Unit-4

Neural Networks

Introduction to Neural Networks: Neural processing, Neural Networks - an overview, the rise of Neuro computing.

Introduction to Artificial Neural Networks: Introduction, Artificial Neural Networks, historical development of Neural Networks, biological neural networks, comparison between the brain and the computer, comparison between artificial and biological neural network, Basic building blocks of ANN, ANN terminologies.

Fundamental Models of Artificial Neural Networks: Introduction, McCulloch-Pitts Neuron Model, Learning rules, Hebb net.

Introduction to Neural Networks

Neural processing

Neural Networks (NNs) represent a meaningfully different approach to using computers in the workplace. A neural network is used to learn patterns and relationships in data.

Who is concerned with Neural Networks?

1) Computer scientists want to find out about the properties of non-symbolic information processing with neural networks and about learning systems in general.

2) Engineers of many kind want to exploit the capabilities of neural networks in many areas (eg signal processing) to solve their application problems.

3) Cognitive scientists view neural networks as a possible apparatus to describe models of thinking and conscience (high-level brain function)

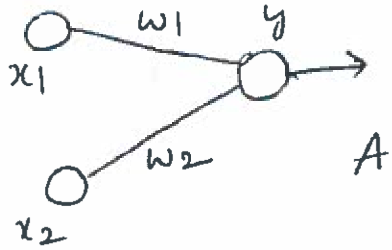
4) Neuro-physiologists use neural networks to describe and explore medium-level brain function (eg, memory, sensory system).

5) physicists use neural networks to model phenomena in statistical mechanics and for a lot of other tasks.

:) Biologists use Neural Networks to interpret nucleotide sequences.

:) philosophers use NN to gain knowledge about the human systems namely behavior, conduct, character, intelligence, brilliance and other psychological feelings.

Neural Networks - An overview

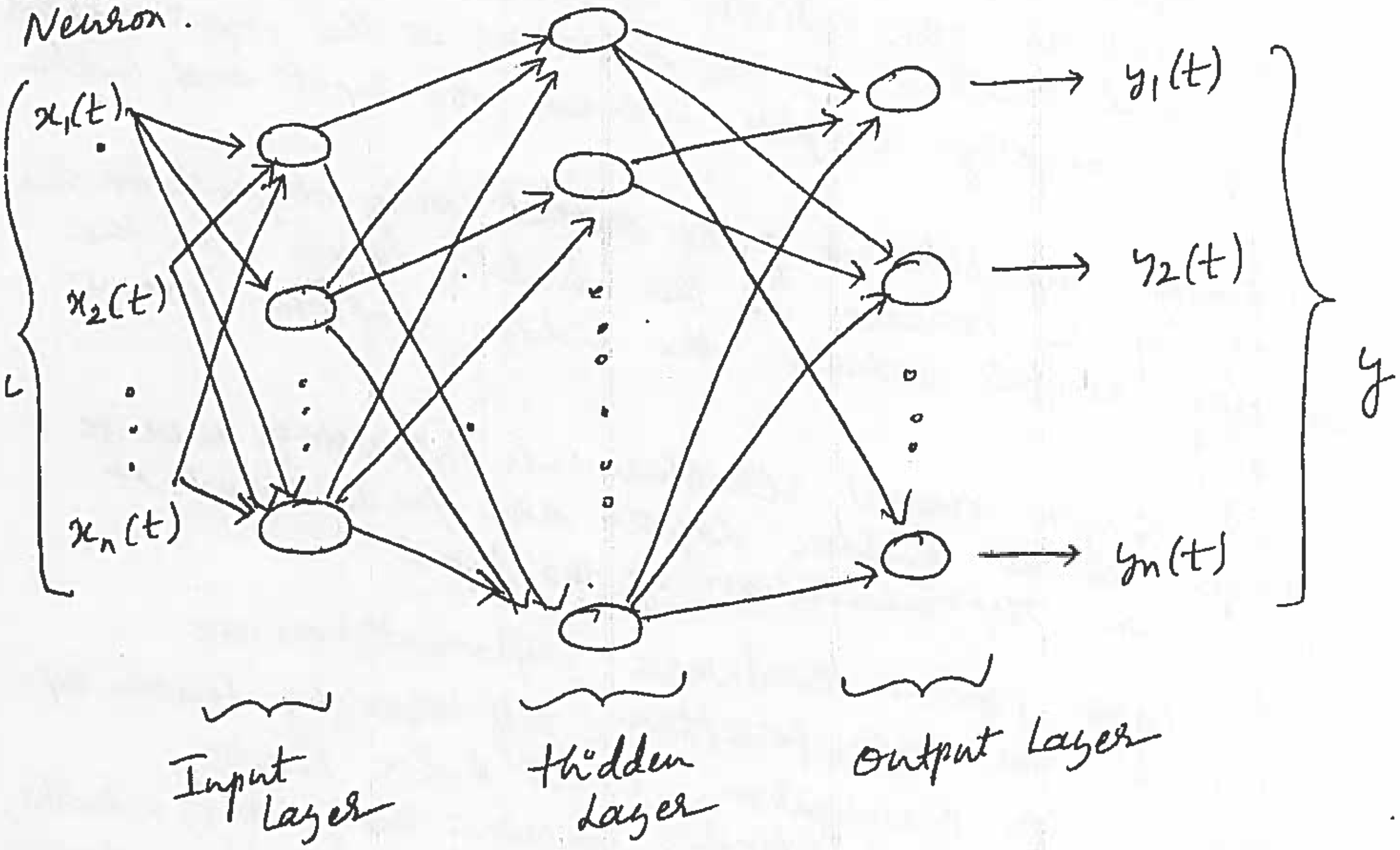


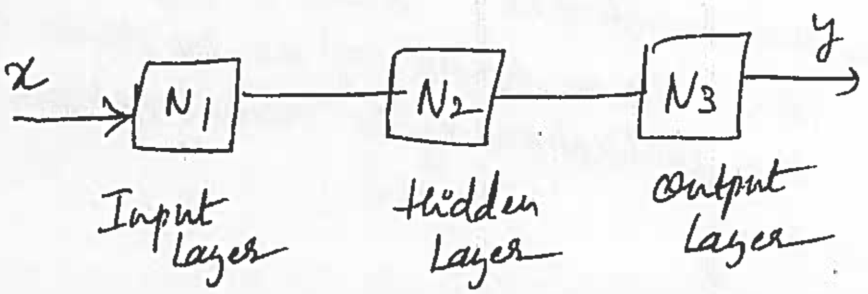
A Simple Artificial Neural Net.

- Artificial Neural Networks (ANNs) are gross simplifications of real (biological) networks of neurons.
- The aim of neural networks is to mimic the human ability to adapt to changing circumstances and the current environment.
- ANNs consist of many nodes i.e., processing units analogous to neurons in the brain. Each node has a node function, associated with it which along with a set of local parameters determines the output of the node, given an input
- Modifying the local parameters may alter the node function.
- ANNs thus is an information-processing system. In this information-processing system, the elements called neurons process the information. The signals are transmitted by means of connection links. The links possess an associated weight, which is multiplied along with the incoming signal (net input) for any typical neural net. The output signal is obtained by applying activations to the net input

- The neural net can generally be a single layer (4.2") or a multi-layer net.
- The structure of the simple artificial neural net is shown in Fig ①
- Fig ① shows a simple artificial neural net with two input neurons (x_1, x_2) and one output neuron (y). The inter connected weights are given by w_1 and w_2. In a single layer net there is a single layer of weighted interconnections.

A densely interconnected Three-Layered static Neural Network. Each shaded circle, or Node represents an Artificial Neuron.





- A Block diagram representation of a three-layered MNN.

- A typical multi-layer artificial neural network, (MNN) comprises an input layer, output layer and hidden (intermediate) layer of neurons.
- MNNs are often called layered networks. They can implement arbitrary complex input/output mappings or decision surfaces separating different patterns.
- A three-layer MNN is shown in Fig ② and a simplified block diagram representation in Fig ③.
- In a MNN, a layer of input units is connected to a layer of hidden units, which is connected to the layer of output units. The activity of neurons in the input layer represents the raw information that is fed into the network. The activity of neurons in the hidden layer is determined by the activities of the input neurons and the connecting weights between the input and hidden units.
- Similarly, the behavior of the output units depends on the activity of the neurons in the hidden layer and the connecting weights between the hidden and the output layers.
- This simple neural structure is interesting because neurons in the hidden layers are free to construct their own representation of the input.
- The most popular hardware implementations are Hopfield, Multi-layer perceptron, self-organizing feature map, Learning Vector Quantization, Radial Basis Function, Cellular Neural, and Adaptive Resonance Theory (ART) networks, Counter propagation networks, Back Propagation networks, Neo-Cognitron etc., As a result of the existence of all these networks, the application of the neural network is increasing tremendously.

The Rise of NeuroComputing. (4.3)

— Digital computers developed rapidly in and after the late 1940's and after originally being applied to the field of mathematical computations, have found expanded applications in a variety of areas, like text (word), symbol, image and voice processing i.e, pattern information processing, robotic control and artificial intelligence.

- However. the human nervous system, it is now known consists of an extremely large number of nerve cells, or neurons, which operate in parallel to process various types of information. By taking a hint from the structure of the human nervous system, we should be able to build a new type of advanced parallel information processing device.

- In addition to the increasingly large volumes of data that we must process as a result of recent developments in sensor technology and the progress of information technology, there is also a growing requirement to simultaneously gather and process huge amounts of data from multiple sensors and other sources. This situation is creating a need in various fields to switch from conventional computers that process information sequentially, to parallel computers equipped with multiple processing elements, aligned to operate in parallel to process information.

Research in the fields of mathematical science and physics is also concentrating more on the mathematical analysis of systems comprising multiple elements that interact in complex ways. These factors gave birth to a major research trend aimed at clarifying the structures and operating principles inherent in the information processing systems of human beings and other animals, and constructing an information processing device based on these structures and operating principles. The term " Neuro computing " is used to refer to the information engineering aspects of this research

— x —

Introduction to Artificial Neural Networks

Introduction

A brief summary of the history of neural networks, in terms of the development of architectures and algorithms, the structure of the biological neuron is discussed and compared with the artificial neuron. The basic building blocks and the various terminologies of the ANN are explained. Summary of notations, which are used in ~~the~~ all the network algorithms, architectures etc are discussed.

Artificial Neural Networks (ANN)

. Artificial neural networks are non-linear information (signal) processing devices, which are built from interconnected elementary processing devices called neurons.

An 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 union to solve specific problems. ANNs like people, learn by example.

. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process.

. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons.

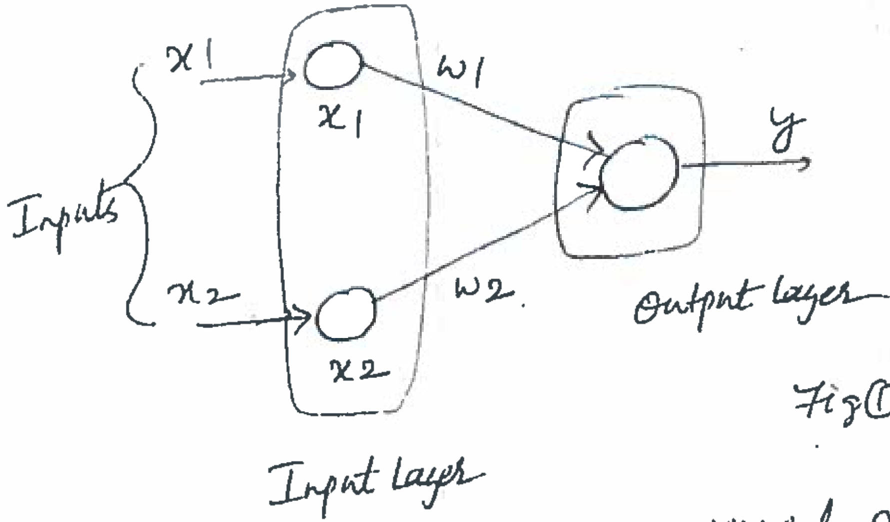
. This is true of ANNs as well.

. Rather than using a digital model, in which all computations manipulate zeros and ones, a neural network works by creating connections between processing elements, the computer equivalent of neurons. The organization and weights of the connections determine the output.

- A neural network is a massively parallel-distributed (4·4) processor that has a natural propensity for storing experimental knowledge and making it available for use.
- It resembles the brain in two respects:
  1. Knowledge is acquired by the network through a learning process, and
  2. Inter-neuron connection strengths known as synaptic weights are used to store the knowledge.
- Neural networks can also be defined as parameterized computational non-linear algorithms for (numerical) data/signal/image processing.
- These algorithms are either implemented on a general-purpose computer or are built into a dedicated hardware.

