Soft Computing
Paper: MTM 402 (Unit-2)
Artificial Neural Network

Dr. Raghu Nandan Giri

Department of Applied Mathematics

April 3, 2020

- Artificial neural networks (ANNs) are biologically inspired computer programs designed to simulate the way in which the human brain processes information.
- An ANN is formed from hundreds of single units, artificial neurons or processing elements, connected with coefficients (weights), which constitute the neural structure and are organized in layers.
- The power of neural computations comes from connecting neurons in a network.
- The behavior of a neural network is determined by the transfer functions of its neurons, by the learning rule, and by the architecture itself.

- In 1943, McCulloch and Pitts first modeled a simple neural network using electrical circuits in order to describe how neurons in the brain might work.
- In 1949, first learning law for artificial neural networks was designed by Donald Hebb.
- In 1958, a learning method for McCulloch and Pitts neuron model named Perceptron was invented by Rosenblatt.
- In 1960, Widrow and Hoff developed models called "ADALINE" and "MADALINE".
- In 1961, Rosenblatt made an unsuccessful attempt but proposed the "backpropagation" scheme for multilayer networks.
- In 1969, Multilayer perceptron (MLP) was invented by Minsky and Papert.

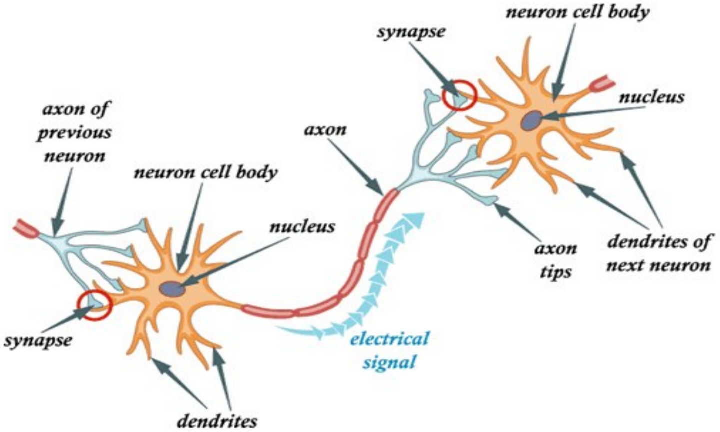


Fig. 1: Neuron cell.

- The term ‘Neural’ is derived from the basic functional unit ‘neuron’ of human (animal) nervous system.
- Brain contains approximate 10^{11} neurons, each of which has 10^2–10^5 connections with other neurons.
- Neurons are organized in a fully connected network and act like messenger in receiving and sending impulses.
- The connections can be inhibitory (decreasing strength) or excitatory (increasing strength) in nature.
- The result is an intelligent brain capable of learning, prediction and recognition.
- Neurons consist of four basic parts–Dendrite, Soma (cell body), Axon, Synapse.

- **Dendrite:** They are tree-like branches, responsible for receiving the information from other neurons. In other sense, we can say that they are like the ears of neuron.
- **Soma (cell body):** It sums all the incoming signals have received from dendrites.
- **Axon:** When the sum reaches a threshold value, neuron fires and the signal travels down the axon to the other neurons i.e. it is just like a cable through which neurons send the information.
- **Synapse:** The point of interconnection of one neuron with other neurons. The amount of signal transmitted depend upon the strength (synaptic weights) of the connections.

ANNs are digitized model of a human brain i.e. a biologically inspired computational model to simulate the way in which human brain processes information. It is formed from hundreds of single units, artificial neurons, connected with coefficients (weights) which constitute the neural structure. They are also known as processing elements as they process information. Each processing element has weighted inputs, transfer function and one output. Processing element is essentially an equation which balance inputs and outputs. ANNs learn (or are trained) through experience with appropriate learning example just like people do, not from programming.

Comparison between Artificial and Biological Neural Network

The following table shows the comparison between ANN and BNN based on some criteria mentioned.

Criteria	**BNN**	**ANN**
Processing	Massively parallel, slow but superior than ANN	Massively parallel, fast but inferior than BNN
Size	10^{11} neurons and 10^{15} interconnections	10^{2} to 10^{4} nodes (mainly depends on the type of application and network designer)
Learning	They can tolerate ambiguity	Very precise, structured and formatted data is required to tolerate ambiguity
Fault tolerance	Performance degrades with even partial damage	It is capable of robust performance, hence has the potential to be fault tolerant
Storage capacity	Stores the information in the synapse	Stores the information in continuous memory locations

The advantages of artificial neural networks include:

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

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

c. **Real Time Operation:** ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.

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

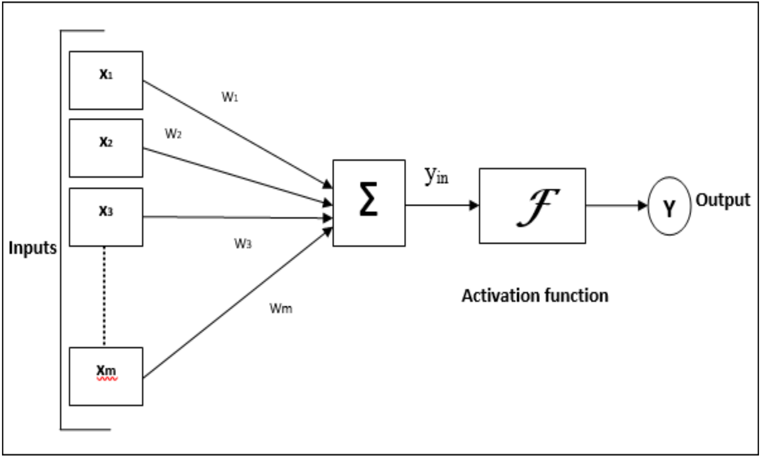


Fig. 2: Artificial Neural Network.

