Q. Out line the salient properties of Neural Network?

Ans There are 3 main properties of Neural N/W -

1. Robustness →

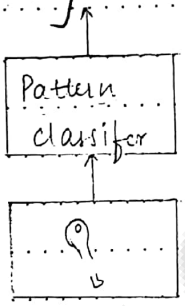
Neural N/W continue to operate, even if some connection in the N/W get damaged.

will work even if

i.e Even if there is a damage in its internal structure, Neural N/W tends to degrade smoothly, which is called as graceful degradation.

N.N can also tolerate distortion in the inputs for which they are programmed. This property is called fault tolerance.

graceful degradation – degrades smoothly
fault tolerance
← Tolerate distortion



Partially occluded image of key

According to the figure -

Though the i/p is an occuled image then also it can be successful classified by a neural N/W.

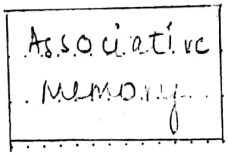
2. Associative Recall →

It is a natural capability of neural N/W because there is dense inter-connectivity b/w neurons.

In Associative recall, one concept involes related memories.

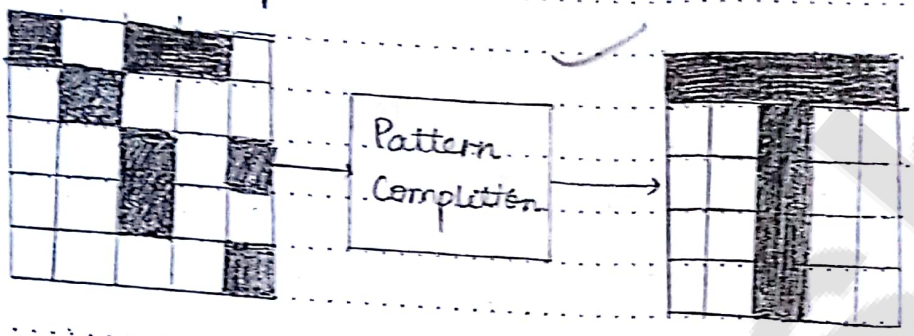
i.e when we see our friends photograph, immidiatly our brain start recognising his/her by her name, related emotions and contact no, mail ID etc.

Photograph of friend



Name
phone no
E-mail

Neural N/W also has Pattern completion property. Here, if i/p is incomplete pattern then N/W gives O/p as complete pattern.

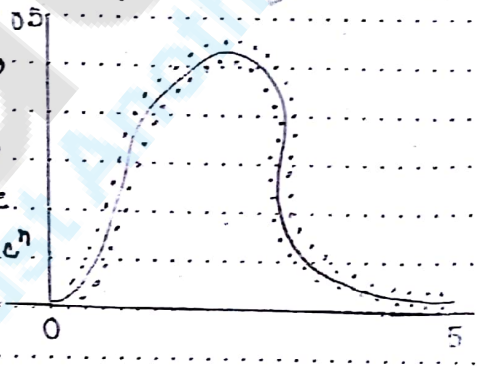


Neural N/W can clean up noisy images or implement associative memory.

3. Func^n Approximation and Generalization -

Neural N/Ws are able to create internal representation only through examples not by any mathematical model. This is called similarity based generalization.

Using powerful learning algos, they are able to approximate fun^n. Therefore, due to their learning capability, they are also called as adaptive func^n estimators.



Neural N/W can approximate a func^n from a noisy and sparse data scatter.

- Properties of NN
  - Robustness
    - Graceful Degradation
    - Fault Tolerance
  - Associative Recall
    - one concept involves related Memory
    - Patter completion
  - func^n Approximation & Generalization
    - Similarity based generalization
    - Adaptive func^n estim

UL 9 6

§ Fundamental Concept →

Artificial Neural Network → (ANN)

ANN is an efficient Info^n processing sys which resembles in characteristics with a biological neural network.

ANN has large number of highly interconnected processing elements called as nodes / units / neurons that operated parallely.

Each neuron is connected with the other by a connection link.

Each connection link is associated with weights which contains information about the input signal.

This information is cused by the neuron net to solve a particular problem.

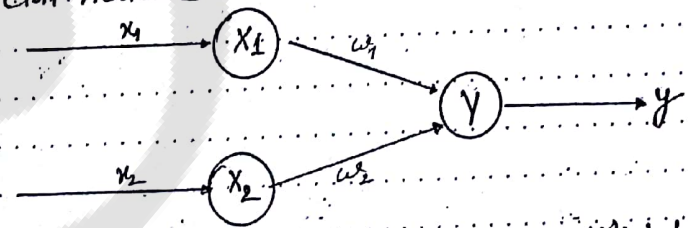
ANN's collective behavior projects —

- ability to learn
- recall
- classify the data

similar to that of a human brain.

They have the capability to model networks of original neurons as found in the brain.

Thus, the ANN processing elements are called neurons or artificial neurons.



Architecture of simple artificial neuron net

Each neuron has an internal state of its own. This internal state is called the activation, which is the function of the inputs the neuron receives:

The activation signal of a neuron is transmitted to other neurons.
A neuron can send only one signal at a time which can be transmitted to several other neurons.