Artificial Neural Networks thus is an information-processing system. In this information-processing system, the elements called as neurons, process the information. The signals are transmitted by means of connection links. The links possess an associated weight, which is multiplied along with the incoming signal (net input) for any typical neural net.
- The output signal is obtained by applying activations to the net input.
- An artificial neuron is characterized by:
  1. Architecture (connection between neurons)
  2. Training & learning (determining weights on the connections)
  3. Activation function.
- The structure of the simple artificial neural network is shown in Fig 1

w_1 & w_2 = weights



Fig①: A simple Artificial Neural Net.

Figure ① shows a simple artificial neural network with two input neurons ($x1$, x_2) and the one output neuron (y).

- The inter connected weights are given by w_1 and w_2.
- An artificial neuron is a p-input single-output signal processing element, which can be thought of as a simple model of a non-branching biological neuron.
- In Fig①, various inputs to the network are represented by the mathematical symbol, $x(n)$. Each of these inputs are multiplied by a connection weight. these weights are represented by $w(n)$.
- In the simplest case, these products are simply summed, fed through a transfer function to generate a result, and then delivered as output.

This process lends itself to physical implementation on a large scale in a small package. This electronic implementation is still possible with other network structures, which utilize different summing functions as well as different transfer functions.

Why Artificial Neural Networks?

The long course of evolution has given the human brain many desirable characteristics not present in ... Neumann. or modern Parallel computers. these include.

(4.5)

- Massive parallelism
- Distributed representation and computation.
- Learning ability
- Generalization ability
- Adaptivity
- Inherent contextual information processing
- Fault tolerance and
- Low energy consumption.

Modern digital computers outperform humans in the domain of numeric computation and related symbol manipulation. However, humans can effortlessly solve complex perceptual problems (like recognizing a man in a crowd from a mere glimpse of his face) at such a high speed and extent as to dwarf the world's fastest computer.

– Numerous efforts to develop "intelligent" programs based on Von Neumann's centralized architecture have not resulted in any general-purpose intelligent programs. Inspired by biological neural networks, ANNs are massively parallel computing systems consisting of an extremely large number of simple processors with many interconnections. ANN models attempt to use some "organizational" principles believed to be used in the human brain.

Von Neumann Computer V/s Biological Neural System

	Von Neumann V.N	BNN
Processor	Complex High speed one or a few	Simple Low speed A large number
Memory	separate from a processor localized, Non content addressable.	Integrated into processor Distributed content addressable
computing	centralized, sequential, stored programs.	Distributed, parallel, self-learning.
Reliability Expertise	Very Vulnerable, Numerical & symbolic manipulations.	Robust, perceptual problems.
Operating Environment	Well defined, well-constrained.	poorly defined unconstrained.

Either humans or other computer techniques can use neural networks with their remarkable ability to derive meaning from complicated or imprecise data, to extract patterns and detect trends that are too complex to be noticed.

A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. 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, using special hardware devices designed and manufactured to take advantage of this capability.
4) Fault Tolerance via redundant information coding: Partial destruction of a network leads to a corresponding degradation of performance. However, some network capabilities may be retained even after major network damage due to this feature.

Historical Development of Neural Networks

The historical development of the neural networks can be treated as follows:

1) 1943 — McCulloch and Pitts: start of the modern era of neural networks.

This forms a logical calculus of neural networks. A network consists of sufficient number of neurons (using a simple model) and properly set synaptic connections can compute any computable function. A simple logic function is performed by a neuron in this case based upon the weights set in the McCulloch-Pitts neuron.

The arrangement of neuron in this case may be represented as a combination of logic-functions.

The most important feature of this type of neuron is the concept of threshold. When the net input to a particular neuron is greater than the specified threshold by the user, then the neuron fires. Logic circuits are found to use this type of neurons extensively. (4.6)

1949 – Hebb's book " The organization of behavior"

An explicit statement of a physiological learning rule for synaptic modification was presented for the first time. Hebb proposed that the connectivity of the brain is continually changing as an organism learns differing functional tasks, and that neural assemblies are created by such changes.

Hebb's work was immensely influential among psychologists. The concept behind the Hebb theory is that if two neurons are found to be active simultaneously the strength of connection between the two neurons should be increased. This concept is similar to that of the correlation matrix learning.

1958 – Rosenblatt introduces Perceptron.

In perceptron network the weights on the connection paths can be adjusted. A method of iterative weight adjustment can be used in the perceptron net. The perceptron net is found to converge if the weights obtained allow the net to reproduce exactly all the training input and target output vector pairs.

1960 – Widrow and Hoff introduce Adaline.

ADALINE – Abbreviated from Adaptive Linear Neuron uses a learning rule called as Least Mean Square rule or Delta rule. This rule is found to adjust the weights so as to reduce the difference between the net input to the output unit and the desired output. The convergence criteria in this case are the reduction of mean square error to a minimum value. This delta rule for a single layer net can be called a precursor of the backpropagation net used for multi-layer nets. The multi-layer extensions of Adaline formed the Medaline.

1962 – John Hopfield's networks

Hopfield showed how to use "Ising spin glass" type of model to store information in dynamically stable networks. this work paved the way for physicists to enter neural modeling, thereby transforming the field of neural networks. These nets are widely used as associative memory nets. The Hopfield nets are found to be both continuous valued and discrete valued. This net provides an efficient solution for the "Travelling sales-man Problem".

972 – Kohonen's self-organizing Maps (SOM)

Kohonen's Self-organizing Maps are capable of reproducing important aspects of the structure of biological neural nets. They make use of data representation using topographic maps, which are common in the nervous systems. SOM also has a wide range of applications. It shows how the output layer can pick up the correlational structure (from the inputs) in the form of the spatial arrangement of units. These nets are applied to many recognition problems.

1985 – Parker 1986 – Lecum

During this period the backpropagation net paved its way into the Neural Networks. This method propagates the error information at the output units back to the hidden units using a generalized delta rule. This net is basically a multilayer, feed forward net trained by means of backpropagation. Originally even though the work was performed by Parker the credit of publishing this net goes to Rumelhart, Hinton and Williams. Backpropagation net emerged as the most popular learning algorithm for the training of multi layer perceptrons and has been the workhorse for many neural network applications.

1988 – Grossberg

Grossberg developed a learning rule similar to that of Kohonen, which is widely used in the Counter Propagation net. This Grossberg type of learning is also used as Outstar learning. This learning occurs for all the units in a particular layer; no competition among these units is assumed.

1987, 1990 – Carpenter and Grossberg (4.7)

Carpenter and Grossberg invented Adaptive Resonance theory (ART). ART was designed for both binary inputs and the continuous valued inputs. The design for the binary inputs formed ART1, and ART2, came into being when the design became applicable to the continuous valued inputs. The most important feature of these nets is that the input patterns can be presented in any order.

1988 – Broomhead and Lowe developed Radial Basis Functions (RBF, This is also a multi-layer net that is quiet similar to the back propagation net.

1990 – Vapnik – developed the Support Vector Machine.

Biological Neural Networks.

A biological neuron or a nerve cell consists of synapses, dendrites, the cell body (or hillock) and the axon.

– The "building blocks" are discussed as follows:

- The synapses are elementary signal processing devices
  - A synapse is a biochemical device, which converts a pre-synaptic electrical signal into a chemical signal and then back into a post-synaptic electrical signal.
  - The input pulse train has its amplitude modified by parameters stored in the synapse. The nature of this modification depends on the type of the synapse, which can be either inhibitory or excitatory
- The post-synaptic signals are aggregated and transferred along the dendrites to the nerve cell body.
- The cell body generates the output neuronal signal, a spike, which is transferred along the axon to the synaptic terminals of other neurons.
- The frequency of firing of a neuron is proportional to the total synaptic activities and is controlled by the synaptic parameters (weights)

- The pyramidal cell can receive 104 synaptic inputs and it can fan-out the output signal to thousands of target cells – a connectivity difficult to achieve in the ANNs.

– In general the function of the main elements can be given as:

Dendrite – Receives signals from other neurons

Soma – Sums all the incoming signals.

Axon – when a particular amount of input is received then the cell fires. It transmits signal through axon to other cells.

— The fundamental processing element of a neural network is a neuron. This building block of human awareness encompasses a few general capabilities. Basically, a biological neuron receives inputs from other sources, combines them in some way, performs a generally non-linear operation on the result, and then outputs the final result. Fig 2 shows the relationship of these four parts.

4 parts of a Typical Nerve Cell

Dendrites: Accept inputs

Soma: Process the inputs

Axon: Turn the processed inputs into outputs

Synapses: The electrochemical contact between neurons.

Fig 2: A Biological Neuron.

The properties of the biological neuron pose some features on the artificial neuron, they are:

1) Signals are received by the processing elements. This elements sums the weighted inputs.
2) The weights at the receiving end has the capability to modify the incoming signal.

3) The neuron fires (transmits output), when sufficient input is obtained. (4·8

4) The output produced from one neuron may be transmitted to other neurons.

5) The processing of information is found to be local.

6) The weights can be modified by experience.

7) Neurotransmitters for the synapse may be excitatory or inhibitory.

8) Both artificial and biological neurons have inbuilt fault tolerance.

Fig ③ & ④ indicate how the biological neural net is associated with the artificial neural net.

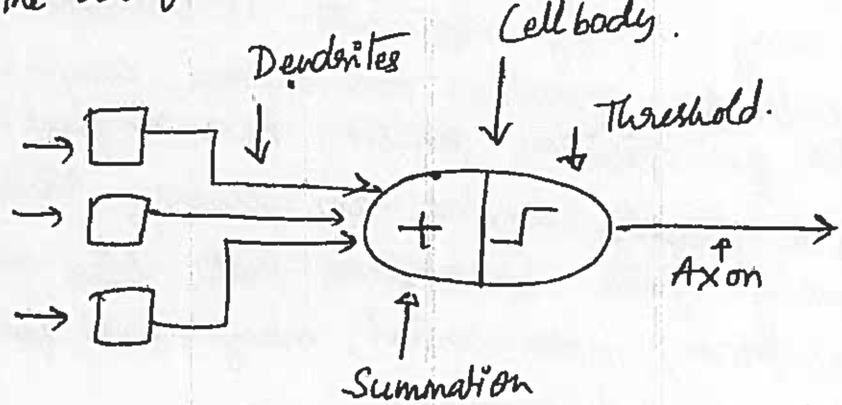


Fig ③ Association of Biological Net with Artificial Net

Fig ④ Associated Terminologies of Biological and Artificial Neural Net

Biological Neural Network	Artificial Neural Network
Cell Body	Neurons
Dendrite	Weights or interconnections
Soma	Net input
Axon	output.

Comparison Between the Brain and the Computer.

The main differences between the brain and the computer are:

1) Biological Neurons, the basic building blocks of the brain, are slower than silicon logic gates. The neurons operate in milliseconds, which is about six orders of magnitude slower than the silicon gates operating in the nanosecond range.

2) The brain makes up for the slow rate of operation with two factors:
   - A huge number of nerve cells (neurons) and interconnections between them. The human brain contains approximately 10^{14} to 10^{15} interconnections.
   - the function of a biological neuron seems to be much more complex than that of a logic gate.

3) The brain is very energy efficient. It consumes only about 10-16 joules per operation per second, comparing with 10-6 joules per operation per sec, for a digital computer.

4) The brain is a highly complex, non-linear, parallel-information processing system. It performs tasks like pattern recognition, perception, motor control, many times faster than the fastest digital computers.

5) Consider an efficiency of the visual system which provides a representation of the environment which enables us to interact with the environment. For ex, a complex task of perceptual recognition eg; recognition of a familiar face embedded in an unfamiliar scene can be accomplished in 100-200 ms, whereas tasks of much lesser complexity can take hours if not days on conventional computers.

Comparison between Artificial and Biological NN (4.9)

Characteristics	Artificial Neural N/w	Biological NN
Speed.	Neural Networks are faster in processing information. The cycle time corresponding to execution of one step of a program in the central processing unit is in the range of few nano seconds	Biological neurons are slow in processing information. The cycle time corresponding to a neural event prompted by an external stimulus occurs in a milli second range.
Processing	Many programs have large number of instructions, and they operate in a sequential mode one instruction after another on a conventional computer.	Biological neural networks can perform massively parallel operations. The brain possesses the capability to operate with massively parallel operations, each of them having only few steps.
Size and complexity.	These do not involve as much computational neurons. Hence, it is difficult to perform complex pattern recognition.	Neural networks have large number of computing elements, and the computing is not restricted to within neurons. The number of neurons in the brain is estimated to about 10^{11} and the total number of interconnections to be around 10^{15}. The size and complexity of connections gives the brain the power of performing complex pattern recognition tasks which cannot be realized on a computer.
Storage	In a computer, the information is stored in the memory, which is addressed by its location. Any new information in the same location destroys the old information. Hence, here it is strictly replaceable.	Neural networks store information in the strengths of the interconnections. Information in the brain is adaptable, because new information is added by adjusting the interconnection strengths, without destroying the old information

Fault Tolerance	Artificial nets are inherently not fault tolerant, since the information corrupted in the memory cannot be retrieved.	They exhibit fault [illegible] since the information is distributed in the connections throughout the network. Even though if few connections are not working, the information is still preserved due to the distributed nature of the encoded information.
Control mechanism	There is a control unit, which monitors all the activities of computing.	There is no central control for processing information in the brain. The neuron acts based on the information locally available & transmits its output to the neurons connected to it. There is no specific control mechanism external to the computing task.

