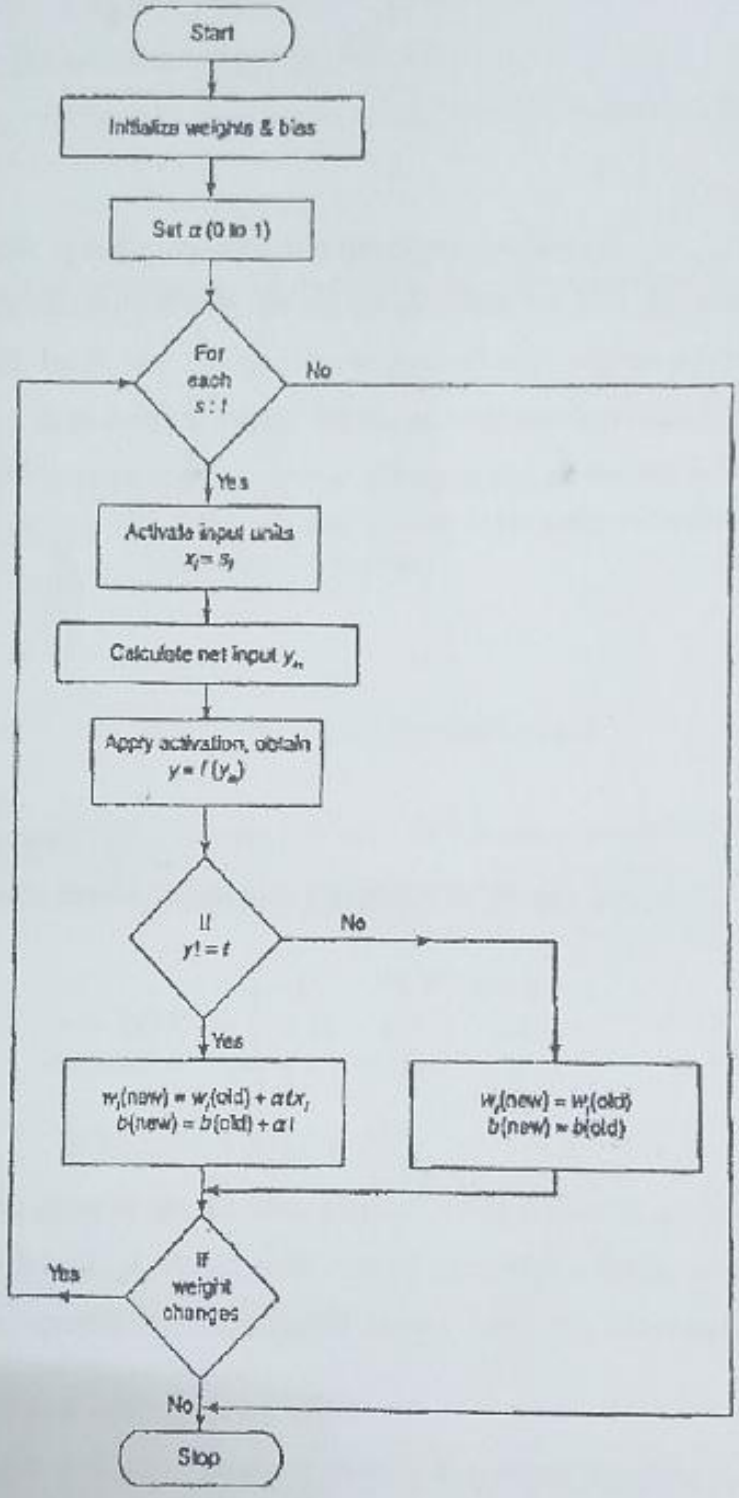


Figure 2.3: Flowchart for perceptron network with single output

2.1.5 Perceptron Training Algorithm for Single Output Classes

Step 0: Initialize the weights and the bias (for easy calculation they can be set to zero). Also initialize the learning rate α ($0 < \alpha \leq 1$). For simplicity α is set to 1.

Step 1: Perform Steps 2-6 until the final stopping condition is false.

Step 2: Perform Steps 3-5 for each training pair indicated by s:t.

Step 3: The input layer containing input units is applied with identity activation functions:

$$x_i = s_i$$

Step 4: Calculate the output of the network. To do so, first obtain the net input:

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

Where "n" is the number of input neurons in the input layer. Then apply activations over the net input calculated to obtain the output:

$$y = f(y_{in}) = \begin{cases} 1 & if\, y_{in} > \theta \\ 0 & if\, -\theta \leq y_{in} \leq \theta \\ -1 & if\, y_{in} < -\theta \end{cases}$$

Step 5: Weight and bias adjustment: Compare the value of the actual (calculated) output and desired (target) output.

If $y \neq t$, then

$$w_i(new) = w_i(old) + \alpha\, t x_i$$

$$b(new) = b(old) + \alpha\, t$$

else,

$$w_i(new) = w_i(old)$$

$$b(new) = b(old)$$

Step 6: Train the network until there is no weight change. This is the stopping condition for the network. If this condition is not met, then start again from Step 2.

2.1.6 Perceptron Network Testing Algorithm

Step 0: The initial weights to be used here are taken from the training algorithms (the final weights obtained during training).

Step 1: For each input vector X to be classified, perform Steps 2-3.

Step 2: Set activations of the input unit.