Artificial neural networks can be viewed as weighted directed graphs in which artificial neurons are nodes and directed edges with weights are connections between neuron outputs and neuron inputs shown in Fig. 2. It receives input from the external world in the form of pattern and image in vector form. These inputs are mathematically designated by the notation $x_1, x_2, \ldots, x_m$ for m number of inputs with their corresponding weights $w_1, w_2, \ldots, w_m$ respectively. Weights are the information used by the neural network to solve a problem. Typically weight represents the strength of the interconnection between neurons inside the neural network. The weighted inputs are all summed up inside computing unit (artificial neuron). For the general model of artificial neural network shown in Fig. 2, the net input can be calculated as follows:

$$y_{in} = x_1.w_1 + x_2.w_2 + x_3.w_3 + \ldots + x_m.w_m = \sum_{i=1}^{m} x_i.w_i$$

In case the weighted sum is zero, bias is added to make the output non-zero or to scale up the system response. Bias has the weight and input always equal to '1'

$$y_{in} = \sum_{i=1}^{m} x_i.w_i + b = \sum_{i=0}^{m} x_i.w_i, \text{ where } w_0 = b, \; x_0 = 1.$$

The sum corresponds to any numerical value ranging from 0 to ∞. In order to limit the response to arrive at desired value, the threshold value is set up. For this, the sum is passed through activation function. The output can be calculated by applying the activation function over the net input y_{in} as $y = f(y_{in})$. The activation function is set of the transfer function used to get desired output. There are linear as well as the non-linear activation function.

There have been many types of neural networks designed but all can be described by depends upon the following three building blocks -

- Network Topology
- Adjustments of Weights or Learning
- Activation Functions

Network Topology

The main architectures of artificial neural networks, considering the neuron disposition, as well as how they are interconnected and how its layers are composed, can be divided as follows: (i) Single-Layer Feedforward Networks, (ii) Multilayer Feedforward Networks, (iii) Recurrent Networks.

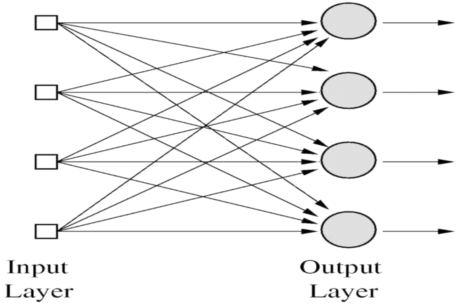


Fig. 3: Single-Layer Feedforward Networks.

Single-Layer Feedforward Networks

In this network, the neurons are organized in the form of layers. Its simplest form, we have an input layer of source nodes that projects directly onto an output layer of neurons (computation nodes), but not vice versa. In other words, this network is strictly of a feedforward type. It is illustrated in Fig. 3 for the case of four nodes in both the input and output layers. Such a network is called a single-layer network, with the designation “single-layer” referring to the output layer of computation nodes (neurons). We do not count the input layer of source nodes because no computation is performed there.

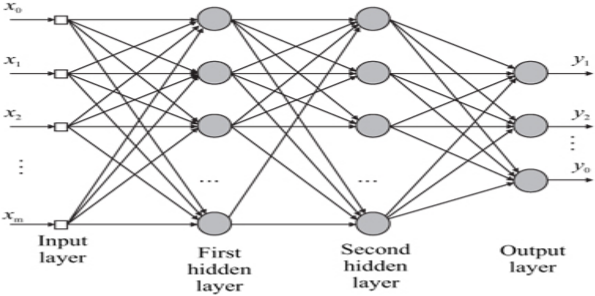


Fig. 4: Multilayer Feedforward Networks.

Multilayer Feedforward Networks

The second class of a feedforward neural network shown in Fig. 4 distinguishes itself by the presence of one or more hidden layers, whose computation nodes are correspondingly called hidden neurons or hidden units; the term "hidden" refers to the fact that this part of the neural network is not seen directly from either the input or output of the network. The function of hidden neurons is to interfere between the external input and the network output in some useful manner. The neurons in each layer of the network have as their inputs the output signals of the preceding layer only. The set of output signals of the neurons in the output (final) layer of the network constitutes the overall response of the network to the activation pattern supplied by the source nodes in the input (first) layer. A feedforward network with m source nodes, l neurons in the first hidden layer, p neurons in the second hidden layer, and q neurons in the output layer is referred to as an m-l-q-q network.

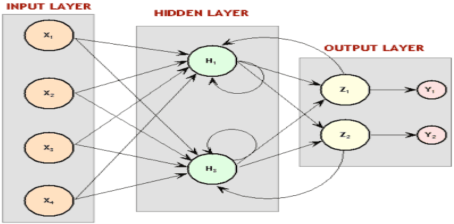


Fig. 5: Recurrent Networks.

Recurrent Networks

A recurrent neural network distinguishes itself from a feed-forward neural network in that it has at least one feedback loop. It may consist of a single layer of neurons with each neuron feeding its output signal back to the inputs of all the other neurons shown in Fig. 5. The idea behind recurrent neural network is to make use of sequential information. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. But for many tasks that's a very bad idea. If you want to predict the next word in a sentence you better know which words came before it. Recurrent neural networks are called recurrent because they perform the same task for every element of a sequence with the output being depended on the previous computations. Another way to think about recurrent neural network is that they have a "memory" which captures information about what has been calculated so far. In theory, recurrent neural network can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps.