Operation of neural net →

X_1 and X_2 are neurons, transmitting signals to another neuron, Y.

here X_1 & X_2 are input neurons and

Y is output neuron.

X_1 & X_2 are connected to output neuron Y via weighted interconnection link (W_1 & W_2).

So, the net input has to be calculated in the following way —

$$y_{in} = x_1 w_1 + x_2 w_2 \quad \text{—(1)}$$

where x_1, x_2 are activation of i/p neuron
(i.e. the output of input signals)

The output y of the output neuron can be obtained by applying activations over the net input.

∴ The function of the net input:

(2) — $y = f(y_{in})$ → This function is called activation function.

Output = function (net input calculated)

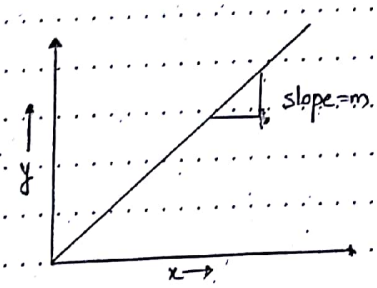
The calculation of the net input from Eq. (2) is similar to the calculation of output of a pure linear straight line equation ($y = mx$)

Input → x → [m] → y → mx

Here, to obtain the output y, the slope m is directly multiplied with the input signal.

This is a line equation.

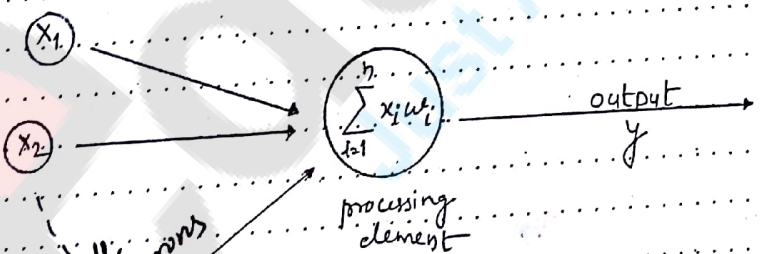
Thus, when slope and input are linearly varied, the output is also linearly varied.



Graph for $y = mx$

This shows weight involved in ANN is equivalent to the slope of the straigh line.

Mathematical Model of Artificial neuron →



The weight represents the strength of synapse connecting the i/p & o/p neurons

+ve weight corresponds to an excitatory synapse उत्तेजक

-ve weight corresponds to an inhibitory synapse अवरोधी

So, net input

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

where i represents ith processing element.

The activation function is applied over it to calculate the output.

The weigh

Assignment (1) Write note on –

a) Comparison b/w Biological Neuron & Artificial Neuron in 6 points refer pg 14-15 from Sivanandam & Deepa

§ Basic Model of Artificial Neural Network →

Model of ANN can be specified on the basis of following 3 entities →

1. The model's synaptic interconnections [Network Layers Architecture]
2. The training or learning rules adopted for updating & adjusting the connection weights.
3. The Activation functions.

Assignment (1)

b) Explain different network architecture

There are total 5 network's

refer Sivanandam & Deepa page 17–20
+
Rajasekaran & Vijayalakshmi page 16 – 18
+
Satish Kumar

Lea

ANN is a self adaptive approach, it means NN can "learn by example". (12) (13)

3) Learning →

Q. What is learning in NN? Differentiate b/w Supervised & Unsupervised Learning. (10)

Q. What is learning? Compare different Learning rules (10)

Q. Explain with neat diagram Supervised & Unsupervised learning. (6)

Q. Write 3 differences b/w supervised & Unsupervised learning. (5)

The main property of an ANN is its capability to learn.

Learing or training is the process by which neural network adapts changes to improve its performance.

Soln: can say ANN learn from its environment and improve its performance.

There are 2 types learning in ANNs →

1) Parameter learning → It updates weights in neural net

2) Structure learning → It changes network structure
it includes →
a) number of processing elements
b) connection types

Above 2 types of learning can be performed simultaneously or separately apart from these, there are 3 kinds of learning.

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement learning

1. Supervised learning →

It is learning with a teacher that is why it is called as supervised learning.

for Example -

The learning process of a small child starts at home and then by the teacher in school.

The child doesn't know how to read / write. The children are trained and molded to recognize the alphabets, numerals etc.
Their each and every action is supervised by a teacher.

Actually, a child works on the basis of the output that he/she has to produce.