Step 3: Obtain the response of output unit.

$$y_{in} = \sum_{i=1}^{n} x_i w_i$$

$$y = f(y_{in}) = \begin{cases} 1 & if\, y_{in} > \theta \\ 0 & if\, -\theta \le y_{in} \le \theta \\ -1 & if\, y_{in} < -\theta \end{cases}$$

Thus, the testing algorithm tests the performance of network. In the case of perceptron network, it can be used for linear separability. Here the separating line may be based on the value of threshold that is, the threshold used in the activation function must be a non negative value.

The condition for separating the response from the region of positive to region of zero is

$$w_1 x_1 + w_2 x_2 + b > \theta$$

The condition for separating the response from the region of zero to region of negative is

$$w_1 x_1 + w_2 x_2 + b < -\theta$$

The conditions above are stated for a single layer perceptron network with two input neurons and one output neuron and one bias.

2.2 Adaptive Linear Neuron

2.2.1 Theory

The units with linear activation function are called linear units. A network with a single linear unit is called an Adaline (adaptive linear neuron). That is, in an Adaline, the input-output relationship is linear. Adaline uses bipolar activation for its input signals and its target output. The weights between the input and the output are adjustable. The bias in

Adaline acts like an adjustable weight, whose connection is from a unit with activations being always 1. Adaline is a net which has only one output unit. The Adaline network may be trained using delta rule. The delta rule may also be called as least mean square (LMS) rule or Widrow-Hoff rule. This learning rule is found to minimize the mean squared error between the activation and the target value.

2.2.2 Delta Rule for Single Output Unit

The perceptron learning rule originates from the Hebbian assumption while the delta rule is derived from the gradient descendent method (it can be generalized to more than one layer). Also, the perceptron learning rule stops after a finite number of leaning steps, but the gradient-descent approach continues forever, converging only asymptotically to the solution. The delta rule updates the weights between the connections so as to minimize the difference between the net input to the output unit and the target value. The major aim is to minimize the error over all training patterns. This is done by reducing the error for each pattern, one at a time.

The delta rule for adjusting the weight of *i* th pattern (i = 1 to n) is

$$\Delta w_i = \alpha(t - y_{in})x_i$$

Where Δw_i is the weight change; α the learning rate; x the vector of activation of input unit; y_{in} the net input to output unit, i.e., $y_{in} = \sum_{i=1}^{n} x_i w_i$; t the target output. The delta rule in case of several output units for adjusting the weight from *i*th input unit to the *j*th output unit (for each pattern) is

$$\Delta w_{ij} = \alpha(t_j - y_{inj})x_i$$

2.2.3 Architecture

Adaline is a single unit neuron, which receives input from several units and also from one unit called bias. The basic Adaline model consists of trainable weights. Inputs are either of the two values (+ 1 or -1) and the weights have signs (positive or negative). Initially, random weights are assigned. The net input calculated is applied to a quantizer transfer function (possibly activation function) that restores the output to + 1 or -1. The Adaline

model compares the actual output with the target output and on the basis of the training algorithm, the weights are adjusted.