Learning Processes

There are many different algorithms that can be used when training artificial neural networks, each with their own separate advantages and disadvantages. The learning process within artificial neural networks is a result of updating the network's weights, with some kind of learning algorithm. Just as there are different ways in which we learn from our surrounding environments, so it is with neural networks. In a broad sense, we may categorize the learning processes as follows: (i). Supervised Learning (ii). Unsupervised Learning (iii). Reinforcement Learning.

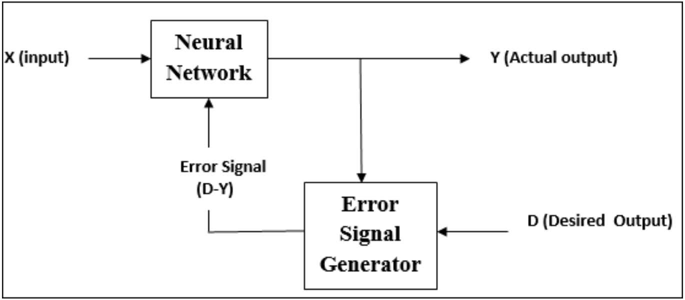
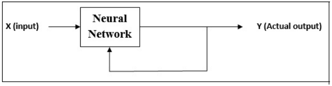


Fig. 6: Supervised Learning.

Supervised Learning

As the name suggests, this type of learning is done under the supervision of a teacher. During the training of ANN under supervised learning, the desired output for the network is provided with the input shown in Fig. 6. By providing the neural network with both an input and output pair it is possible to calculate an error based on it's target output and actual output. Then, the network parameters (weights) are adjusted by a combination of the input and the corresponding error. So, supervised learning is a closed-loop feedback system, where the error is the feedback signal. The trained network is used to emulate the system.



Unsupervised Learning

Unsupervised learning shown in Fig. 7 involves no target values and learning is done without the supervision of a teacher. It tries to autoassociate information from the inputs with an intrinsic reduction of data dimensionality or total amount of input data. Unsupervised learning is solely based on the correlations among the input data, and is used to find the significant patterns or features in the input data. Particularly suitable for biological learning in that it does not depend on a teacher and it uses intuitive primitives like neural competition and cooperation.

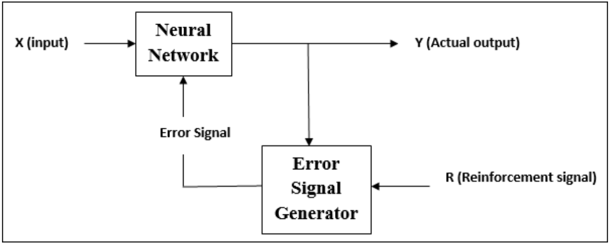


Fig. 8: Reinforcement Learning.

Reinforcement Learning

Reinforcement learning shown in Fig. 8 is a special case of supervised learning, where the exact desired output is unknown. It is based only on the information as to whether or not the actual output is close to the estimate. This learning procedure rewards the neural network for its good output result and punishes it for the bad output result. The aim of reinforcement learning is to maximize the reward the system receives through trial-and-error. It is used in the case when the correct output for an input pattern is not available and there is need for developing a certain output.
There are many types of Neural Network Learning Rules, they based on two supervised learning, and unsupervised learning processes.

Hebbian Learning Rule

This rule, one of the oldest and simplest, was introduced by Donald Hebb in his book 'The Organization of Behavior' in 1949. It is a kind of feed-forward, unsupervised learning. This rule is that the connections between two neurons might be strengthened if the neurons fire at the same time and might weaken if they fire at different times. According to Hebbian learning rule, following is the formula to increase the weight of connection at every time step.

$$\Delta w_{ij}(t) = \eta x_i(t).y_j(t)$$

Here, $\Delta w_{ij}(t)$= increment by which the weight of connection increases at time step t, η = the positive and constant learning rate, $x_i(t)$ = the input value from pre-synaptic neuron at time step t, $y_i(t)$ = the output of pre-synaptic neuron at same time step t.

This learning rule required the weight initialization at small random values around 0. When inputs of both the nodes are either positive or negative, then a strong positive weight exists between the nodes. If the input of a node is positive and negative for other, a strong negative weight exists between the nodes.

Perceptron Learning Rule

This rule is an error correcting the supervised learning algorithm of single layer feedforward networks with hard-limit activation function, introduced by Rosenblatt. The basic concept is to calculate the error by comparison between the desired/target output and the actual output. If there is any difference found, then a change must be made to the weights of connection. According to perceptron learning rule, following is the formula to update the weight of each connection.

$$\Delta w_i = \eta(t - y)x_i$$

where η is positive constant called the learning rate, y is actual output of the neuron and t is the desired/target output.
Comments about the perceptron learning rule:

- If the example is correctly classified the term $(t - y)$ equals zero, and no update on the weight is necessary.
- If the perceptron outputs 1 and the real answer is 1, the weight is increased.
- If the perceptron outputs a 1 and the real answer is -1, the weight is decreased.
- Provided the examples are linearly separable and a small value for η is used, the rule is proved to classify all training examples correctly (i.e, is consistent with the training data).