Basic Building Blocks of Artificial Neural Networks.

The basic building blocks of the Artificial NN are.

1) Network Architecture
2) Setting the Weights
3) Activation Function.

Network Architecture

The arrangement of Neurons into layers and the pattern of connection within and in-between layer are generally called as the architecture of the net.

The neurons within a layer are found to be fully interconnected or not interconnected.

The number of layers in the net can be defined to be the number of layers of weighted interconnected - links between the Particular slabs of neurons.

If two layers of interconnected weights are present, then it is found to have hidden layers.

There are various types of network architectures:

Feed forward, feedback, fully interconnected net, Competitive net etc.,

Fig: Some Artificial Neural Network Connection Structures.

1) Feed Forward Net: Feed Forward networks may have a single layer of weights where the inputs are directly connected to the outputs, or multiple layers with intervening sets of hidden units (Fig). Neural networks use hidden units to create internal representations of the input patterns. Infact, it has been shown that given enough hidden units, it is possible to approximate arbitrarily any function with a simple feed forward network.
This result has encouraged people to use neural networks to solve many kinds of problems.

   1) Single Layer net: It is a feedforward net. It has only one layer of weighted interconnections. The inputs may be connected fully to the output units. But there is a chance that none of the input units and output units are connected with other input and output units respectively. There is also a case where, the input units and output units ~~are~~ connected with other input units and ~~on~~ with other output units.

In a single layer net, the weights from one output unit do not influence the weights for other output units.

2) Multi-layer net: It is also a feed forward net i.e, the net where the signals flow from the input units to the output units in a forward direction. The multi-layer net pose one or more layers of nodes between the input and output units. It can be used to solve more complicated problems.

Competitive Net

The competitive net is similar to a single-layered feed forward network except that there are connections, usually negative, between the output nodes. Because of these connections the output nodes tend to compete to represent the current input pattern. Sometimes the output layer is completely connected and sometimes the connections are restricted to units that are close to each other. With an appropriate learning algorithm the latter type of network can be made to organize itself topologically. In a topological map, neurons near each other represent similar input patterns. Networks of this kind have been used to explain the formation of topological maps that occur in many animal sensory systems including vision, audition, touch and smell.

Recurrent Net

The fully recurrent network is perhaps the simplest of neural network architectures. All units are connected to all other units and every unit is both an input and an output. Typically, a set of patterns is instantiated on all of the units, one at a time. As each pattern is instantiated the weights are modified. When a degraded version of ~~the~~ one of the patterns is presented, the network attempts to reconstruct the pattern.

R. Ns allow networks to process sequential information. The response to the current input depends on previous inputs. Ex. (Fig) The simple recurrent network and the Jordan network.

2) Setting the Weights

(4.11)

The method of setting the value for the weights enables the process of learning or training. The process of modifying the weights in the connections between network layers with the objective of achieving the expected output is called training a network. The internal process that takes place when a network is trained is called learning.

Generally there are three types of training:

1) Supervised Training

Supervised training is the process of providing the network with a series of sample inputs and comparing the output with the expected responses. The training continues until the network is able to provide the expected response. In a neural net, for a sequence of training input vectors there may exist target output vectors. The weights may then be adjusted according to a learning algorithm. This process is called supervised training.

Ex: Hebb net, pattern association memory net, Back propagation net, counter propagation net etc.

2) Un-supervised Training.

In a neural net, if for the training input vectors, the target output is not known, the training method adopted is called as unsupervised training. The net may modify the weight so that the most similar input vector is assigned to the same output unit. The net is found to form a exemplar or code book vector for each cluster formed.

These networks are far more complex and difficult to implement. It involves looping connections back into feedback layers and iterating through the process until some sort of stable recall can be achieved.

Unsupervised networks are also called self-learning networks or self-organizing networks because of their ability to carry out self-learning.

3) Reinforcement Training.

Reinforcement learning is a very general approach to learning that can be applied when the knowledge required to apply supervised learning is not available.

If sufficient information is available, the reinforcement learning can readily handle a specific problem.

Reinforcement training is related to supervised training. the output in this case may not be indicated as the desired output, but the condition whether it is "success" (+1) or 'failure' (0) may be indicated. Based on this, error may be calculated and the training process may be continued. The error signal produced from reinforcement training is found to be binary. Reinforcement learning attempts to learn the input-output mapping through trial and error with a view to maximize a performance index called the reinforcement signal.

Many of these learning methods are closely connected with a certain (class of) network topology.

1) Unsupervised learning
   a) Feed back Nets
      1) Binary Adaptive Resonance Theory (ART1)
      2) Analog Adaptive Resonance Theory (ART2, ART2a)
      3) Discrete Hopfield (DH)
      4) Continuous Hopfield (CH)
      5) Discrete Bi-directional Associative Memory (BAM)
      6) Temporal Associative Memory (TAM)
      7) Adaptive Bi-directional Associative Memory (ABAM)
      8) Kohonen Self-Organizing Map/Topology-preserving map (SOM/TPM)
      9) Competitive learning.

   b) Feedforward-only Nets:
      1) Learning Matrix (LM)
      2) Driver-Reinforcement Learning (DR)
      3) Counter Propagation (CPN)

2) Supervised learning. (4.12)

1) Feedback Nets:

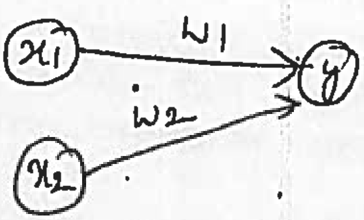
1) Boltzmann Machine (BM)
2) Mean Field Annealing (MFT)
3) Recurrent Cascade Correlation (RCC)
4) Learning Vector Quantization (LVQ)
5) Backpropagation through time (BPTT)
6) Real-time recurrent learning (RTRL)

2) Feedforward-only Nets:

1) Perceptron
2) Adaline, Madaline
3) Backpropagation (BP)
4) Cauchy Machine (CM)
5) Artmap
6) Cascade Correlation (CasCor)

Artificial Neural Network (ANN) Terminologies:

1) Weights



A simple Neural Net.

A neural network consists of a large number of simple processing elements called neurons. These neurons are connected to each other by directed communication links, which are associated with weights.

" Weight is an information used by the neural net to solve a problem".

Fig. shows a simple neural network. The weights that carry information are denoted by w_1 and w_2. They may be fixed, or can take random values. Weights can be set to zero, or can be calulated by some methods.

Initialization of weights is an important criteria in a neural net. The weight changes indicate the overall performance of the neural net.

x_1 = Activation of neuron 1 (input signal)
x_2 = Activation of neuron 2 (input signal)
y = output neuron
w_1 = Weight connecting neuron 1 to output
w_2 = Weight connecting neuron 2 to output.

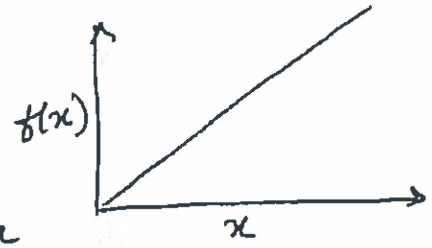
Based on all these parameters, the net input 'Net' is calculated. The Net is the summation of the products of the weights and the input signals.

$$Net = x_1 w_1 + x_2 w_2$$

Generally, it can be written as

$$\text{Net input} = Net = \sum_i x_i w_i$$

From the calculated net input, applying the activation functions, the output may be calculated.



Identity function.

Fig 6

Activation Functions

The activation function is used to calculate the output response of a neuron. The sum of the weighted input signal is applied with an activation to obtain the response. For neurons in same layer, same activation functions are used. There may be linear as well as non-linear activation functions. The non-linear activation functions are used in a multi-layer net.

A few linear and non linear activation functions are discussed here:

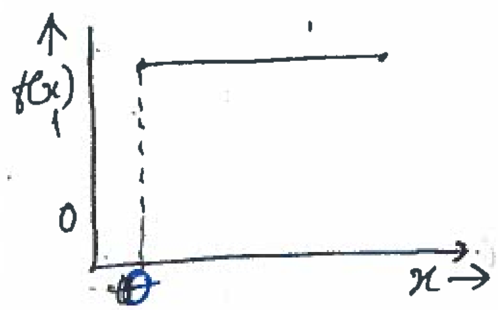
Identity Function

This function is given by $f(x) = x$; for all x.
Fig 6

Binary step function

This function is given by

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



Binary step function.

Mostly single layer nets use binary step function for calculating the output from the net input. The binary step function is also called as threshold function or Heaviside function. (4.13)

3) Sigmoidal functions

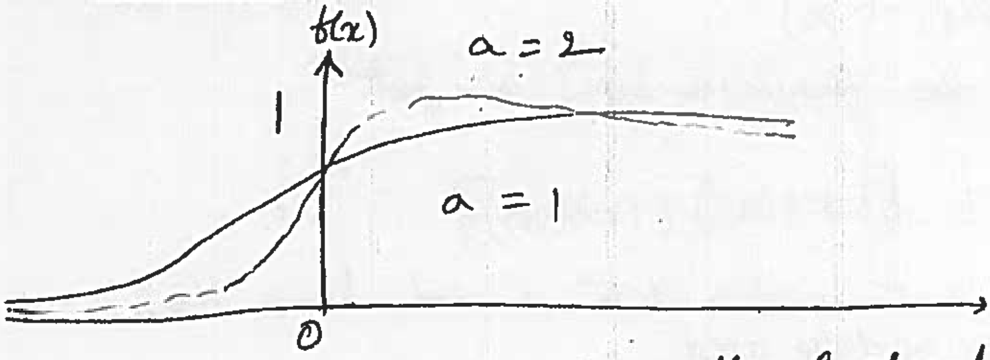


Fig 8
Binary sigmoidal function.

These functions are usually S-shaped curves. The hyperbolic and logistic functions are commonly used. These are used in multilayer nets like back propagation network, radial basis function network etc. There are 2-main types of sigmoidal functions.

Binary Sigmoidal function. and
Bipolar — a — — a —

a) Binary sigmoidal function (Fig 8)
This is also called as logistic function. It ranges from 0 to 1

$$f(x) = \frac{1}{1+\exp(-\alpha x)}$$

Where α is called the steepness parameter.
If $f(x)$ is differentiated we get $f'(x) = \alpha f(x)[1-f(x)]$

b) Bipolar Sigmoidal function. (Fig-9)
The desired range here is between +1 and −1. This function is related to the hyperbolic tangent function. The bipolar sigmoidal function is given as.

$$b(x) = 2f(x) - 1$$

$$b(x) = 2 \times \frac{1}{1+\exp(-\alpha x)} - 1$$

$$= \frac{2-1-\exp(-\sigma x)}{1+\exp(-\sigma x)}$$

$$b(x) = \frac{1-\exp(-\sigma x)}{1+\exp(-\sigma x)}$$

on differentiating the function $b(x)$, we get

$$b'(x) = \frac{\sigma}{2}\left[\{1+b(x)\}(1-b(x))\right]$$

Mostly it is found that bipolar data is used, hence this activation function is widely used.

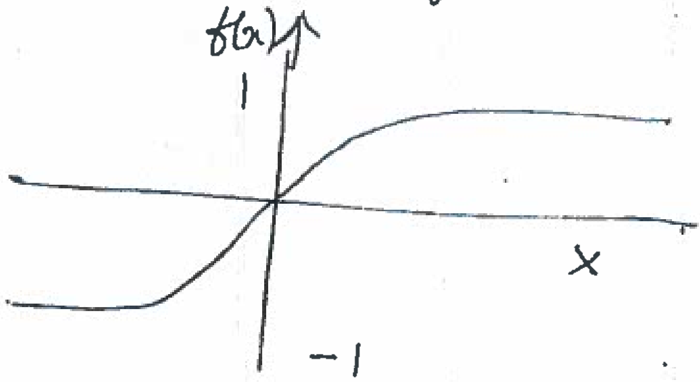


Fig (9)
Bipolar Sigmoidal function.

Calculation of Net Input using Matrix Multiplication Method.

If the weights are given as $W = (w_{ij})$ in a matrix form, The net input to output unit y_j is given as the dot product of the input vectors $x = (x_1, \ldots x_i \ldots x_n)$ and w_j (j^{th} column of the weight vector matrix).

$$y_{inj} = x_i w_j$$

$$y_{inj} = \sum_{i=1}^{n} x_i w_{ij}$$

Hence net input can be calculated using matrix multiplication method.

Bias

(4.14)

A bias acts exactly as a weight on a connection from a unit whose activation is always 1. Increasing the bias increases the net input to the unit ($b = w_0$). Fig ⑩ shows a simple neural net with the bias included.