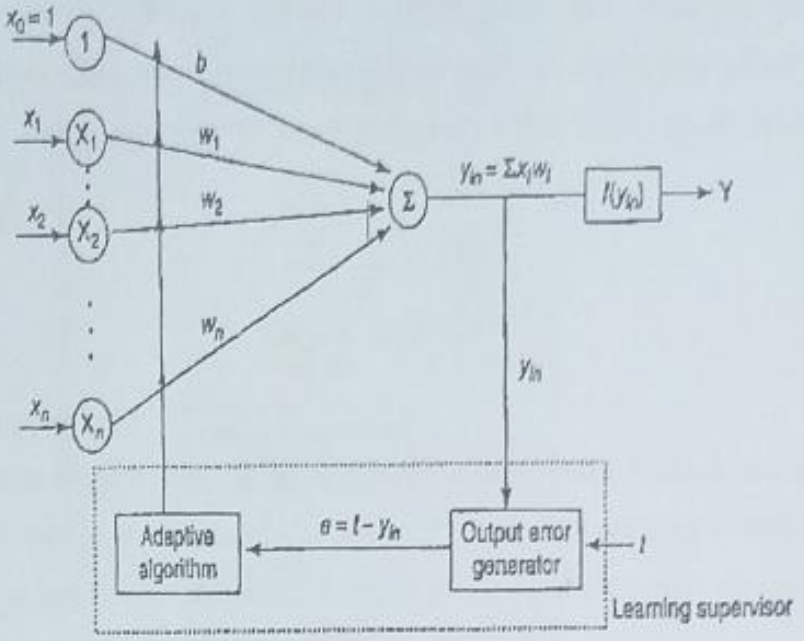


Figure 2.4: Adaline model

2.2.4 Flowchart for Training Process

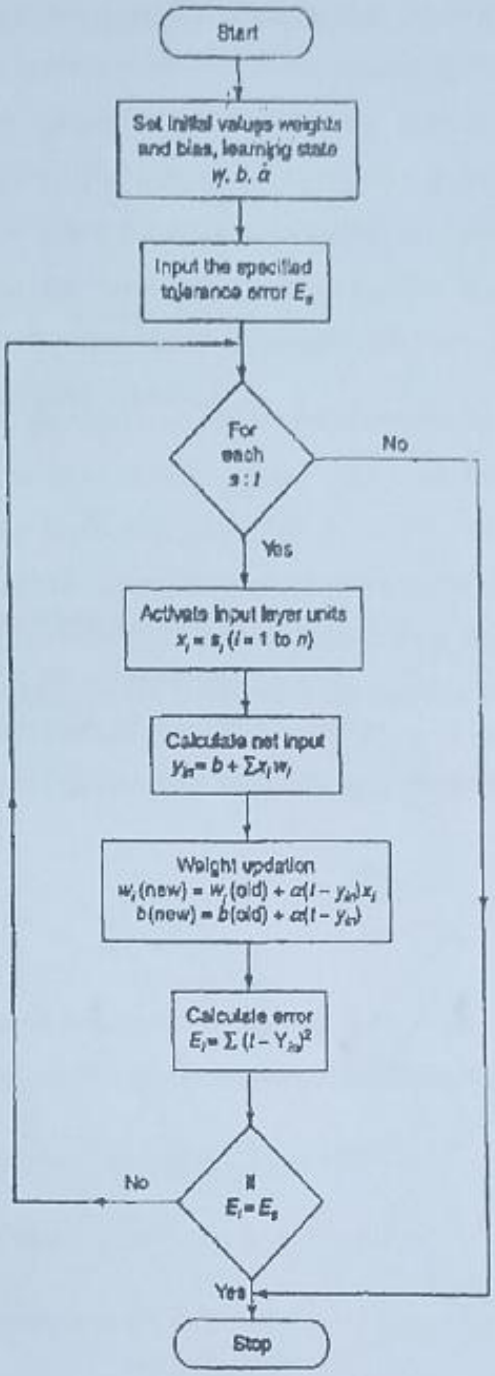


Figure 2.5: Flowchart for Adaline training process

2.2.5 Training Algorithm

Step 0: Weights and bias are set to some random values but not zero. Set the learning rate parameter α.

Step 1: Perform Steps 2-6 when stopping condition is false.

Step 2: Perform Steps 3-5 for each bipolar training pair *s:t*.

Step 3: Set activations for input units $i = 1$ to n.

$$x_i = s_i$$

Step 4: Calculate the net input to the output unit.

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

Step 5: Update the weights and bias for i= 1 to n

$$w_i(new) = w_i(old) + \alpha(t - y_{in})x_i$$

$$b(new) = b(old) + \alpha(t - y_{in})$$

Step 6: If the highest weight change that occurred during training is smaller than a specified tolerance then stop the training process, else continue. This is the rest for stopping condition of a network.

2.2.6 Testing Algorithm

Step 0: Initialize the weights. (The weights are obtained from the training algorithm.)

Step 1: Perform Steps 2-4 for each bipolar input vector x.

Step 2: Set the activations of the input units to x.

Step 3: Calculate the net input to the output unit:

$$y_{in} = b + \sum x_i w_i$$

Step 4: Apply the activation function over the net input calculated:

$$y = \begin{cases} 1 & if\ y_{in} \geq 0 \\ -1\ if\ y_{in} < 0 \end{cases}$$

2.3 Back propagation Network

2.3.1 Theory

The back propagation learning algorithm is one of the most important developments in neural networks (Bryson and Ho, 1969; Werbos, 1974; Lecun, 1985; Parker, 1985;

Rumelhart, 1986). This learning algorithm is applied to multilayer feed-forward networks consisting of processing elements with continuous differentiable activation functions. The networks associated with back-propagation learning algorithm are also called back-propagation networks. (BPNs). For a given set of training input-output pair, this algorithm provides a procedure for changing the weights in a BPN to classify the given input patterns correctly. The basic concept for this weight update algorithm is simply the gradient descent method. This is a methods were error is propagated back to the hidden unit. Back propagation network is a training algorithm.

The training of the BPN is done in three stages - the feed-forward of the input training pattern, the calculation and back-propagation of the error, and updation of weights. The testing of the BPN involves the computation of feed-forward phase only. There can be more than one hidden layer (more beneficial) but one hidden layer is sufficient. Even though the training is very slow, once the network is trained it can produce its outputs very rapidly.

2.3.2 Architecture

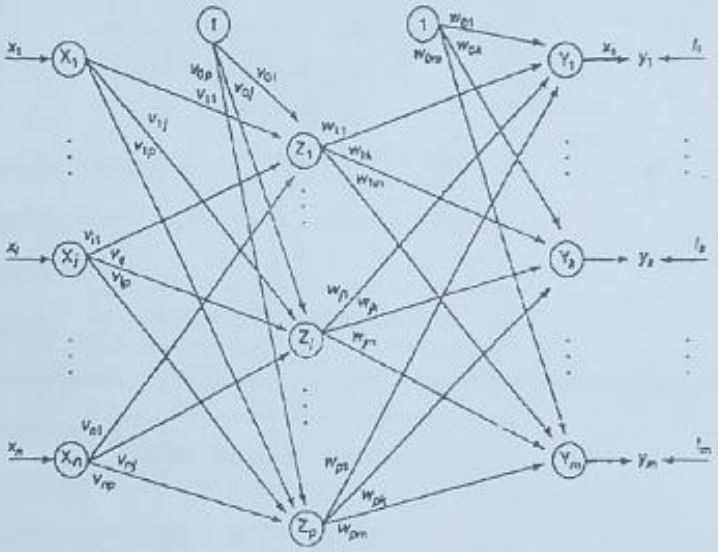


Figure 2.6: Architecture of a back propagation network

A back-propagation neural network is a multilayer, feed-forward neural network consisting of an input layer, a hidden layer and an output layer. The neurons present in the hidden and output layers have biases, which are the connections from the units whose activation is always 1. The bias terms also acts as weights. During the back propagation phase of learning, signals are sent in the reverse direction. The inputs sent to the BPN and the output

obtained from the net could be either binary (0, 1) or bipolar (-1, +1). The activation function could be any function which increases monotonically and is also differentiable.