Delta Learning Rule

It is introduced by Bernard Widrow and Marcian Hoff, also called Least Mean Square (LMS) method, to minimize the error over all training patterns. It is kind of supervised learning algorithm with having continuous activation function. The base of this rule is gradient-descent approach, which continues forever. Delta rule updates the synaptic weights so as to minimize error between the target output and the actual output. To update the synaptic weights, delta rule is given by

$$\Delta w_i = \eta(t - y) f'(y_{in}) x_i$$

The delta rule is commonly stated in simplified form for a neuron with a linear activation function as $\Delta w_i = \eta(t - y)x_i$.
Note: Delta rule is similar to the perceptron learning rule, with some differences: (i) Error in perceptron learning rule is restricted to having values of 0, 1 or -1 but in delta rule may have any value. (ii). Delta rule can be derived for any differentiable output/activation function f, whereas in perceptron learning rule only works for step activation function.

Activation Functions

The basic processing element, neuron, of ANN apply a nonlinear mapping (not necessarily linear) called an activation function before delivering the output to the next neuron. Depending on the problem at hand and on the location of the node within a given layer, the activation functions can take different forms.

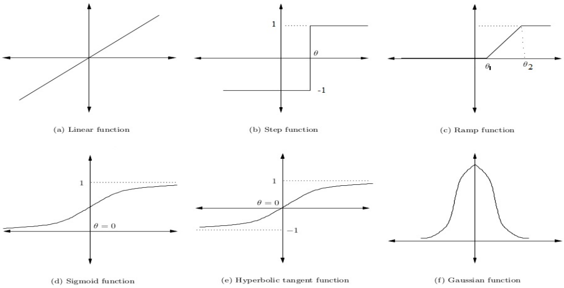


Fig. 9: Activation Functions.

Activation Functions

Let us denote the weighted sum of input into a neuron by y_{in}. The activation or output of the neuron, y, is then given by applying the activation (or transfer) function, f, to y_{in}: $y = f(y_{in})$. Some typical choices are given below.
Linear or Identity Function: If f is an identity function,

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

As we will see later, identity activation functions are used for neurons in the input layer of an ANN.
Binary Step Function with Threshold:

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

The threshold value is specified by θ. The output has binary values (0 or 1) only.
Bipolar Step Function with Threshold:

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

The threshold value is specified by θ. The output has bipolar values (1 or -1) only.
Ramp Function:

$$y = f(y_{in}) = \begin{cases} 0 & if \quad y_{in} \leq \theta_1, \\ \frac{y_{in} - \theta_1}{\theta_2 - \theta_1} & if \quad \theta_1 \leq y_{in} \leq \theta_2, \\ 1 & if \quad y_{in} > \theta_2. \end{cases}$$

Activation Functions

The ramp function is a truncated version of the linear function. From its shape, the ramp function looks like a more definitive version of the sigmoid function in that its maps a range of inputs to outputs over the range [0, 1] but this time with definitive cut off points θ_1 and θ_2.

Binary Sigmoid Function:

$$y = f(y_{in}) = \frac{1}{1 + e^{-\alpha(y_{in} - \theta)}},$$

where α is a positive parameter. This function switches from 0 to 1 in the vicinity of θ as the argument goes from $-\infty$ to $+\infty$.

Bipolar Sigmoid Function:

$$y = f(y_{in}) = \frac{1 - e^{-\alpha(y_{in} - \theta)}}{1 + e^{-\alpha(y_{in} - \theta)}},$$

where α is a positive parameter. This function switches from -1 to 1 in the vicinity of θ as the argument goes from $-\infty$ to $+\infty$.

Gaussian function:

$$y = f(y_{in}) = e^{-\frac{(y_{in} - \theta)^2}{2\sigma^2}}$$

The maximal function value of a gauss function is found for θ activation. The function is even: f(-x)=f(x).

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. If a bias is included, the activation function is typically taken to be

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

where,

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

Some authors do not use a bias weight, but instead use a fixed threshold θ for the activation function. In that case,

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

where,

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

However, as the next example will demonstrate, this is essentially equivalent to the use of an adjustable bias.

The role of a bias or a threshold

In this example, we consider the separation of the input space into regions where the response of the net is positive and regions where the response is negative. To facilitate a graphical display of the relationships of two input neurons and one output neuron, the architecture is given in Fig. 10.

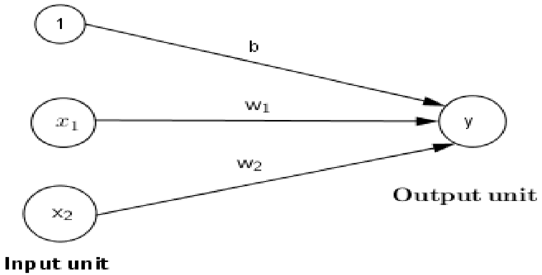


Fig. 10: Single-layer neural network for logic function.

The role of a bias or a threshold

The boundary between the values of x_1 and x_2 for which the net gives a positive response and the values for which it gives a negative response is the separating line

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

or, (assuming that $w_2 \neq 0$),

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

For a positive response from the output unit, the net input it receives, namely, $b + x_1 w_1 + x_2 w_2$, be greater than 0. During training, values of w_1, w_2 and b are determined so that the net will have the correct response for the training data.
In terms of a threshold, a positive response from the output unit, the net input it receives, namely, $x_1 w_1 + x_2 w_2$, be greater than the threshold θ. This gives the equation of the line separating positive from negative output as

$$x_1 w_1 + x_2 w_2 = \theta,$$

or, (assuming that $w_2 \neq 0$),

$$x_2 = \frac{\theta}{w_2} + \frac{-w_1}{w_2} x_1.$$

The form of the separating line found by using above two concepts, there is no advantage to including both a bias and a nonzero threshold for a neuron that uses the step function as its activation function. On the other hand, including neither a bias nor a threshold is equivalent to requiring the separating line (or plane or hyperplane for inputs with more components) to pass through the origin.