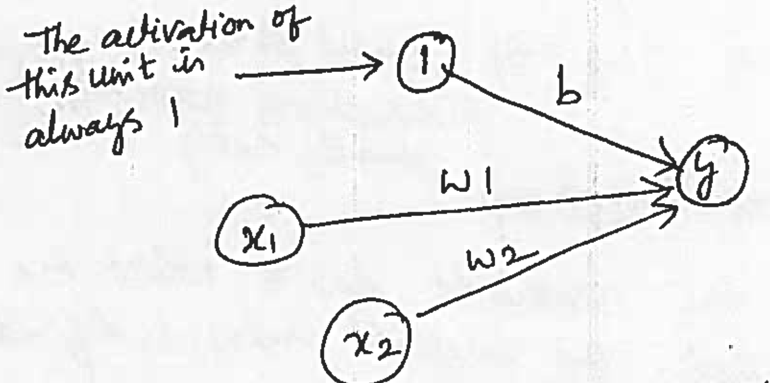


Fig ⑩ A Simple Net with Bias included.

The bias improves the performance of the neural network. Similar to initialization of weights, bias should also be initialized either to 0, or to any specified value, based on the neural net. If bias is present, then net input is calculated as.

$$Net = b + \sum x_i w_i$$

where Net = net input

b = bias

x_i = Input from neuron i

w_i = Weight of the neuron i to the output neuron

Hence, the activation function is obtained as

$$f(Net) = \begin{cases} +1 ; & \text{if net} \geq 0; \\ -1 ; & \text{if net} < 0 \end{cases}$$

if bias is included.

Threshold : The threshold 'θ' is a factor which is used in calculating the activations of the given net. Based on the value of threshold the output may be calculated, i.e., the activation function is based on the value of θ.

For example, the activation functions may be

(i) $y = f(Net) = \begin{cases} +1 & \text{if } net \geq \theta \\ -1 & \text{if } net < \theta \end{cases}$

(ii) $y_j = f(Net) = \begin{cases} 1 & \text{if } y_{inj} > \theta_j \\ y_j & \text{if } y_{inj} = \theta_j \\ -1 & \text{if } y_{inj} < \theta_j \end{cases}$ (used for a bidirectional associative memory net)

Hence θ and θ_j indicate the thresholds, due to which the systems response is calculated. The threshold value is defined by the user.

<u>Example</u>. If the net input to an output neuron is 0.64 calculate its output when the activation function is

1) binary sigmoidal
2) bipolar "

Solution: Net input to the neuron = 0.64

ⓐ For binary activation function

$$y = f(net\ input) = \frac{1}{1+e^{-0.64}} = 0.6548$$

ⓑ For bipolar activation function

$$y = f(net\ input) = \frac{2}{1+e^{-0.64}} - 1 = 0.3095$$

———— X ————

Fundemental Models of Artificial Neural N/ws. (4.15)

Introduction

McCulloch-Pitts Neuron Model. (1943)

The McCulloch-pitt's model of a neuron is characterized by its formalism, elegant and precise mathematical definition.

McCulloch-Pitts neuron allows binary 0 or 1 states only, i.e., it is binary activated. These neurons are connected by direct weighted path. The connected path can be excitatory or inhibitory. Excitatory connections have positive weights and inhibitory connections have negative weights. There will be same weights for the excitatory connection entering into a particular neuron. The neuron is associated with the threshold value. The neuron fires if the net input to the neuron is greater than the threshold. The threshold is set so that the inhibition is absolute, because, non-zero inhibitory input will prevent the neuron from firing. It takes only one time step for a signal to pass over one connection link.

Architecture

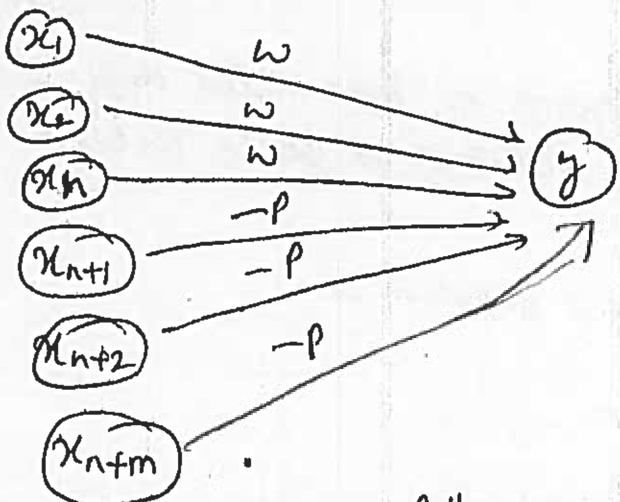


Fig (3.1) Architecture of a McCulloch-Pitts Neuron

'y' is the McCulloch-Pitts neuron, it can receive signal from any number of other neurons. The connection weights from $x_1 \cdots x_n$ are excitatory, denoted by 'w' and the connection weights from $x_{n+1} \cdots x_{n+m}$ are inhibitory denoted by '$-p$'. The McCulloch-pitts neuron y has the activation function.

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

Where θ is the threshold and y-in is the total net input signal received by neuron Y.

- The threshold θ should satisfy the relation

$$\theta > nw - p.$$

This is the condition for absolute inhibition.

The McCulloch-Pitts neuron will fire if it receives k or more excitatory inputs and no inhibitory inputs, where

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

Examples

1) Generate the output of logic AND function by McCulloch-Pitts neuron model.

Soln

3.2 McCulloch-Pitts Neuron to Perform logical AND function.

The AND function returns a true value only if both the inputs are true, else it returns a false value.

'1' = true '0' = false.

The truth table for AND function is

x_1	x_2	y
1	1	1
1	0	0
0	1	0
0	0	0

A McCulloch-Pitts neuron to implement AND function is shown in Fig 3.2. The threshold on unit Y is 2

The output y is

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

The net input is given by

$$y_{in} = \sum_i \text{weights} * \text{input}$$

$$y_{in} = 1 * x_1 + 1 * x_2$$

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

From this the activations of output neuron can be formed.

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y\text{-}in \geq 2 \\ 0 & \text{if } y\text{-}in < 2 \end{cases}$$

11
10
01
00

Now present the inputs

(1) $x_1 = x_2 = 1$, $y_{in} = x_1 + x_2 = 1 + 1 = 2$

$y = f(y_{in}) = 1$ since $y_{in} = 2$

(2) $x_1 = 1$, $x_2 = 0$, $y_{in} = x_1 + x_2 = 0 + 1 = 1$

$y = f(y_{in}) = 0$ since $y_{in} = 1 < 2$

(3) $x_1 = 0$ and $x_2 = 1$ $y_{in} = x_1 + x_2 = 0 + 1 = 1$

$y = f(y_{in}) = 0$ since $y_{in} = 1 < 2$

(4) $x_1 = 0$, $x_2 = 0$ $y_{in} = x_1 + x_2 = 0 + 0 = 0$

Hence $y = f(y_{in}) = 0$ since $y_{in} = 0 < 2$

Ex-2 Generate OR function using McCulloch-Pitts neuron model.

Soln

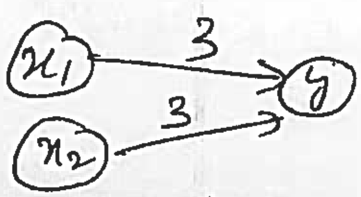


Fig 3.3
McCulloch-Pitts Neuron for OR function.

The OR function returns a high ('1') if any one of the input is high, returns a low ('0') if none of the inputs is high.

The TT for OR function is

x_1	x_2	y
1	1	1
1	0	1
0	1	1
0	0	0

- The threshold for the unit is 3.
- The net input is calculated as

$$y_{in} = 3x_1 + 3x_2$$

- The output is given by

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geq 3 \\ 0 & \text{if } y_{in} < 3. \end{cases}$$

Presenting the inputs

1) $x_1 = x_2 = 1$ $\quad y_{in} = 3x_1 + 3x_2 = 3 + 3 = 6 > \text{threshold } 3.$

Hence $y = 1$

2) $x_1 = 1 \quad x_2 = 0$

$y_{in} = 3x_1 + 3x_2 = 3 + 0 = 3 = \text{threshold}.$

Applying activation formula

$y = f(y_{in}) = 1$

3) $x_1 = 0 \quad x_2 = 1$

$y_{in} = 3x_1 + 3x_2 = 0 + 3 = 3 = \text{threshold}.$

Applying activation formula $y = f(y_{in}) = 1$.

4) $x_1 = x_2 = 0$

$y_{in} = 3x_1 + 3x_2 = 0 < \text{threshold}.$

Hence output $y = 0$.

Ex-3: Realize NOT function using McCulloch-Pitts Neuron model. (4•17)

$$(x) \xrightarrow{1} (y)$$

Fig 3.4. McCulloch-Pitts Neuron for NOT function

Soln: The NOT function returns a true value ('1') if the input is false ('0') and returns a false value ('0') if the input is true ('1')

- The TT for NOT function is

x	y
1	0
0	1

- The McCulloch-Pitts Neuron for this function is given Fig 3.4.
- The threshold for unit y is 1.
- The net input is

$$y_{in} = x \cdot w$$

since $w = 1$ $\quad y_{in} = x$.

- The output activation is given by

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

- presenting the input

1) $x_1 = 1$ $\quad y_{in} = 1$

Applying activation $y = f(y_{in}) = 0$

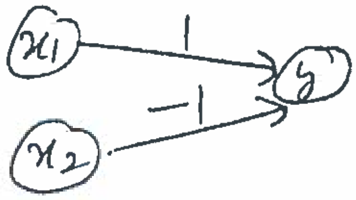
2) $x_1 = 0$ $\quad y_{in} = 0$

Applying activation $y = f(y_{in}) = 1$.

Induction Program

Phase I

Ex-4: Generate the output of ANDNOT function using McCulloch-Pitts Neuron.



Soln: The ANDNOT function returns a true value ('1') if the first input value is true ('1') and the second input value is false ('0')

- The TT for ANDNOT

x_1	x_2	y
1	1	0
1	0	1
0	1	0
0	0	0

- The threshold of unit y is 1.
- The net input is

$$y_{in} = x_1 w_1 + x_2 w_2$$
$$= x_1 * 1 + x_2 * (-1) =$$
$$y_{in} \; x_1 - x_2$$

- The output activation is given as

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

Phase II

Presenting the input

1) $x_1 = x_2 = 1$

$y_{in} = x_1 - x_2 = 1 - 1 = 0 < 1$

Hence $y = f(y_{in}) = 0$

2) $x_1 = 1, x_2 = 0$

$y_{in} = x_1 - x_2 = 1 - 0 = 1 = 1$

$y = f(y_{in}) = 1$

3) $x_1 = 0, x_2 = 1$

$y_{in} = x_1 - x_2 = 0 - 1 = -1 < 1$

$y = f(y_{in}) = 0$

4) $x_1 = x_2 = 0$

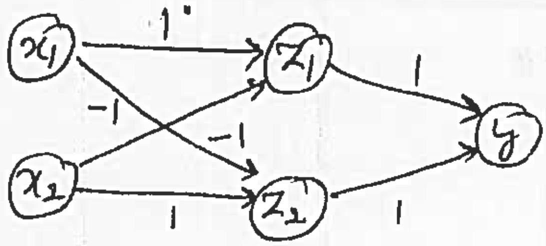
$y_{in} = x_1 - x_2 = 0 - 0 = 0 < 1$

$y = f(y_{in}) = 0$

thus ANDNOT function is realized.

Ex-5 : Realize the Exclusive-OR function using McCulloch-Pitts neuron. (M.18)

Soln



model

XOR function returns a true value if exactly one of the input values is true; otherwise it returns the response as false. The truth table for XOR function is:

x_1	x_2	y
1	1	0
1	0	1
0	1	1
0	0	0

With one-layer alone, it is not possible to predict the value of the threshold for the neuron to fire, hence another layer is introduced.

x_1 XOR x_2 = (x_1 ANDNOT x_2) OR (x_2 ANDNOT x_1)

x_1 XOR x_2 = Z_1 OR Z_2

Where $Z_1 = x_1$ ANDNOT x_2

and $Z_2 = x_2$ ANDNOT x_1

The activations of Z_1 and Z_2 are given as

$$Z_1 = (Z_{in-1}) = \begin{cases} 1 & \text{if } Z_{in-1} \geq 1 \\ 0 & \text{if } Z_{in-1} < 1 \end{cases}$$

$$Z_2 = (Z_{in-2}) = \begin{cases} 1 & \text{if } Z_{in-2} \geq 1 \\ 0 & \text{if } Z_{in-2} < 1 \end{cases}$$

The Calculation of net input and activations of Z_1 and Z_2 are shown below

Z_1 = (x_1 ANDNOT x_2) $\quad Z_{in-1} = x_1 w_1 + x_2 w_2$

x_1	x_2	z_{in-1} $w_1=1$ $w_2=-1$	z_1
1	1	0	0
1	0	1	1
0	1	-1	0
0	0	0	0

$Z_2 = (x_2 \text{ ANDNOT } x_1)$ $\quad z_{in-2} = x_1 w_1 + x_2 w_2$

x_1	x_2	z_{in-2} $w_1=-1$, $w_2=1$	z_2
1	1	0	0
1	0	-1	0
0	1	1	1
0	0	0	0

The activation for the output unit y is 1

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

Presenting the input patterns (z_1 and z_2) and calculating net input and activations gives output of XOR.