2.3.3 Flowchart

The terminologies used in the flowchart and in the training algorithm are as follows:

x = input training vector $(x_1,\ldots,x_i,\ldots x_n)$

t = target output vector $(t_1,\ldots,t_k,\ldots t_m)$

α= learning rate parameter

x_i = input unit i. (Since the input layer uses identity activation function, the input and output signals here are same.)

v_{0j} = bias on jth hidden unit

w_{0k} = bias on kth output unit

z_j=hidden unit j. The net input to z_j is

$$z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$$

and the output is

$$z_j = f(z_{inj})$$

y_k = output unit k. The net input to y_k is

$$y_{ink} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$$

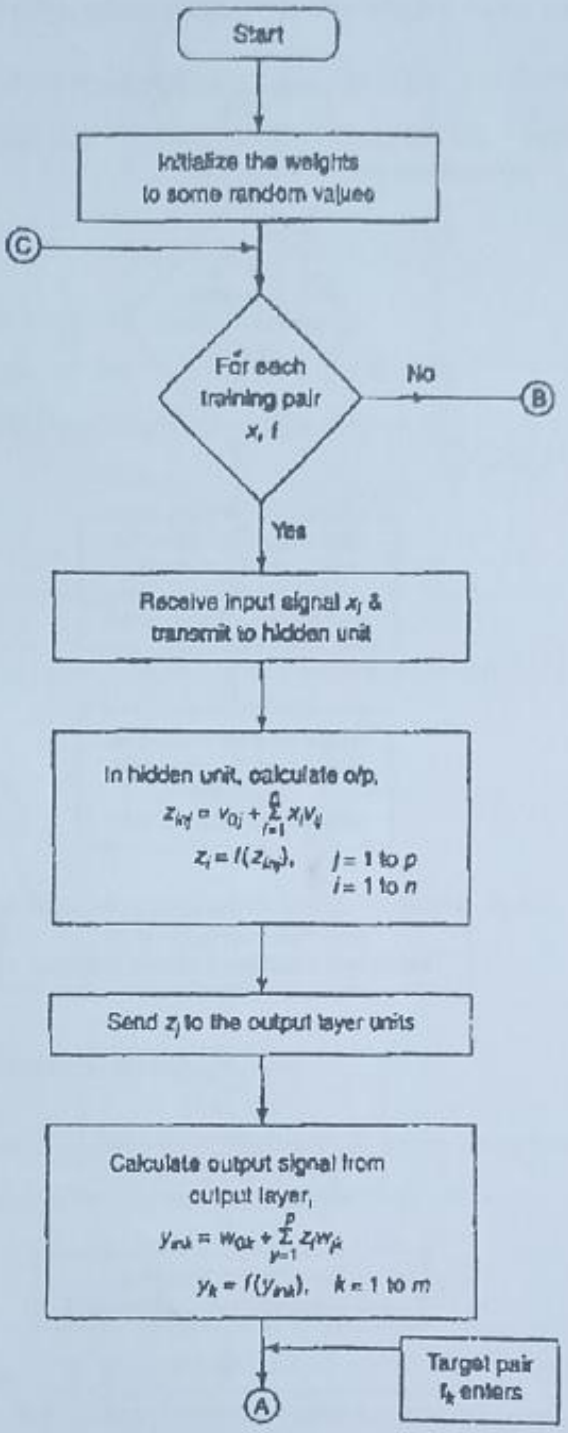
and the output is

$$y_k = f(y_{ink})$$

δ_k = error correction weight adjustment for w_{jk} that is due to an error in unit y_k, which is back-propagated to the hidden units that feed into unit y_k

δ_j = error correction weight adjustment for v_{ij} that is due to the back-propagation of error to the hidden unit is z_j.

The commonly used activation functions are binary, sigmoidal and bipolar sigmoidal activation functions. These functions are used in the BPN because of the following characteristics: (i) Continuity (ii) Differentiability iii) Non decreasing monotonic.