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 made 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 1 if net input is positive and a -1 if the net input is negative. The net input to the output neuron is, $y_{in} = b + \sum_i w_i x_i$. The relation, $b + \sum_i w_i x_i = 0$ gives the boundary region of the net input. 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 one side of the boundary, then the problem is linear separable else it is linearly non-separable.
Say with two input vectors, the equation of line separating the positive region and negative region is given by, $b + x_1 w_1 + x_2 w_2 = 0$

$$x_2 = \frac{-b}{w_2} + \frac{-w_1}{w_2} x_1, \; w_2 \neq 0$$

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

Example 3.1.

Suppose we have two Boolean inputs $x_1, x_2 \in \{0, 1\}$, one Boolean output $t \in \{0,\ 1\}$ and the training set is given by the following input/output pairs

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

Then the learning problem is to find weight w_1 and w_2 and threshold (or bias) value θ such that the computed output of our network (which is given by the binary step function) is equal to the desired output for all examples.

A straightforward solution is $w_1 = w_2 = 1/2$, $\theta = 0.6$. Really, from the equation

$$x_1 \wedge x_2 = \begin{cases} 1 & \text{if } x_1/2 + x_2/2 \geq 0.6 \\ 0 & \text{otherwise} \end{cases}$$

it follows that the output neuron fires if and only if both inputs are on.

Example 3.2.

Suppose we have two Boolean inputs $x_1, x_2 \in \{0, 1\}$, one Boolean output $t \in \{0,\ 1\}$ and the training set is given for logical OR function. Then the learning problem is to find weight w_1 and w_2 and threshold (or bias) value θ such that the computed output of our network (which is given by the binary step function) is equal to the desired output for all examples.

A straightforward solution is $w_1 = w_2 = 1$, $\theta = 0.8$. Really, from the equation

$$x_1 \vee x_2 = \begin{cases} 1 & \text{if } x_1/2 + x_2/2 \geq 0.8 \\ 0 & \text{otherwise} \end{cases}$$

it follows that the output neuron fires if and only if at least one of the inputs is on. The removal of the threshold from our network is very easy by increasing the dimension of input patterns. Really, the identity

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n > \theta \Longleftrightarrow w_1x_1 + w_2x_2 + \ldots + w_nx_n - 1 \times \theta > 0$$

means that by adding an extra neuron to the input layer with fixed input value -1 and weight θ the value of the threshold becomes zero. It is why in the following we suppose that the thresholds are always equal to zero.

The McCulloch-Pitts neuron is perhaps the earliest artificial neuron was discovered in 1943. The requirements for McCulloch-Pitts neurons may be summarized as follows:

1. The activation of the McCulloch-Pitts neuron is binary as
$$y = f(y_{in}) = \begin{cases} 1 & if \quad y_{in} \geq \theta, \\ 0 & if \quad y_{in} < \theta. \end{cases}$$
2. Neurons is a McCulloch-Pitts network are connected by directed, weighted paths.
3. If the weight on a path is positive the path is excitatory, otherwise it is inhibitory.
4. All excitatory connections into a particular neuron have the same weight, although different weighted connections can be input to different neurons.
5. Each neuron has a fixed threshold. If the net input into the neuron is greater than the threshold, the neuron fires.
6. The threshold is set such that any non-zero inhibitory input will prevent the neuron from firing.
7. It takes one time step for a signal to pass over one connection.

Architecture

The architecture of the McCulloch-Pitts neuron is shown in the Fig. 11. 'y' is the McCulloch-Pitts neuron, it can receive signal from any number of other neurons. The connections weights from x_1, x_2, . . ., x_n are excitatory, denoted by 'w' and the connections weights from x_{n+1}, x_{n+2}, . . ., x_{n+m} are inhibitory denoted by '-p'.

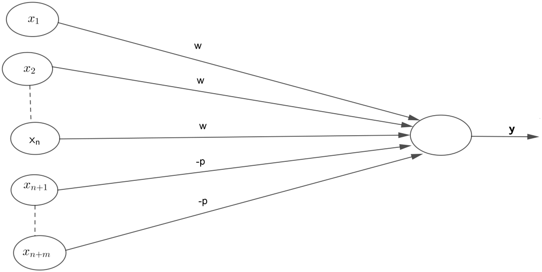


Fig. 11: Architecture of the McCulloch-Pitts neuron.

Architecture

The McCulloch-Pitts neuron, y, has the activation function

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

where θ is the threshold and y_{in} is the net input signal received by neuron y.
The threshold should satisfy the relation $\theta > nw - p$ if at least one inhibition connection present. 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 $kw \geq \theta > (k-1)w$.
Limitations:

- Weights and thresholds are analytically determined, cannot learn.
- Very difficult to minimize size of a network.
- What about non-discrete and/or non-binary tasks?

Example 3.3.

Generate the output of logical AND function by McCulloch-Pitts neuron model.

Solution: The logical AND function returns a true value only if both the inputs true, else it returns a false value. '1' represents true value and '0' represents false value. The truth table for AND function is

McCulloch-Pitts neuron example

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

A McCulloch-Pitts neuron to implement logical AND function is shown in the Fig. 12. The threshold on output unit 'y' is 2.