Here $y_{in} = z_1 w_1 + z_2 w_2$

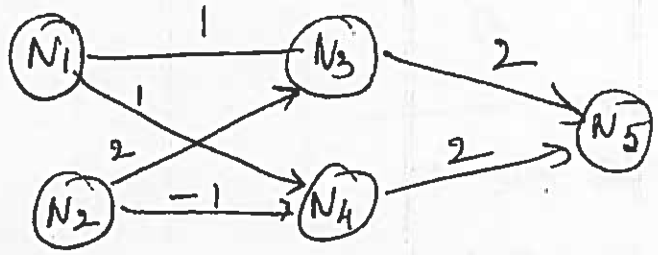
z_1	z_2	y_{in} $w_1=1$ $w_2=1$	$y = z_1$ or z_2
0	0	0	0
1	0	1	1
0	1	1	1
0	0	0	0

Thus Exclusive-OR function is realized.

Ex6: Consider the neural network of McCulloch-Pitts (4·19) neuron shown in Fig below. Each neuron (other than the input neurons N_1 and N_2) has a threshold of 2.

ⓐ Define the response of neuron N_5 at time t in terms of the activations of the input neurons, N_1 and N_2 at the appropriate time.

ⓑ Show that the activation of each neuron that results from an input signal of $N_1 = 1$, $N_2 = 0$ at $t = 0$



Soln: ⓐ To define the response of neuron N_5 at time t

(i) Response of N_3

Given $\omega_1 = 1$, $\omega_2 = 2$, $\theta = 2$

N_1	N_2	$N_{3\text{-}in}$ ($w_1=1$ $\omega_2=2$)	N_3
1	1	3	1
1	0	1	0
0	1	2	0
0	0	0	0

The activation of response N_3 is

$$N_3 = \begin{cases} 1 & \text{if } y_{3\text{-}in} > 2 \\ 0 & \text{if } y_{3\text{-}in} \leq 2 \end{cases}$$

Therefore N_3 is realized as,

$N_3 = N_1 \cdot N_2$ or $N_3 = N_1$ AND N_2

(ii) Response of N_4 Given $w_1 = 1$, $w_2 = -1$, $\theta = 2$

$$N_{4\text{-}in} = N_1 w_1 + N_2 w_2$$

N_1	N_2	$N_{4\text{-}in}$ ($w_1 = 1, w_2 = -1$)	N_4
1	1	0	0
1	0	1	1
0	1	-1	0
0	0	0	0

The activation of response N_4 is

$$N_4 = \begin{cases} 1 & \text{if } y_{4\text{-}in} \geq 1 \\ 0 & \text{if } y_{4\text{-}in} < 1 \end{cases}$$

Therefore, N_4 is realized as

$N_4 = N_1$ AND NOT N_2

(iii) The response of N_5 with the inputs from N_3 and N_4

Given $w_1 = 2$, $w_2 = 2$, $\theta = 2$

$$N_{5\text{-}in} = N_3 w_1 + N_4 w_2$$

N_3	N_4	$N_{5\text{-}in}$ ($w_1 = 2$ $w_2 = 2$)	N_5
1	1	2	1
1	0	2	1
0	1	0	0
0	0	0	0

Therefore N_5 is realized as:

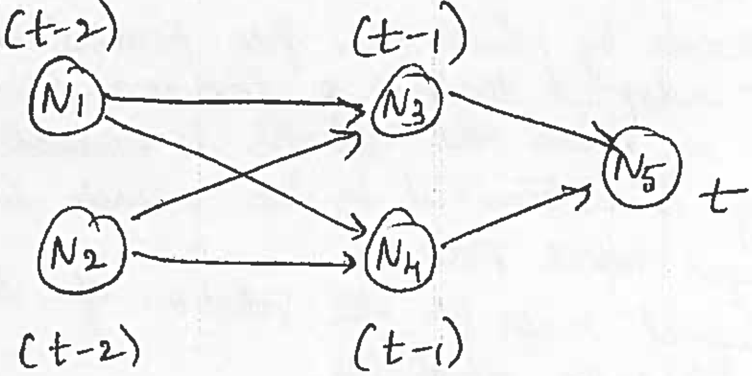
$N_5 = N_3 + N_4$.

The activation of response N_5 is

$$N_5 = \begin{cases} 1 & \text{if } y_{5in} \geq 2 \\ 0 & \text{if } y_{5\text{-}in} < 2 \end{cases}$$

(7·20)

(iv) To define the response of N_5 at time t considering N_5 activates at t, N_3 and N_4 activates at $(t-1)$ and N_1 and N_2 activates at $(t-2)$



Hence $\quad N_5(t) = N_3(t-1) + N_4(t-1)$

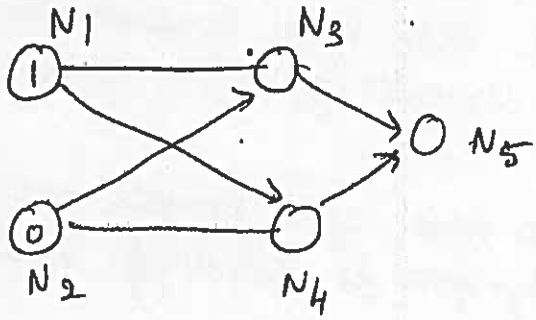
where, $\quad N_3(t-1) = N_1(t-2) \cdot N_2(t-2)$

$N_4(t-1) = N_1(t-2)$ AND NOT $N_2(t-2)$

Hence, $\quad N_5(t) = [N_1(t-2) \cdot N_2(t-2) + N_1(t-2)$ AND NOT $N_2(t-2)]$

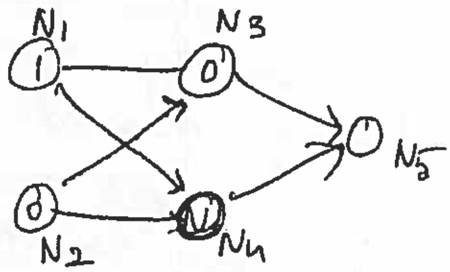
(b) Activation of each neuron with input signals at $N_1 = 1$, $N_2 = 0$ at time $t=0$ At

At $t=0$



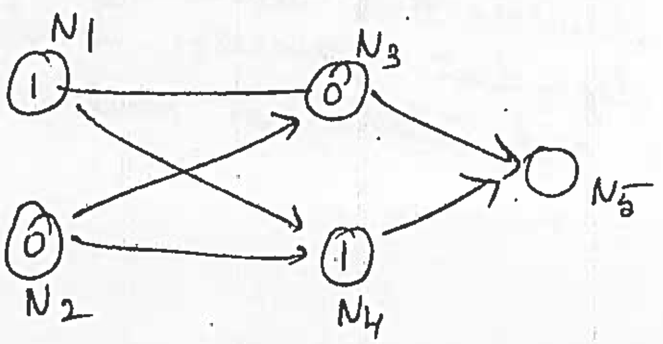
At $t=1$ Calculate N_3

$N_3 = N_1 \cdot N_2 = 0 \cdot 1 = 0$



At $t=2$, Calculate N_4.

$N_4 = N_1$ AND NOT $N_2 = 1$ AND NOT $0 = 1$



Hence

$N_5 = N_3 + N_4 =$

$0 + 1 = 1$

Learning Rules

- A neural network learns about its environment through an interactive process of adjustments applied to its synaptic weights and bias levels.
- Learning is the process by which the free parameters of a neural network get adopted through a process of stimulation by the environment in which the network is embedded.
- The type of learning is determined by the manner in which the parameter changes takes place.
- The set of well defined rules for the solution of a learning problem is called a learning algorithm.

There are Various Learning Rules:

1) **Hebbian Learning Rule**

Hebb's learning rule is the oldest and most famous of all learning rules. It states that "when an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B, is increased."

- This learning can also be called correlational learning.

This statement may be split into a two-part rule:

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

This type of synapse is called Hebbian synapse.

The four key mechanisms that characterize a Hebbian synapse are time dependent mechanism, local mechanism, interactive mechanism and correlational mechanism.

The simplest form of Hebbian learning is described by (4.21)

$$\Delta w = x_i y$$

This Hebbian learning rule represents a purely feed forward, unsupervised learning. It states that if the cross product of output and input is positive, this results in increase of weight, otherwise the weight decreases.

2) Perceptron Learning Rule

For the perceptron learning rule, the learning signal is the difference between the desired and actual neuron's response. This type of learning is supervised.

- The fact that the weight vector is perpendicular to the plane separating the input patterns during the learning processes, can be used to interpret the degree of difficulty of training a perceptron for different types of input.

The perceptron learning rule states that for a finite 'n' number of input training vectors,

$x(n)$ where $n = 1$ to N

each with an associated target value

$t(n)$ where $n = 1$ to N

Which is $+1$ or -1, and an activation function $y - f(y_{in})$, where

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

The weight updation is given by

if $y \neq t$ then

$$W_{new} = W_{old} + tx.$$

if $y = t$, then there is no change in weights.

The perceptron learning rule is of central importance for supervised learning of neural networks. The weights can be initialized at any values in this method.

3) Delta Learning Rule (Widrow-Hoff Rule or Least Mean Square (LMS) Rule)

The delta learning rule is also referred to as Widrow-Hoff rule, named due to the originators (Widrow & Hoff). The delta learning rule is valid only for continuous activation functions and in the supervised training mode. The learning signal for this rule is called delta. The delta rule may be stated as,

~~Delta~~ "The adjustment made to a synaptic weight of a neuron is proportional to the product of the error signal and the input signal of the synapse".

The delta rule assumes that the error signal is directly measurable. The aim of the delta rule is to minimize the error over all training patterns.

Delta rule can be applied for single output unit and several output units. The derivations of these are given below:

a) Delta Rule for Single Output Unit:

The delta rule changes the weight of the connections to minimize the difference between the net input to the output unit, y_{in} and the target value t.

The delta rule is given by

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

Where, x is the vector of activation of input units

y_{in} is the net input to output unit $- \sum x \cdot w_i$

t is the target vector

α - learning rate

The derivation is as follows: - (4.22)

The mean square error for a particular training pattern is

$$E = \sum_j (t_j - y_{-inj})^2$$

The gradient of E is a vector consisting of the partial derivatives of E with respect to each of the weights. The error can be reduced rapidly by adjusting weight w_{ij}

taking partial differentiation w.r.t w_{ij} of E

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_i (t_j - y_{inj})^2 = \frac{\partial}{\partial w_{ij}} (t_j - y_{inj})^2$$

Since the weight w_{ij} influences the error only at output unit y_j.

Also, $$y_{inj} = \sum_{i=1}^{n} (t_j - y_{inj})^2$$

we get $$\frac{\partial E}{\partial w_{ij}} = \frac{E}{\partial w_{ij}} (t_j - y_{inj})^2$$

$$= 2(t_j - y_{inj})(-1)\frac{\partial y_{inj}}{\partial w_{ij}}$$

$$\frac{\partial E}{\partial w_{ij}} = -2(t_j - y_{inj})\frac{\partial y_{inj}}{\partial w_{ij}}$$

$$= -2(t_j - y_{inj})x_i$$

Thus the error will be reduced rapidly depending upon the given learning by adjusting the weights accordingly to the delta rule given by

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

(b) Delta Rule for Several output units.

The derivation of delta rule for several output units is similar to that in prev. section. The weights are changed to reduce the difference between net input and target.

– The weight correction involving delta rule for adjusting the weight from the Ith input unit to the Jth output unit is:

$$\Delta w_{IJ} = \alpha (t_J - y_{in_J}) x_I$$

(c) Extended Delta Rule.

This can also be called as generalized delta rule.

The update for the weight from the Ith input unit to the Jth output unit is,

$$\Delta w_{IJ} = \alpha (t_J - y_{in_J}) x_I f'(y_{in-J})$$

The derivation is as follows:

The squared error for a particular training pattern is:

$$E = \sum_J (t_j - y_j)^2$$ where E is a function of all the weights.