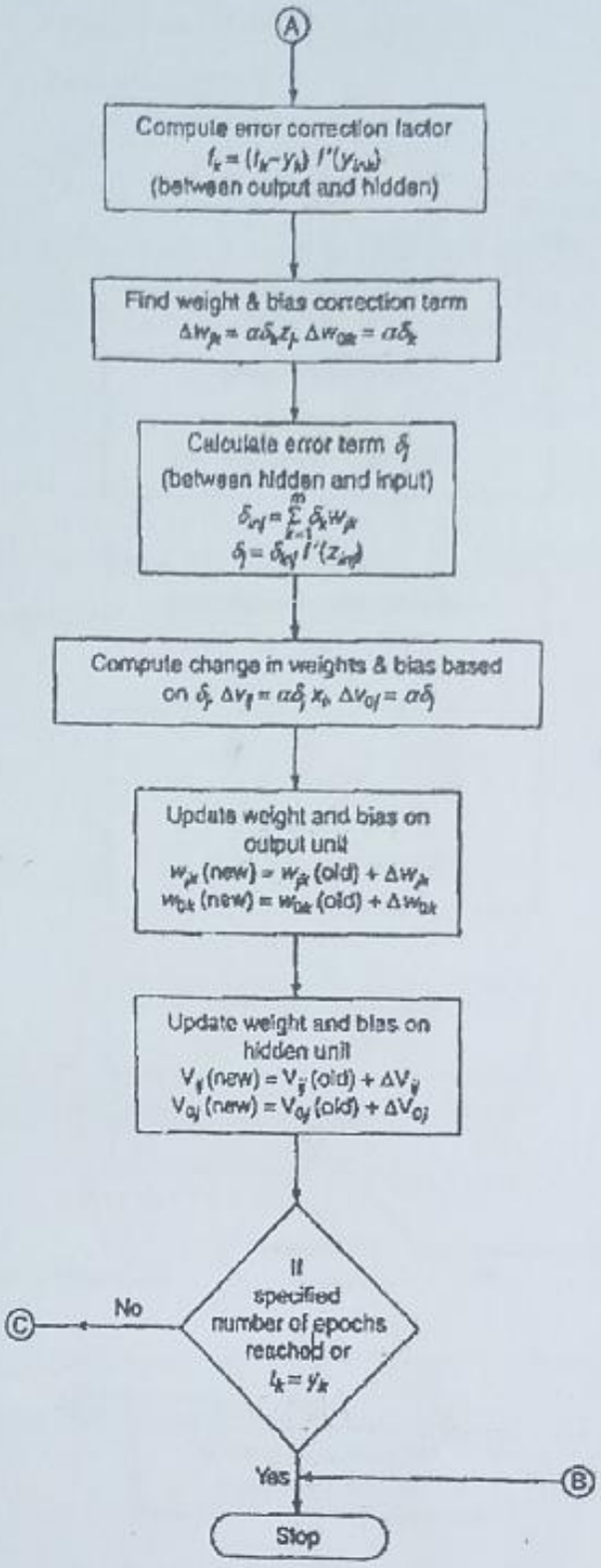


Figure 2.7: Flowchart for back - propagation network training

2.3.4 Training Algorithm

Step 0: Initialize weights and learning rate (take some small random values).

Step 1: Perform Steps 2-9 when stopping condition is false.

Step 2: Perform Steps 3-8 for each training pair.

Feedforward Phase 1

Step 3: Each input unit receives input signal x_i and sends it to the hidden unit (i = 1 to n).

Step 4: Each hidden unit z_j (j = 1 to p) sums its weighted input signals to calculate net input:

$$z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$$

Calculate output of the hidden unit by applying its activation functions over z_{inj} (binary or bipolar sigmoidal activation function):

$$z_j = f(z_{inj})$$

and send the output signal from the hidden unit to the input of output layer units.

Step 5: For each output unit y_k (k = 1 to m), calculate the net input:

$$y_{ink} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$$

and apply the activation function to compute output signal

$$y_k = f(y_{ink})$$

Back-propagation of error (Phase II)

Step 6: Each output unit y_k(k=1 to m) receives a target pattern corresponding to the input training pattern and computes the error correction term:

$$\delta_k = (t_k - y_k) f'(y_{ink})$$

The derivative $f'(y_{ink})$ can be calculated as in activation function section. On the basis of the calculated error correction term, update the change in weights and bias:

$$\Delta w_{jk} = \alpha \delta_k z_j; \quad \Delta w_{0k} = \alpha \delta_k$$

Also, send δ_k to the hidden layer backwards.

Step 7: Each hidden unit (z_j = 1 to p) sums its delta inputs from the output units:

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$$

The term δ_{inj} gets multiplied with the derivative of $f(z_{inj})$ to calculate the error term:

$$\delta_j = \delta_{inj} f'(z_{inj})$$

The derivative $f'(z_{inj})$ can be calculated as activation function section depending on whether binary or bipolar sigmoidal function is used. On the basis of the calculated δ_j, update the change in weights and bias:

$$\Delta v_{ij} = \alpha \delta_j x_i; \quad \Delta v_{0j} = \alpha \delta_j$$

Weight and bias updation (Phase III):

Step 8: Each output unit (y_k, k = 1 to m) updates the bias and weights:

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$$

$$w_{0k}(new) = w_{0k}(old) + \Delta w_{0k}$$

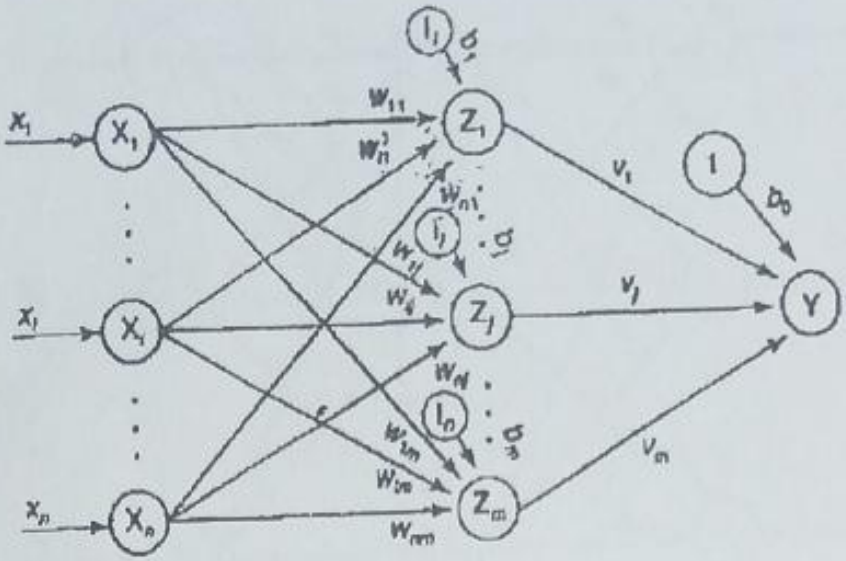
Each hidden unit (z_j, j = 1 to p) updates its bias and weights:

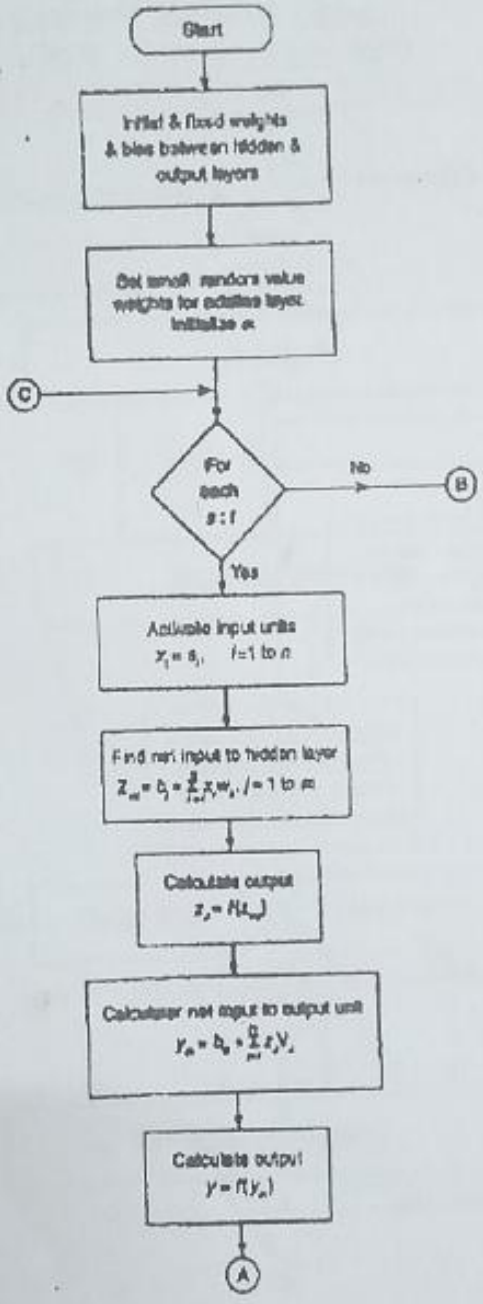
$$v_{ij}(new) = v_{ij}(old) + \Delta v_{ij}$$

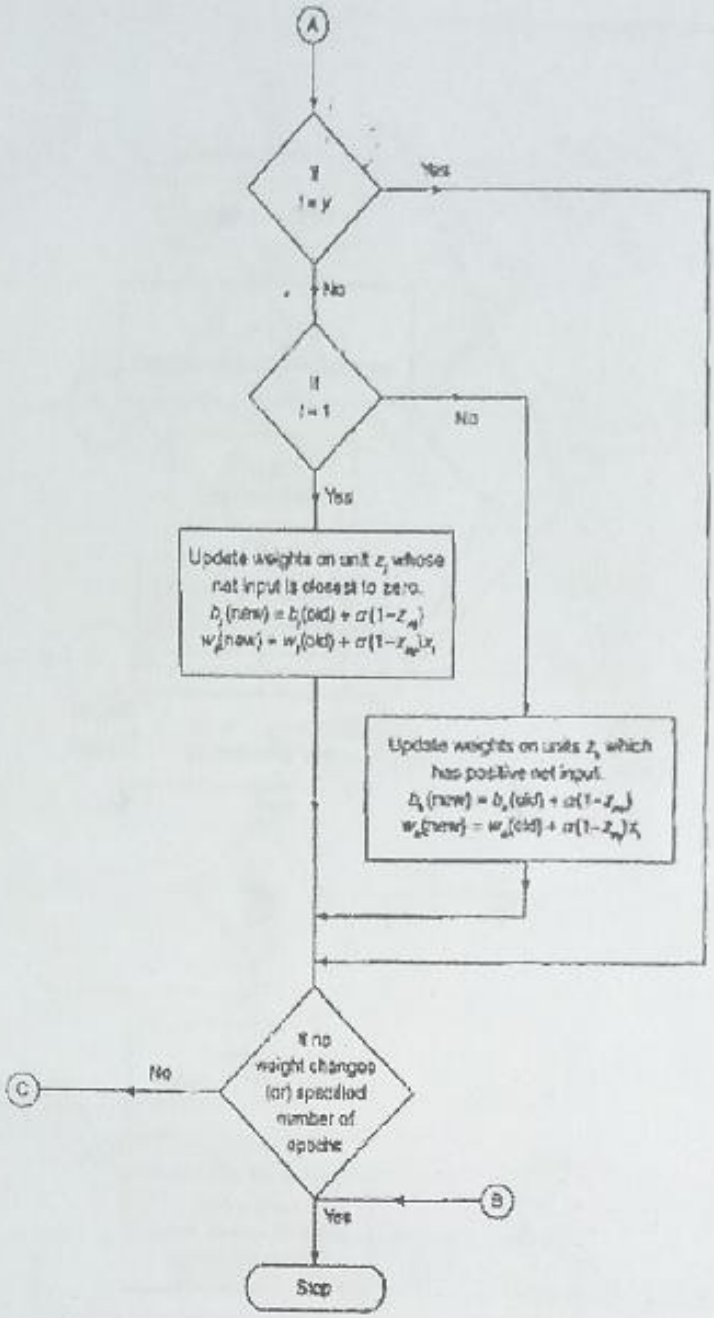
$$v_{0j}(new) = v_{0j}(old) + \Delta v_{0j}$$