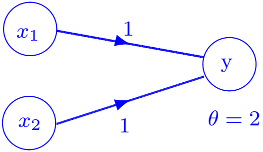


Fig. 12: MP neuron for AND gate

The net input is given by

$$y_{in} = \sum_i weights * input$$
$$y_{in} = 1 * x_1 + 1 * x_2$$
$$y_{in} = x_1 + x_2$$

The output is, $y = f(y_{in})$, where

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

Now present the inputs:

1. $x_1 = x_2 = 0$, $y_{in} = x_1 + x_2 = 0$, therefore, $y = f(y_{in}) = 0$ since $y_{in} = 0 < 2$.
2. $x_1 = 0$, $x_2 = 1$, $y_{in} = x_1 + x_2 = 1$, therefore, $y = f(y_{in}) = 0$ since $y_{in} = 1 < 2$. This is same for the input $x_1 = 1$, $x_2 = 0$.
3. $x_1 = x_2 = 1$, $y_{in} = x_1 + x_2 = 2$, therefore, $y = f(y_{in}) = 1$ since $y_{in} = 2$.

McCulloch-Pitts neuron example

Example 3.4.

- Generate the output of logical OR function by McCulloch-Pitts neuron model.
- Realize the NOT function using McCulloch-Pitts neuron model.
- Generate the output of ANDNOT function by McCulloch-Pitts neuron model.
- Realize the Exclusive-OR function using McCulloch-Pitts neuron model.
- Realize the following functions using McCulloch-Pitts neuron model:
  - NOR gate
  - NAND gate
  - $f(x_1, x_2, x_3) = x_1\, x_2'\, x_3 + x_1'\, x_2'\, x_3 + x_1\, x_2\, x_3'$

Hebb Net

The first learning law for artificial neural networks 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. For Hebb net, the input and 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.

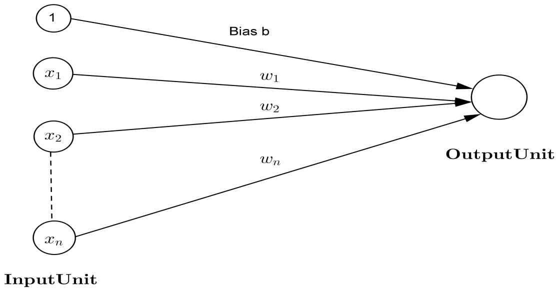


Fig. 13: Architecture of a Hebb Net.

Architecture

The architecture of Hebb net is shown in Fig. 13. The shown Hebb net is a single layer net, which consists of an input layer with any 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 always 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:

Hebb learning Algorithm

1. **Step 1:** Initially all weights and bias to zero

$$w_i = 0,\ b = 0 \text{ for } i = 1 \text{ to } n,$$

   where n is the number of input neurons. Set $\eta = 1$.

2. **Step 2:** For each input training vector and target output pair (S, t) perform Step 3-6.
3. **Step 3:** Set activations for input units with input vector

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

4. **Step 4:** Set activation for output unit with the output neuron $y = t$.
5. **Step 5:** Adjust the weights by applying Hebb rule,

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

6. **Step 6:** Adjust the bias

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

This algorithm requires only one pass through the training set.

Hebb Net examples

Example 3.5.

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

Solution: The training patters AND function are

Input			Target
x_1	x_2	x_0	y
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1

The weight change is calculated using

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

Input			Target	Weights Changes			Weights		
(x_1	x_2	x_0)	y	$\triangle w_1$	$\triangle w_2$	$\triangle b$	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

Hebb Net examples

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

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

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

Similarly, after 2nd, 3rd and 4th epochs, the separating lines are,

$$x_2 = 0,\ x_2 = 1 - x_1,\ x_2 = 1 - x_1.$$

For the 3rd and 4th epoch the separating lines remains the same, hence this line separates the boundary regions as shown in Fig. 14.

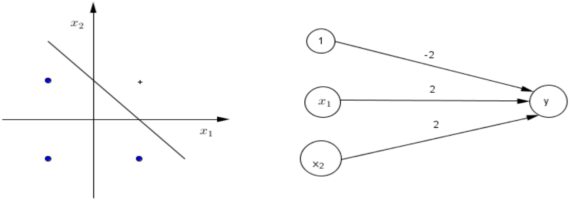


Fig. 14: Hebb Net for AND function.

Hebb Net examples

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

Example 3.6.

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

Solution: The training patters for XOR function

Input			Target
x_1	x_2	x_0	y
1	1	1	-1
1	-1	1	1
-1	1	1	1
-1	-1	1	1

Input			Target	Weights Changes			Weights		
$(x_1$	x_2	$x_0)$	y	$\triangle w_1$	$\triangle w_2$	$\triangle b$	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	0	0	0

Hebb Net examples

The final weights are obtained for XOR function are $w_1 = w_2 = b = 0$. 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.

Example 3.7.

(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.

Input					Target	Weights Changes					Weights				
(x_1	x_2	x_3	x_4	x_0)	y	$\triangle w_1$	$\triangle w_2$	$\triangle w_3$	$\triangle w_4$	$\triangle b$	w_1	w_2	w_3	w_4	b
										Initial	(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	2	0	2	0
-1	1	-1	-1	1	1	-1	1	-1	-1	1	-1	3	-1	1	1
1	-1	-1	1	1	-1	-1	1	1	-1	-1	-2	4	0	0	0

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

Hebb Net examples

Example 3.8.

Limitations of Hebb rule training for binary patterns.

Consider the following input and target output pairs:

Input			Target
x_1	x_2	x_3	y
1	1	1	1
1	1	0	0
1	0	1	0
0	1	1	0

This example shows that the Hebb rule may fail, even if the problem is linearly separable (and even if 0 is not the target). The plane $x_1 + x_2 + x_3 + (-2.5) = 0$, i.e., a weight vector of $(1, 1, 1)$ and a bias of -2.5 is separates the input patterns. It is easy to see that the updated weights do not produce the correct output for the first pattern.