The gradient of E is a vector consisting of the partial derivatives of E w.r.t each of the weights. The error can be reduced rapidly by adjusting the weight w_{IJ} in the direction of

$$\frac{-\partial E}{\partial w_{IJ}}$$

– Differentiating E partially w.r.t w_{IJ},

$$\frac{\partial E}{\partial w_{IJ}} = \frac{\partial}{\partial w_{IJ}} \left(\sum_J (t_j - y_j)^2 \right) = \frac{\partial E}{\partial w_{IJ}} = \frac{\partial}{\partial w_{IJ}} \sum_j (t_j - y_J)^2$$

Since the weight w_{IJ} only influences the error at output unit y_J. Since

$$y_{in J} = \sum_{I=1} x_i w_{iJ}.$$

$$y_J = f(y_{in-J})$$

$$\frac{\partial E}{\partial w_{IJ}} = 2(t_j - y_j)(-1)\frac{\partial y_j}{\partial w_{IJ}}$$

$$= 2(t_j - y_{inj})\frac{\partial f(y_{inJ})}{\partial w_{IJ}}$$

$$\frac{\partial E}{\partial w_{IJ}} = 2(t_j - y_j)\, x\, f^{-1}(y_{inJ})$$

Hence, the error is reduced rapidly for a given learning rate ∞ by adjusting the weights according to the delta rule.

$\Delta w_{IJ} = \infty(t_J - y_J)\, x_I\, f^{-1}(y_{inJ})$ gives the extended delta rule.

(4) Competitive Learning Rule.

In this learning, the output neurons of a neural network compete among themselves to become active. The basic idea behind this rule is that there are a set of neurons that are similar in all aspects except for some randomly distributed synaptic weights, and therefore respond differently to a given set of input patterns. However, a limit is imposed on the strength of the neurons. This rule has a mechanism that permits the neurons to compete for the right to respond to a given subset of inputs, such that only one output neuron, or only one neuron per group, is active at a time. The winner neuron during competition is called winner-takes-all neuron.

For a neuron p to be the winning neuron, its induced local field V_p, for a given particular input pattern must be largest among all the neurons in the network. The output signal of winning neuron is set to one and the signals that lose the competition are set to zero. Hence,

$$N = \begin{cases} 1 & \text{if } V_p > V_q \text{ for all } q,\ p \neq q \\ 0 & \text{otherwise.} \end{cases}$$

This rule is suited for unsupervised network training. The winner-takes-all or the competitive learning is used for learning statistical properties of inputs. This uses the standard Kohonen learning rule.

- Let w_{ij} denote the weight of input node j to neuron i. Suppose the neuron has a fixed weight, which are distributed among its input nodes;

$$\sum_j w_{ij} = 1 \quad \text{for all } i.$$

- A neuron then learns by shifting weights from its inactive to active input nodes. If a neuron does not respond to a particular input pattern, no learning takes place in that neuron. If a particular neuron wins the competition, its corresponding weights are adjusted.
- Using standard competitive rule, the change Δw_{ij} is given as

$$\Delta w_{ij} = \begin{cases} \alpha(x_j - w_{ij}) & \text{if neuron } i \text{ wins the competition} \\ 0 & \text{if neuron } i \text{ loses the competition.} \end{cases}$$

where α is the learning rate. This rule has the effect of moving the weight vector w_i of winning neuron i toward the input pattern x. Through competitive learning, the neural network can perform clustering.

5 Out Star Learning Rule.

Out star learning rule can be well explained when the neurons are arranged in a layer. This rule is designed to produce the desired response t from the layer of n neurons. This type of learning is also called as Grossberg learning.

Out star learning occurs for all units in a particular layer and no competition among these units are assumed. However, the forms of weight updates for Kohonen learning and Grossberg learning are closely related.

In the case of out star learning (4·24)

$$\Delta w_{jk} = \begin{cases} \alpha(y_k - w_{jk}) & \text{if neuron } j \text{ wins the competition} \\ 0 & \text{if neuron } j \text{ losses the} \text{ —} \end{cases}$$

The rule is used to provide learning of repetitive and characteristic properties of input-output relationships. Though it is concerned with supervised learning, it allows the network to extract statistical properties of the input and output signals. It ensures that the output pattern becomes similar to the undistorted desired output after repetitively applying on distorted output versions. The weight change here will be α times the error calculated.

(6) Boltzmann Learning:

The learning is a stochastic learning. A neural net designed based on this learning is called Boltzmann learning. In this learning, the neurons constitute a recurrent structure and they work in binary form. This learning is characterized by an energy function, E, the value of which is determined by the particular states occupied by the individual neurons of the machine, given by,

$$E = -\frac{1}{2} \sum_i \sum_j w_{ij} x_j x_i \quad i \neq j$$

Where x_i is the state of neuron i and w_{ij} is the weight from neuron i to neuron j. The value $i \neq j$ means that none of the neurons in the machine has self feedback. The operation of machine is performed by choosing a neuron at random.

The neurons of this learning process are divided into two groups; visible and hidden. In visible neurons there is an interface between the network and the environment in which it operates but in hidden neurons, they operates independent of the environment. The visible neurons might be clamped onto specific states determined by the environment, called as clamped condition. On the other hand, there is free-running condition, in which all the neurons are allowed to operate freely.

(7) Memory Based Learning.

In memory based learning, all the previous experiences are stored in a large memory of correctly classified input-output examples: $(x_i, t_i)_{i=1}^{N}$ where x_i is the input vector and t_i is the desired response. The desired response is a scalar.

The memory based algorithm involves two parts. They are:

1) Criterion used for defining the local neighborhood of the test vector, and
2) Learning rule applied to the training in the local neighborhood.

There are various algorithms in which these parts with neighborhoods are defined.

One of the most widely used memory based learning is the nearest neighbor rule, where the local neighborhood is defined as the training example that lies in the immediate neighborhood of the test vector x. The vector

$$x_n' \in \{x_1 \ldots x_n\}$$

is said to be nearest neighbor of x_t if, ~~min~~

$$\min d(x_i, x_t) = d(x_n', x_t)$$

where $d(x_i, x_t)$ is the Euclidean distance between the vectors x_i and x_t.

- A variant of nearest neighbor classifier is the K-nearest neighbor classifier, which is stated as
  - Identify the K-classified patterns that is nearest to test vector x_t for some integer K.
  - Assign x_t to the class that is most frequently represented in the K-nearest neighbors to x_t.

Hence K-nearest neighbor classifier acts like an averaging device.

—x—

Hebb Net

(4·25)

The first learning law for artificial neural network was designed by Donald Hebb in 1949.

The law states that if two neurons are activated simultaneously, then the strength of the connection between them should be increased.

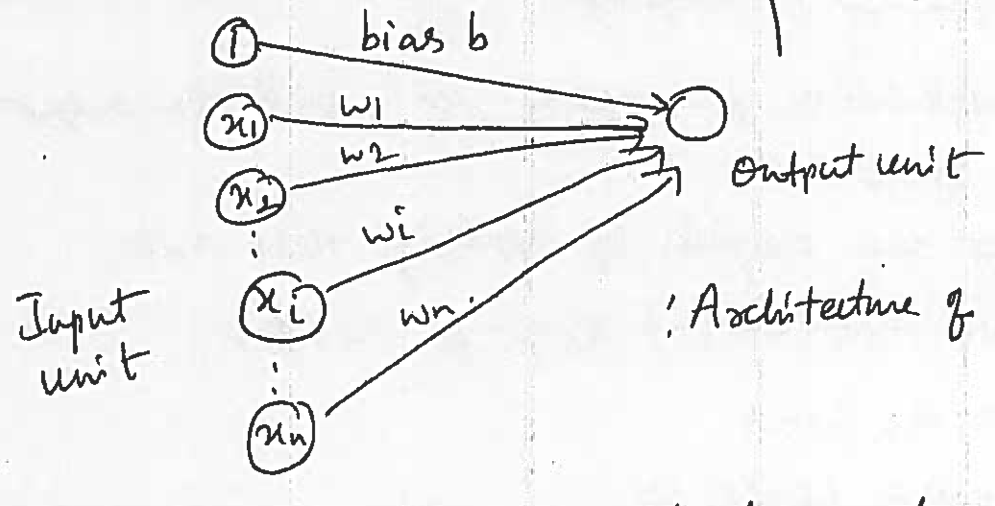
The Hebbian rule developed by McClelland and Rumelhart 1988 formed the Hebb net.

This Hebb net consists of bias which acts exactly as a weight on a connection from a unit whose activation is always 1. If the bias is increased, it increases the net input of the unit.

For Hebb net, the input and the output data should be in bipolar form. If it is in binary form, the Hebb net cannot learn, which is an extreme limitation of the Hebb rule for binary data. The Hebb rule is also used for training other nets.

Hebbian Learning law:
w_{ij} increases only when both i and j are "on"

Architecture



: Architecture of a Hebb Net

The above Fig shows a single layer net, which consists of an input layer with many input units and an output layer with only one output unit. This is the basic architecture that performs pattern classification. The bias included for the net is found to be '1', which helps in increasing the net input.

This architecture resembles a single layer feed forward network.

Algorithm

Initially all the weights and bias are set to zero. Then we can present the input pattern to be classified. At the input layer, the activation function used is identity, hence the output from the input layer remains same as the input presented.

Also, the activation for the output unit is also set. Then, the weights are updated based on the Hebb learning rule. An epoch is completed after presenting all the samples of the input pattern. The step wise algorithm to train Hebb net is as follows:

step1 : Initialize all weights and bias to zero

$w_i = 0$ for $i = 1$ to n. where n is the number of input neurons.

step2 : For each input training vector and target output pair (s, t) perform steps 3-6.

step3 : Set activations for input units with input vector

$x_i = s_i \ (i = 1 \text{ to } n)$

step4 : Set activation for output unit with the output neuron $y = t$.

step5 : Adjust the weights by applying Hebb rule,

$w_i(\text{new}) = w_i(\text{old}) + x_i y$ for $i = 1$ to n.

step 6 : Adjust the bias

$b(\text{new}) = b(\text{old}) + y$

This algorithm requires only one pass through the training set.

Linear separability

17·26

In general, for any output unit, the desired response is '1' if its corresponding input is a member of class or '0' if it is not. The purpose of training is to make the input pattern to get similar with the training pattern by adjusting the weights.

- The activation function is taken as step function. This function retains a high 1 if net input is positive and a low 1 if the net input is negative. The net input to the output neuron is

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

The relation $b + \sum_i x_i w_i = 0$ gives the boundary region of the net input.

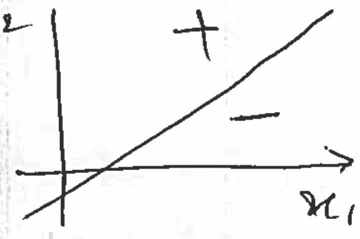
In general, a decision boundary $b + \sum_{i=1}^{n} x_i w_i = 0$ is a $(n-1)$ dimensional hyper-plane in an n dimensional space, which partition the space into two decision regions

- The boundary between the region where $y_{in} > 0$ and $y_{in} < 0$ is called the 'decision boundary'. The equation denoting this decision boundary can represent a line, plane or hyper plane.

- On training, if the weights of training input vectors of correct response +1 lie on the side of the boundary and of the training input vectors of response −1 lie on the other side of the boundary, then the problem is linear separable else it is linearly non-separable.

- Say, with two input vectors, the equation of the line separating the positive region and negative region is given by

$n = 2$ $b \neq 0$



$$b + x_1 w_1 + x_2 w_2 = 0$$

$$x_2 = \frac{-b}{w_2} - x_1 \frac{w_1}{w_2}$$

These two regions are called the decision regions of the net.

1) If a point/pattern (x_1, x_2) is in the positive region, then $b + x_1 w_1 + x_2 w_2 \geq 0$, and the output is one (belongs to class one)
2) otherwise, $b + x_1 w_1 + x_2 w_2 < 0$ output is −1 (belongs

Solved Examples

1) Realise a Hebb net for the AND function with bipolar inputs and targets.

- The AND function gives a high '1' if both the inputs are high else returns a low '−1'
- The training patterns are:

Input x_1	Input x_2	Target B	Target y
1	1	1	1
1	−1	1	−1
−1	1	1	−1
−1	−1	1	−1

- Forming the table, initialize all the weights and the bias to be zero i.e., $w_1 = w_2 = 0$ and $b = 0$
- The weight change is calculated using.

$$\Delta w_i = x_i y \text{ and } \Delta b = y.$$

Input (x_1	x_2	b)	Target y	Weight changes Δw_1	Δw_2	Δb	Weights w_1	w_2	B
						Initial	(0	0	0)
1	1	1	1	1	1	1	1	1	1
1	−1	1	−1	−1	1	−1	0	2	0
−1	1	1	−1	1	−1	−1	1	1	−1
−1	−1	1	−1	1	1	−1	2	2	−2

This completes one epoch of training. The straight line separating the regions can be obtained after presenting each input pair. Thus

$$x_2 = -x_1 \frac{w_1}{w_2} - \frac{b}{w_2}$$

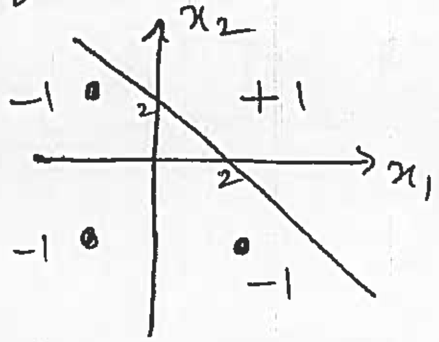
After 1st input $x_2 = -x_1 \frac{1}{1} - \frac{1}{1} = -x_1 - 1$

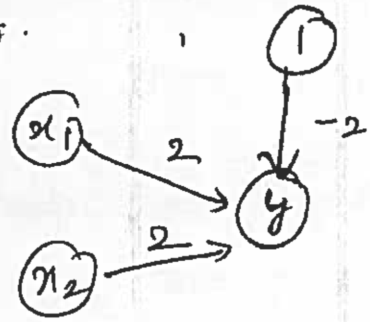
$$x_2 = -x_1 - 1$$

Similarly after 2^{nd}, 3^{rd} and 4^{th} epochs, the separating lines are:

$$x_2 = 0, \quad x_2 = -x_1 + 1 \quad x_2 = -x_1 + 1$$

For the 3^{rd} and 4^{th} epoch the separating line remains the same, hence this line separates the boundary regions as shown in fig.