Step 9: Check for the stopping condition. The stopping condition may be certain number of epochs reached or when the actual output equals the target output.

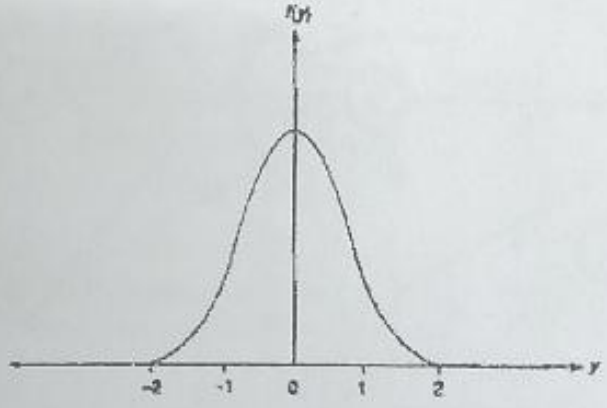
2.4 Madaline

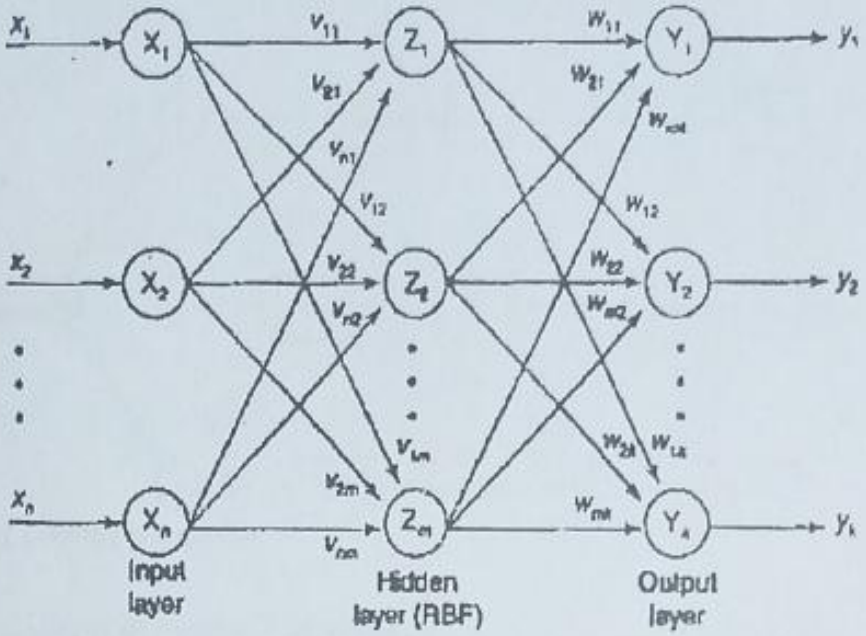



Start
Initial & fixed weights & bias between hidden & output layers
Set small random value weights for activation layer. Initialize α
C
For each s : t
No
B
Yes
Activate input units
$x_i = s_i$ i=1 to n
Find net input to hidden layer
$Z_{in} = b_j = \sum_{i=1}^{n} x_i w_{ij}$ j = 1 to m
Calculate output
$z_j = f(z_{in})$
Calculate net input to output unit
$y_{in} = b_0 + \sum_{j=1}^{m} z_j v_j$
Calculate output
$y = f(y_{in})$
A