Hebb Net examples

Example 3.9.

Limitation of Hebb rule training for bipolar patterns.

Consider the following input and target output pairs:

Input			Target
x_1	x_2	x_3	y
1	1	1	1
1	1	-1	-1
1	-1	1	-1
-1	1	1	-1

This example shows that the Hebb rule may fail, even with the input patterns (and target classifications) in bipolar form. The plane $x_1 + x_2 + x_3 + (-2) = 0$, i.e., a weight vector of (1, 1, 1) and a bias of -2 is separates the input patterns. It is easy to see that the updated weights do not produce the correct output for the first pattern.

Perception

Frank Rosenblatt [1962] and Minsky and Papert [1988], developed large classes of artificial neural networks called Perceptron. The perceptron learning rule uses an iterative weight adjustment that is more powerful than the Hebb rule. The original perceptron is found to have three layers, sensory, associator and response units as shown in Fig. 15.

Fig. 15: Original perception Network.

The sensory and association units have binary activation and an activation of $+1$, 0 and -1 is used for the response unit. All the units have their corresponding weighted interconnection. Training in perceptron will continue until no error occurs. This net solves the problem and is also used to learn the classification. The perceptrons are of two types: single layer and multi layer perceptrons.

Single layer perception

A single layer perceptron is the simplest form of a neural network used for the classification of patterns that are linearly separable. Fundamentally, it consists of a single neuron with adjustable weights and bias. The linearity and integrity learning makes the perceptron network very simple. Training in the perceptron continues till no error occurs.

Architecture:

The input to the response unit will be the output from the associate unit, which is binary vector. Since only the weight between the associate and the response unit is adjusted, the concept is limited to single layer network.

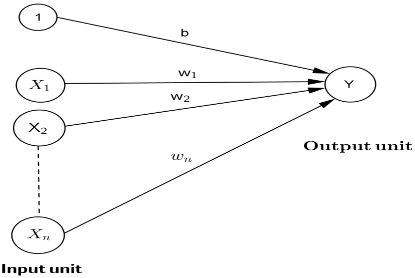


Fig. 16: Single layer perception Network.

Architecture

In the architecture shown in Fig. 16, only the associate unit and the response unit is shown. The sensor unit is hidden, because only the weights between the associate 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 neuron are connected to the output neurons through weighted interconnections. This is a single layer network because it has only one layer of interconnection between the input and the output neurons. This network perceives the input signal received and performs the classification.

Algorithm:

To start the training process, initially the weights and the bias are set to zero. It is also essential to set the learning rate parameter, which ranges between o to 1. The output is compared with the target, where if any difference occurs, we go in for weight updating 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 follows:

Perceptron Algorithm

1. **Step 1** Initialize weights and bias (for simplicity, set weights and bias to zero). Set learning rate η, $(0 < \eta \leq 1)$ (for simplicity, η can be set to 1).
2. **Step 2** While stopping condition is false, do Steps 3-7.
3. **Step 3** For each training pair $S : t$, do Steps 4-6.
4. **Step 4** Set activations of input units: $x_i = s_i$ for $i = 1$ to n.
5. **Step 5** Compute response of output unit:
$$y_{in} = b + \sum_i x_i w_i;$$
$$y = f(y_{in}) = \begin{cases} 1 \quad if \quad y_{in} > \theta, \\ 0 \quad if \quad -\theta \leq y_{in} \leq \theta, \\ -1 \quad if \quad y_{in} < -\theta \end{cases}$$
6. **Step 6** Update weights and bias for $i = 1$ to n.
If $y \neq t$ and $x_i \neq 0$,
$$w_i(new) = w_i(old) + \eta t x_i,$$
$$b(new) = b(old) + \eta t.$$
Else
$$w_i(new) = w_i(old),$$
$$b(new) = b(old).$$
7. **Step 7** Test stopping condition: If no weights changed in Step 3, stop; else, continue.

Perceptron Algorithm

Note that only weights connecting active input units ($x_i \neq 0$) are updated. Also, weights are updated only for patterns that do not produce the correct value of y. This means that as more training patterns produce the correct response, less learning occurs. This is in contrast to the training of the Adaline.

Perceptron Algorithm for Several Output Classes:
The perceptron algorithm for single output class is extended for several output classes. Here, there exits 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:

Perceptron Algorithm for Several Output Classes

1. **Step 1** Initialize weights and bias. Set learning rate η, $(0 < \eta \leq 1)$.
2. **Step 2** While stopping condition is false, do Steps 3-7.
3. **Step 3** For each training pair $S : t$, do Steps 4-6.
4. **Step 4** Set activations of input units: $x_i = s_i$ for $i = 1$ to n.
5. **Step 5** Compute response of output unit:

$$y_{j,in} = b_j + \sum_i x_i w_{ij}; \text{ for } j = 1 \text{ to } m$$

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

6. **Step 6** Update weights and bias for $i = 1$ to n and $j = 1$ to m.
   If $y_j \neq t_j$ and $x_i \neq 0$,

$$w_{ij}(new) = w_{ij}(old) + \eta t_j x_i,$$
$$b_j(new) = b_j(old) + \eta t_j.$$

   Else

$$w_{ij}(new) = w_{ij}(old),$$
$$b_j(new) = b_j(old).$$

7. **Step 7** Test stopping condition: If no weights changed in Step 3, stop; else, continue.

Example

Example 3.10.