Hebb net for AND function.

The same procedure can be repeated for generating the logic function OR, NOT, ANDNOT etc.,

---

2) Apply the Hebb net to the training patterns that define XOR function with bipolar input and targets.

soln:

Input x_1	Input x_2	b	Target y
1	1	1	-1
1	-1	1	1
-1	1	1	1
-1	-1	1	-1

By Hebb training algorithm, assigning initial values of the weights w_1 & w_2 to be zero and bias to be zero.

$w_1 = w_2 = 0$ and $b = 0$

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

$$\boxed{b_{new} = b_{old} + \Delta b}$$

Input			Target	Weight changes			Weights		
(x_1	x_2	b)	y	Δw_1	Δw_2	Δb	w_1	w_2	B
							(0	0	0)
1	1	1	-1	-1	-1	-1	-1	-1	-
1	-1	1	1	+1	-1	1	0	-2	0
-1	1	1	1	-1	1	1	-1	-1	1
-1	-1	1	-1	1	1	-1	0	0	0

The weight changes are called using

$$\Delta w_i = x_i y \text{ and } \Delta b = y$$

The new weights by

$$w_{(n)} = w_{(old)} + \Delta w \text{ and } b_{(n)} = b_{(o)} + \Delta b.$$

The final weights obtained for the XOR function are $w_1 = w_2 = 0$ and $b = 0$. Hence it is clear that the separating line cannot be drawn.

Thus, Hebb rule cannot be used to form a training pattern to define XOR function.

---

(3) (a) Using the Hebb rule, find the weights required to perform the following classifications: Vectors (1 1 1 1) and (-1 1 -1 -1) are members of class (with target value 1) Vectors (1 1 1 -1) and (1 -1 -1 1) are not members of class (with target value -1)

(b) Using each of the training x vectors as input, test the response of the net.

Soln The set of patterns are given.

Initially the weights and bias are taken as zero

$w_1 = w_2 = b = 0.$

The weights change calculated by (4.28)

$\Delta w_1 = x_1 y$ and $\Delta b = y$.

– The new weights are obtained using:

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

$b(new) = b(0) + \Delta b$.

(a) On Calculation.

Input					target	Weight Changes					Weights				
x_1	x_2	x_3	x_4	b	y	Δw_1	Δw_2	Δw_3	Δw_4	Δb	w_1	w_2	w_3	w_4	b
											(0	0	0	0	0)
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	−1	1	−1	−1	−1	−1	1	−1	0	0	0	2	0
−1	1	−1	−1	1	1	−1	1	−1	−1	1	−1	1	−1	−1	[illegible]
1	−1	−1	1	1	−1	−1	1	1	−1	−1	−2	2	0	−2	[illegible]

(b) Testing the response of the net

To test the response

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

The activation is bipolar step function, threshold $\theta = 0$

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

The weight vectors are taken from the previous (a) part. They are

(0 0 0 2 0) and (−2 2 0 −2 0)

For 1st input vector (1 1 1 1)

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

$y_{in} = 0+0+0+2+0=2$

Applying activation $y = f(y_{in}) = 2 > 0$. Hence $y=1$

For 2nd input vector $(1 1 1 -1 1)$

$y_{in} = 0+0+0-2+0 = -2 < 0$ Hence $y=-1$

For 3rd input vector $(-1, 1, -1, -1, 1)$

$y_{in} = 2+2+0+2+0 = 6 > 0$ Hence $y=1$

For 4th input vector $(1, -1, -1, 1, 1)$

$y_{in} = -2-2+0-2+0 = -6 < 0$. Hence $y=-1$

Thus for all the input vectors the target output vector equals the target value mentioned. Hence Hebb rule can be used to train this pattern.

End of Unit-4

Unit-5

- Perceptron Networks: Single Layer perceptron, Brief Introduction to Multilayer Perceptron Networks.

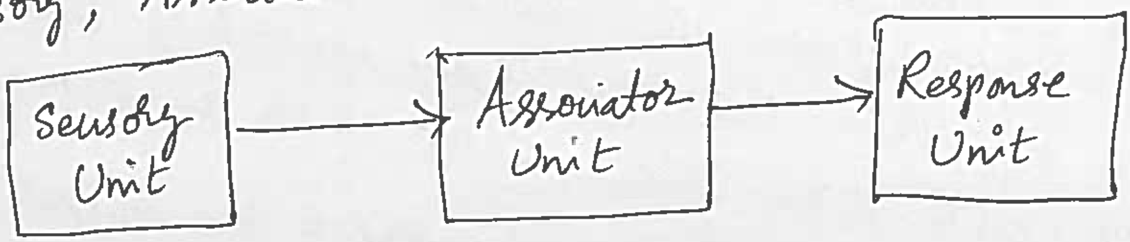
- FeedBack Networks: Introduction, Discrete Hopfield Net, continuous HopField Net.

- Feed Forward Networks: Back propagation Network, Radial Basis Function Network.

- Self organizing Feature Map: Methods used for determining the Winner, Kohonen Self organizing Feature maps, Learning Vector Quantization, MaxNet, Mexican Hat, Hamming Net.

Perceptron Networks

- Frank Rosenblatt [1962] and Minsky and Papert [1988], developed large class of artificial neural networks called Perceptrons.
- The perceptron learning rule uses an iterative weight adjustment that is more powerful than the Hebb rule.
- The perceptrons use threshold output function and the McCulloch-pitts model of a neuron.
- Their iterative learning converges to correct weights, i.e., the weights that produce the exact output value for the training input pattern.
- The original perceptron is found to have three layers, Sensory, Associator and Response Units. (Fig)

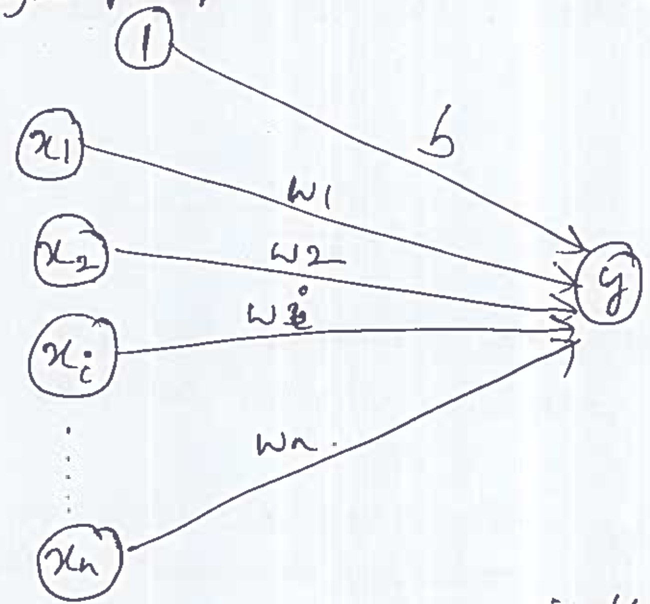


Original perceptron.

 The Sensory and association units have binary activations and an activation of +1, 0 or −1 is used for the response unit. All the units have their corresponding weighted interconnections.

- Training in perceptron will continue until no error occurs.
- This Net solves the problem and is used to learn the classification.
- The perceptrons are of two types:
  1) Single Layer 2) Multi-layer perceptrons.

1) Single Layer perceptron.



Architecture of single layer perceptron.

- A single layer perceptron is the simplest form of a neural network used for the classification of patterns that are linearly separable.
- It consists of a single neuron with adjustable weights and bias.
- In the fig, only the associator unit and the response unit is shown. The sensor unit is hidden, because only the weights between the associator and the response unit are adjusted. The input layer consists of input neurons from $x_1, x_2 \ldots x_n$. There always exists a common bias of 1.
- The input neurons are connected to the output neurons through weighted interconnections.

This is a single layer network because it has only one layer of interconnections between the Input and the Output neurons. This network perceives the input signal received and performs the classification. (5.2)

Algorithm

To start the training process, initially the weights and the bias are set to zero. The initial weights of the network can be formulated from other techniques like Fuzzy systems, Genetic algorithm etc.

- It is also essential to set the learning rate parameter, which ranges between 0 to 1. Then the Input is presented.
- The net input is calculated by multiplying the weights with the inputs and adding the results with the bias entity.
- Once the net input is calculated, by applying the activation function, the output of the network is also obtained.
- This output is compared with the target, where if any difference occurs, we go in for weight updation based on perceptron learning rule, else the network training is stopped.
- The algorithm can be used for both binary and bipolar Input vectors. It uses a bipolar target with fixed threshold and adjustable bias.
- The training algorithm is as follows:

step1 : Initialize weights and bias (initially it can be zero). set learning rate α (0 to 1)

step2 : While stopping condition is false do steps 3-7.

step3 : For each training pair $s:t$ do steps 4-6.

step4 : set activation of input units

$$x_i = s_j \text{ for } i = 1 \text{ to } n.$$

Step 5 : Compute the output unit response

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

The activation function used is

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

Step 6 : The weights and bias are updated if the target is not equal to the output response.

If $t \neq y$ and the value of x_i is not zero

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

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

else

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

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

Step 7 : Test for stopping condition.

The stopping conditions may be the weight changes.

Note :

1. Only weights connecting active input units ($x_i \neq 0$) are updated.
2. Weights are updated only for patterns that do not produce the correct value of y.

Application procedure.

This procedure enables the user to test the network performance. The network should be trained with sufficient number of training data and using the testing data its performance can be tested.

- The application procedure used for testing perceptron network is as follows:

step1: The weights to be used here are taken from the training algorithm.

step2: For each input vector x to be classified do steps 3-4.

step3: Input units activations are set

step4: Calculate the response of output unit

$$y_{-in} = \sum_i x_i w_i$$

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

Perceptron Algorithm for Several Output Classes

- The perceptron network for single output class is extended for several output classes. Here there exist more number of output neurons, but the weight updation in this case also is based on the perceptron learning rule. The algorithm is as follows:

step1: Initialize the weights and biases. Set the learning rate

step2: When stopping condition is false, perform steps 3-7

step3: For each input training pair, do steps 4-6

step4: Set activation for the input units

$x_i = S_i$ for $i = 1$ to n.

 Step 5 : Compute the activation output of each output unit

$$y\text{-}in_j = b_j + \sum_i x_i w_i \quad \text{for } j = 1 \text{ to } m.$$

$$y_j = f(y\text{-}in_j) = \begin{cases} 1 & \text{if } y\text{-}in_j > \theta \\ 0 & \text{if } -\theta \le y\text{-}in_j \le \theta \\ -1 & \text{if } y\text{-}in_j < -\theta \end{cases}$$

Step 6 : The weights and bias are to be updated for $j = 1$ to m and $i = 1$ to n.

If $y_j \neq t_j$ and $x_i \neq 0$ then

$$w_{ij}(new) = w_{ij}(old) + \alpha t_j x_i$$

$$b_j(new) = b_j(old) + \alpha t_j$$

else if $y_j = t_j$

$$w_{ij}(new) = w_{ij}(old)$$

$$b_j(new) = b_j(old)$$

that is, the biases and weights remain unchanged.

Step 7 : Test for stopping condition.

The stopping condition may be the weight changes.

Example : Develop a perceptron for the AND function with bipolar inputs and targets.