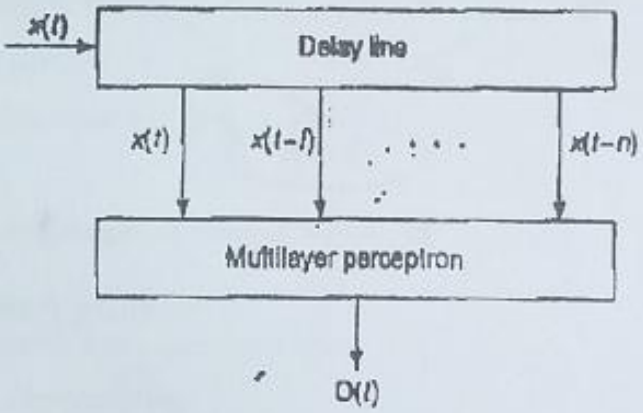


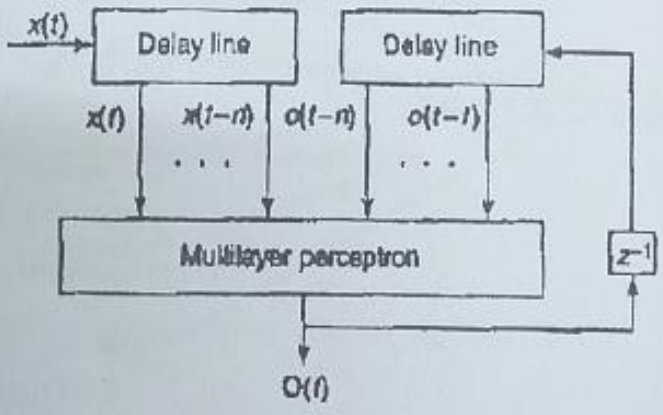
2.5 Radial Basis Function Network



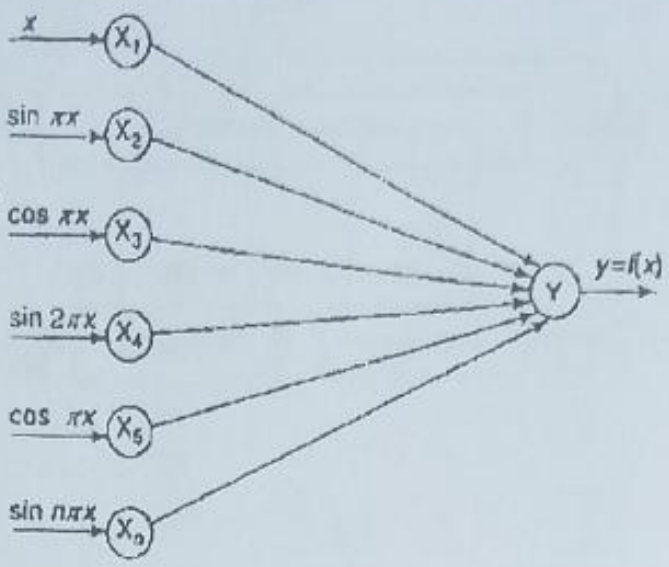


2.6 Time Delay Neural Network

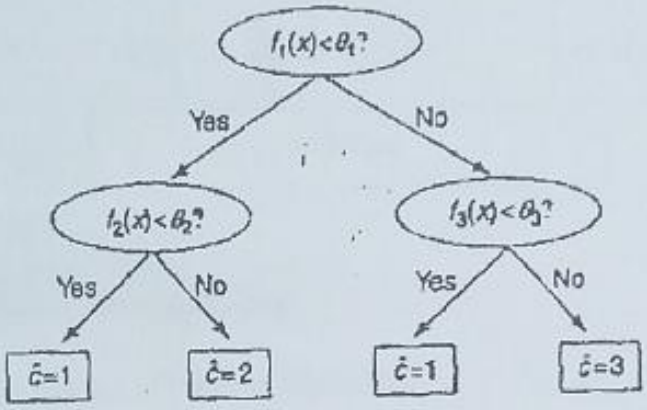




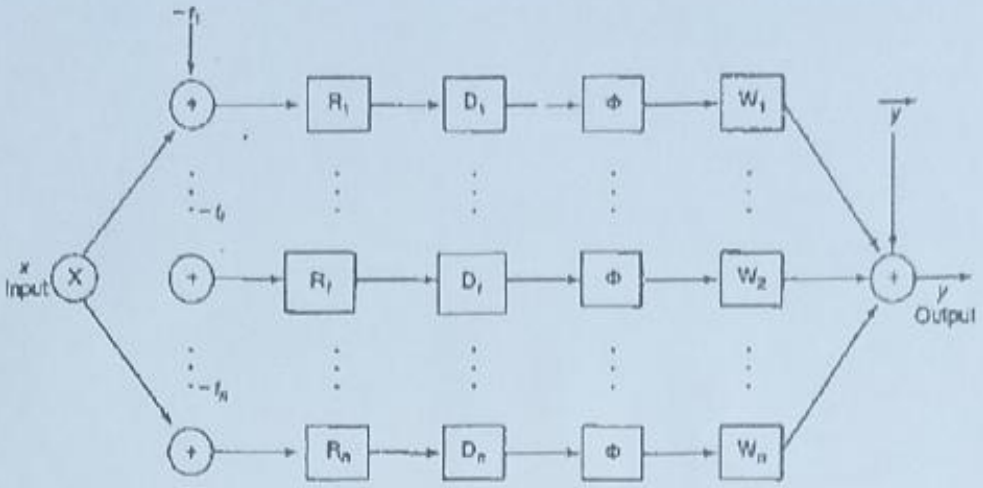
2.7 Functional Link Networks



2.8 Tree Neural Networks



2.9 Wavelet Neural Networks



2.10 Advantages of Neural Networks

1. Mimicks human control logic.
2. Uses imprecise language.
3. Inherently robust.
4. Fails safely.
5. Modified and tweaked easily.

2.11 Disadvantages of Neural Networks

1. Operator's experience required.
2. System complexity.

2.12 Applications of Neural Networks

1. Automobile and other vehicle subsystems, such as automatic transmissions, ABS and cruise control (e.g. Tokyo monorail).
2. Air conditioners.
3. Auto focus on cameras.
4. Digital image processing, such as edge detection.
5. Rice cookers.
6. Dishwashers.
7. Elevators.