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

Solution: The training patters for AND function can be,

Input			Target
x_1	x_2	x_0	t
1	1	1	1
-1	1	1	-1
1	-1	1	-1
-1	-1	1	-1

Forming the table, initialized all the weights and the bias to be zero i.e. $w_1 = w_2 = 0$ and $b = 0$ and take $\eta = 1$, $\theta = 0$.
Calculate the net input as $y_{in} = x_0 b + x_1 w_1 + x_2 w_2$ and then apply the activation function given as

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

Update weights and bias for each activation input x_i, if $y \neq t$, then

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

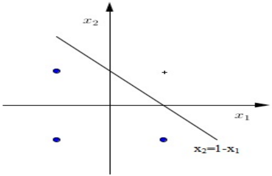
$$b(new) = b(old) + \eta t.$$

All the input vectors, updated weights and bias are presented in the following table:

Example

Input			Net	Output	Target	Weights Changes			Weights		
(x_1	x_2	x_0)	y_{in}	y	t	$\triangle w_1$	$\triangle w_2$	$\triangle b$	w_1	w_2	b
								Initial	(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 training. The final weights after the first epoch is completed are, $w_1 = 1, w_2 = 1$ and $b = -1$. The straight line separating the regions can be obtained after first epoch is $x_2 = 1 - x_1$. The decision boundary is given in Fig. 17.



Example

In similar way, the perceptron network can be developed for logic functions OR, NOT, ANDNOT, etc.

Example 3.11.

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

(a). With (0, 0) and without bias.
Epoch-1:

Input		Net	Output	Target	Weights Changes		Weights	
$(x_1$	$x_2)$	y_{in}	y	t	$\triangle w_1$	$\triangle w_2$	w_1	w_2
						Initial	(0	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 lines for 1st and 2nd inputs are $x_1 + x_2 = 0$ and $x_2 = 0$ respectively. Which are not decision boundary.

Example

(b). Without (0, 0) and bias.
Epoch-1:

Input		Net	Output	Target	Weights Changes		Weights	
(x_1	x_2)	y_{in}	y	t	$\triangle w_1$	$\triangle w_2$	w_1	w_2
						Initial	(0	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

The separating lines here also, without bias are $x_1 + x_2 = 0$ and $x_2 = 0$ respectively. 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 3.12.

Using the perceptron learning rule, find the weights required to perform the following classifications. Vectors (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 members of class (having target value -1). Use learning rate of 1 and starting weights of 0. Using each of the training and vectors as input, test the response of the net.

Example

Input					Net	Output	Target			Weights Changes					Weights		
(x_1	x_2	x_3	x_4	x_0)	y_{in}	y	t	$\triangle w_1$	$\triangle w_2$	$\triangle w_3$	$\triangle w_4$	$\triangle b$	w_1	w_2	w_3	w_4	b
Epoch-1																	
1	1	1	1	1	0	0	1	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	0
-1	1	-1	-1	1	-2	-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	-2	2	0	0	0
Epoch-2																	
1	1	1	1	1	0	0	1	1	1	1	1	1	-1	3	1	1	1
1	1	1	-1	1	3	1	-1	-1	-1	-1	1	-1	-2	2	0	2	0
-1	1	-1	-1	1	2	1	1	0	0	0	0	0	-2	2	0	2	0
1	-1	-1	1	1	-2	-1	-1	0	0	0	0	0	-2	2	0	2	0
Epoch-3																	
1	1	1	1	1	2	1	1	0	0	0	0	0	-2	2	0	2	0
1	1	1	-1	1	-2	-1	-1	0	0	0	0	0	-2	2	0	2	0
-1	1	-1	-1	1	2	1	1	0	0	0	0	0	-2	2	0	2	0
1	-1	-1	1	1	-2	-1	-1	0	0	0	0	0	-2	2	0	2	0

From the last epoch, it seen that final weights are $w_1 = -2$, $w_2 = 2$, $w_3 = 0$, $w_4 = 2$ and $b = 0$.

Example

Example 3.13.

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

$$(0,\ -1,\ 1),\ (0,\ 1,\ -1),\ (0,\ 0,\ 1),\ (0,\ 0,\ -1),\ (0,\ 1,\ 0),\ (1,\ 0,\ 1)$$

$$(1,\ 0,\ -1),\ (1,\ -1,\ 0),\ (1,\ 0,\ 0),\ (1,\ 1,\ 0),\ (0,\ -1,\ 0),\ (1,\ 1,\ 1)$$

Solution: The concept used for this problem is
If $x_1w_1 + x_2w_2 + x_3w_3 > 0$, then the response is correct.
If $x_1w_1 + x_2w_2 + x_3w_3 < 0$, then the response is incorrect.
If $x_1w_1 + x_2w_2 + x_3w_3 = 0$, then the response is indefinite or undetermined.
The weights take for bipolar activation function are, $w_1 = 0$, $w_2 = -2$ and $w_3 = 2$

Multi-layer perception

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 and 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 algorithm. The disadvantages of the single layer perceptron is that it cannot be extended to the multilayer version. In MLP networks there exists a non-linear activation function. The widely used non-linear activation function is logistic sigmoid function. The MLP network also has various layers of hidden neurons. The hidden neurons make the MLP network active for highly complex tasks. The layer of the network are connected by synaptic weights. The MLP thus has a high commotional efficiency.
A disadvantage of MLP may also be the presence of non-linearity and complex connection of the network which leads to highly complex theoritical analysis. Also the existence of hidden neurons makes the learning process tedious.
THe MLP networks are usually fully connected networks. There are various multilayer perceptron networks which includes Back Propagation, Radial basis function, etc.