Soln : The training pattern for AND function can be

Input			Target
x_1	x_2	b	t
1	1	1	1
-1	1	1	-1
1	-1	1	-1
-1	-1	1	-1

 (2.4)

Step 1: Initial weights $w_1 = w_2 = 0$ and $b = 0$, $\alpha = 1$, $\theta = 0$

Step 2: Begin computation

Step 3: For input pair $(1, 1) : 1$, do steps 4–6

Step 4: Set activations of input units

$$x_i = (1, 1)$$

Step 5: Calculate the net input

$$y_{-in} = b + \sum x_i w_i = 0 + 1 \times 0 + 1 \times 0 = 0$$

Applying the activation

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

Therefore $y = 0$

Step 6: $t = 1$ and $y = 0$

Since $t \neq y$, the new weights are

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

$$w_1(new) = 0 + 1 \times 1 \times 1 = 1$$

$$w_2(n) = w_2(o) + \alpha t x_2 = 0 + 1 \times 1 \times 1 = 1$$

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

$$b(n) = b(o) + \alpha t = 0 + 1 \times 1 = 1$$

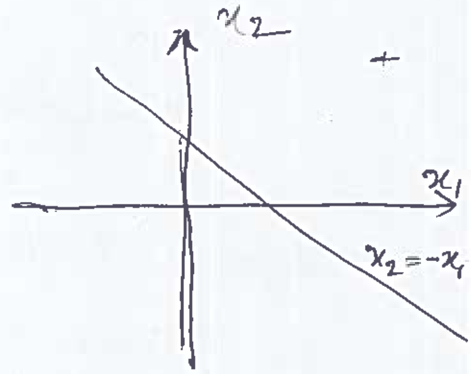
The new weights and bias are $[1\ 1\ 1]$

The algorithm steps are repeated for all the input vectors with their initial weights as the previously calculated weights.

By presenting all the input vectors, the updated weights are shown in table below:

Input			Net	Output	Target	Weight changes			Weights		
x_1	x_2	B	y_{in}	y	t	Δw_1	Δw_2	Δb	w_1	w_2	B
									(0	0	0)
1	1	1	0	0	1	1	1	1	1	1	1
-1	1	1	1	1	-1	1	-1	-1	2	0	0
1	-1	1	2	1	-1	-1	1	-1	1	1	-1
-1	-1	1	-3	-1	-1	0	0	0	1	1	-1

This completes one epoch of the training.
the final weights after the first epoch is completed are.
$w_1 = 1,\ w_2 = 1,\ b = -1$



We know that $b + x_1 w_1 + x_2 w_2 = 0.$

$$x_2 = -x_1 \cdot \frac{w_1}{w_2} - \frac{b}{w_2}$$

$$x_2 = -x_1 \cdot \frac{1}{1} - \frac{(-1)}{1}$$

$x_2 = -x_1 + 1$ is the separating line equation.

Ex (2): Develop a perceptron for the AND function with binary inputs and bipolar targets without bias upto 2 epochs. (Take first with (0,0) and next without (0,0))

Soln: Initializing the weights to be $w_1 = w_2 = 0$ and the bias is neglected here (because the problem is stated without bias). Hence $\alpha = 1$ and threshold $\theta = 0$.

(a) with (0,0) and without bias

The net input is $Y_{in} = \sum x_i w_i$

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

65 The weight change

$\Delta w_i = \alpha t x_i$ and

new weight is

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

(5.5)

$w_1 = w_2 = 0$

$y_{in} = \sum x_i w_i$

$y_1 = 1\times 0 + 1\times 0 = 0$
$y_2 = 1\times 0 + 0\times 0 = 0$
$y_3 = 0\times 0 + 1\times 0 = 0$
$y_4 = 0\times 0 + 0\times 0 = 0.$

Epoch 1:

Input		Net	Output	Target	Weight Changes		Weights	
x_1	x_2	y_{in}	y	t	Δw_1	Δw_2	w_1 (0	w_2 0)
1	1	0	0	1	1	1	1	1
1	0	1	1	−1	−1	0	0	1
0	1	1	1	−1	0	−1	0	0
0	0	0	0	−1	0	0	0	0

The separating line for 1st and 2nd input are $x_1 + x_2 = 0$ and $x_2 = 0$ respectively.

Epoch 2:

The initial weights used here are the final weights from the previous iteration.

Input		Net	Output	Target	Weight changes		Weights	
x_1	x_2	y_{in}	y	t	Δw_1	Δw_2	w_1 (0	w_2 0)
1	1	0	0	1	1	1	1	1
1	0	1	1	−1	−1	0	0	1
0	1	1	1	−1	0	−1	0	0
0	0	0	0	1	0	0	0	0

Without bias for the given inputs, the final weights obtained are same as that for with bias and the equation of separating line also remains same. Thus the equations remain same & are given.

for 1st input : $x_1 + x_2 = 0$

for 2nd input : $x_2 = 0$.

b) without bias and (0,0) without.

Epoch 1 :

Input		Net	Output	Target	Weight Changes		Weights	
x_1	x_2	y_{in}	y	t	Δw_1	Δw_2	w_1 (0	w_2 0)
1	1	0	0	1	1	1	1	1
1	0	1	1	−1	−1	0	0	1
0	1	1	1	−1	0	−1	0	0

In this case, also the final weights are (0,0) and the separating line are $x_2 = -x_1$ and $x_2 = 0$

Epoch 2

The final weights from Epoch 1 are used here as initial weights :

Input		Net	Output	Target	Weight Changes		Weights	
x_1	x_2	y_{in}	y	t	Δw_1	Δw_2	w_1 (0	w_2 0)
1	1	0	0	1	1	1	1	1
1	0	1	1	−1	−1	0	0	1
0	1	1	1	−1	0	−1	0	0

Here also the weights are same as that of previous Epoch. The separating line here also, without bias

1st input:

$$x_2 = -x_1 \frac{w_1}{w_2} = -x \cdot \frac{1}{1} = -x_1$$

$$x_2 = -x_1$$

2nd input:

$$x_2 = -x_1 \frac{w_1}{w_2} = -x_1 \cdot \frac{0}{1} = 0$$

$$x_2 = -0$$

Thus from all this, it is clear that without bias the convergence does not occur. Even after neglecting (0,0) the convergence does not occur.

Example 3:

Using the perceptron learning rule, find the weights required to perform the following classifications. Vectors (1 1 1 1), (−1 1 −1 −1) and (1 −1 −1 1) are members of class (having target value 1);

Vectors (1 1 1 −1) and (1 −1 −1 1) are not members of class (having target value −1).

- Use learning rate of 1 and starting weights of 0.
- Use each of the training and vectors as input, test the response of the net.

soln: The initial weights are assumed to be zero and the learning rate as 1.

The updation is done according to perceptron learning rule.

If $y \neq t$, weight change, $\Delta w = \alpha t x_i$ & $\Delta b = \alpha t$

New weights are $w(new) = w(old) + \Delta w$

$b(new) = b(old) + \Delta b$.

If $t = y$, no weight change.

By using the above, the below tabulation is formed,

where $y_{-in} = b + \sum_i x_i w_i$

and $y = f(y_{in})$ is the activation applied.

Input x_1 x_2 x_3 x_4 1	Net y_{in}	Output y	Target t	Weight changes Δw_1 Δw_2 Δw_3 Δw_4 Δb	Weights w_1 w_2 w_3 w_4
					(0 0 0 0
Epoch 1:					
1 1 1 1 1	0	0	1	1 1 1 1 1	1 1 1 1
1 1 1 −1 1	3	1	−1	−1 −1 −1 1 −1	0 0 0 2
−1 1 −1 −1 1	0	0	1	−1 1 −1 −1 1	−1 1 −1 −1
1 −1 −1 1 1	−1	−1	−1	0 0 0 0 0	−1 1 −1 −1
Epoch 2:				Initial →	0 0 0 2
1 1 1 1 1	2	1	1	0 0 0 0 0	0 0 0 2
1 1 1 −1 1	−2	−1	−1	0 0 0 0 0	0 0 0 2
Initial →					−1 1 −1 −1
−1 1 −1 −1 1	5	1	1	0 0 0 0 0	−1 1 −1 −1 1
1 −1 −1 1 1	−1	−1	−1	0 0 0 0 0	−1 1 −1 −1 1

The final weights from Epoch 1 are used as the initial weights for Epoch 2. Thus the output is equal to target by training for suitable weights.

Testing the response of the net.

The final weights are:

For the 1^{st} set of input

$w_1 = 0 \quad w_2 = 0 \quad w_3 = 0 \quad w_4 = 2 \quad b = 0$ and

For the 2^{nd} set of input

$w_1 = -1 \quad w_2 = 1 \quad w_3 = -1 \quad w_4 = -1 \quad b = 1$

The net input is

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

For the 1^{st} set of inputs

1) $(1\ 1\ 1\ 1\ 1)$ with $x_1\ x_2\ x_3\ x_4\ x_5$

$$y_{-in1} = 0 + 0\times1 + 0\times1 + 0\times1 + 2\times1 = 2 > 0$$

Applying activation.

$$y_1 = f(y_{-in1}) = 1$$

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

2) $(1\ 1\ 1\ -1\ 1)$

$$y_{-in2} = 0 + 0\times1 + 0\times1 + 0\times1 - 1\times2 + 0\times1 = -2 < 0$$

Applying activation

$$y_2 = f(y_{-in2}) = -1.$$

* Note :: Hence the calculated test output values matches with the target output values for the corresponding input vectors

For 2^{nd} set of inputs.

1) $(-1\ 1\ -1\ -1\ 1)$

$$y_{-in1} = 1 + -1\times-1 + 1\times1 + -1\times-1 + -1\times-1 + 1\times1 = 5 > 0.$$

Applying activation $\quad y_1 = f(y_{-in1}) = 1.$

2) $(1\ -1\ -1\ 1\ 1)$

$$y_{-in2} = 1 + -1\times1 + 1\times-1 + -1\times-1 + -1\times1 + -1\times1 = -1 < 0$$

Applying activation $\quad y_2 = f(y_{-in2}) = -1.$

Example 4

For the following noisy versions of training patterns, identify the response of network by seggregating it into correct, incorrect or indefinite.

(0 -1 1), (0 1 -1), (0 0 1), (0 0 -1), (0 1 0), (1 0 1), (1 0 -1), (1 -1 0), (1 0 0), (1 1 0), (0 -1 0), (1 1 1)

Soln The concept used for this problem is

if $x_1 w_1 + x_2 w_2 + x_3 w_3 > 0$, then the response is correct

if $x_1 w_1 + x_2 w_2 + x_3 w_3 < 0$, — " — incorrect

if $x_1 w_1 + x_2 w_2 + x_3 w_3 = 0$ then " — indefinite

Say if the weights taken from the bipolar step functions are $w_1 = 0$, $w_2 = -2$ and $w_3 = 2$

For (0 -1 1) $x_1 = 0$ $x_2 = -1$ $x_3 = 1$.

Vector $(x_1\ x_2\ x_3)$	$x_1 w_1 + x_2 w_2 + x_3 w_3$	Response.
(0 -1 1)	4	correct
(0 1 -1)	-4	incorrect
(0 0 1)	2	correct
(0 0 -1)	-2	incorrect
(0 1 0)	-2	Incorrect
(1 0 1)	2	correct
(1 0 -1)	-2	Incorrect
(1 -1 0)	2	correct
(1 0 0)	-0	indefinite
(1 1 0)	-2	Incorrect
(0 -1 0)	2	correct
(1 1 1)	0	indefinite

Brief Introduction to Multilayer Perceptron Networks. (5.8)

- Multilayer perceptron networks is an important class of neural networks.
- The network consists of a set of sensory units that constitute the input layer and one or more hidden layer of computation nodes.
- The input signal passes through the network in the forward direction. The network of this type is called multilayer perceptron (MLP).
- The multilayer perceptrons are used with supervised learning and have led to the successful backpropagation algorithms. The disadvantage of the single layer perceptron is that it cannot be extended to multi-layered version.
- In MLP networks there exists a non-linear activation function. (ex logistic sigmoidal function).
- The MLP network has various layers of hidden neurons. The hidden neurons make the MLP network active for highly complex tasks. The layers of the network are connected by synaptic weights.
- The MLP thus has a high computational efficiency.
- The disadvantage of MLP may also be the presence of non-linearity and complex connections of the network which leads to highly complex theoretical analysis. Also the existance of hidden neurons makes the learning process tedius.
- The MLP networks are usually fully connected networks
- There are various multilayer perceptron networks which includes Backpropagation network, Radial basis function network etc.,

——x——

Feed Back Networks

- So far we have seen, one output vector was assigned to every input vector. (Feed forward flow of information).
- But there exist situations, in which one can return back the output to the input, thereby giving rise to an iteration process. These types of networks come under the category of feedback networks.
- Feedback networks include simulated annealing, Boltzmann machine, Hopfield net etc.,

Discrete Hopfield Net

- This type of network was described by J.J. Hopfield in 1982. Hopfield while working on the magnetic behavior of the solids (spin glasses) described property of magnetic atoms using two states (1 and −1)
- The topology of Hopfield network is very simple: It has 'n' neutrons, which are all networked with each other.
- A Hopfield network is able to recognise unclear pictures ~~the~~ correctly. However, only one picture can be stored at a time.
- The discrete Hopfield net is a fully interconnected neural net with each unit connected to every other unit. The net has symmetric weights with no self connections. ie, all the diagonal elements of the weight matrix of a Hopfield net are zero.

$$w_{ij} = w_{ji} \quad \text{and} \quad w_{ii} = 0$$