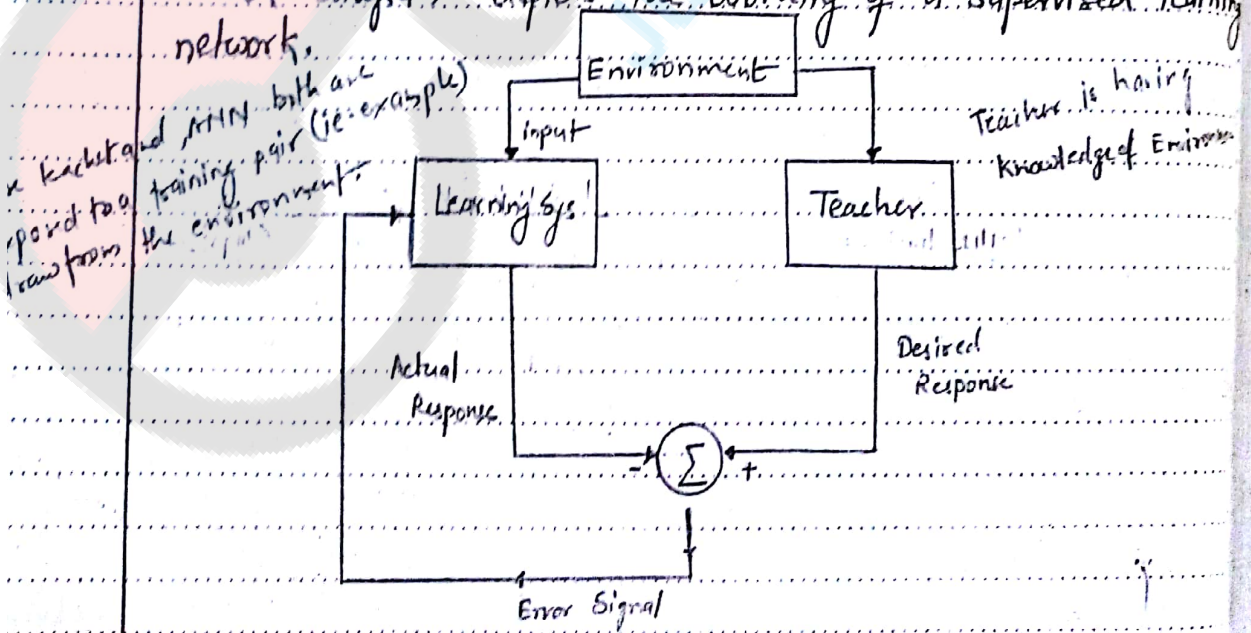
All these real time events involve supervised learning methodology.

Similarly, ANN follows supervised learning →

Each input vector requires a corresponding target vector, which represents the desired output.

The input vector along with the target vector is called training pair. [i.e. example] The neural network is informed beforehand about what should be produced as output.

Block diagram depicts the working of a supervised learning network.



Block diagram of supervised learning.

By virtue of built-in knowledge, the teacher is able to provide the desired response for that training vector.

Then the actual response is compared with the desired response (Output vector). If there exists a difference between the two output vectors then an error signal is generated by the network.

This error signal is used for adjustment of weights until the actual response (output) matches the desired response.

In this type of training, a supervisor or teacher is required for error minimigation.

In this way knowledge of the environment available to teacher is transferred to the neural network.

2) Un Supervised Learning →

Here, learning is performed without the help of a teacher

Consider the learning process of a tadpole, it learns by itself, that is, a child fish learns to swim by itself, it is not taught by its mother.
This its learning process is independent and is not supervised by a teacher. [Most of these algos perform some kind of clustering operation. They learn to categorize the i/p pattern into a finite no of class]
In ANN following unsupervised learning, the input vectors of similar type are grouped without the use of training data.

In the training process, the network receives the input patterns and organizes these patterns to form clusters.

When a new i/p pattern is applied, the neural network gives an output response that indicates the class to which the input pattern belongs.

If for an input a patter class cannot be found then a new class is generated.

The block diagram clearly shows that there is no feedback from the environment to inform what the o/p should be or whether the outputs are correct.

In this case the network must itself discover patterns from the input data. during this process the network undergoes changes in its parameters. This process is called self-organizing in which exact clusters will be formed by discovering similarities and dissimilarities among the objects.

Here the desired response is not known. thus explicit error info cannot be used to improve N/w Behavior.

Learning must somehow be accomplished based on observations of responses

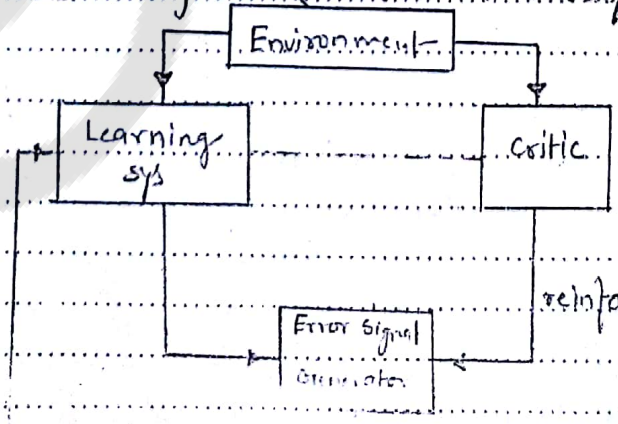
Block diagram of unsupervised learning

3 Reinforcement Learning →

This learning process is similar to supervised learning. In the case of supervised learing the correct target output values are known for each input pattern. But in some cases less information might be available.

For Ex- the network might be told that its actual output is only "50% correct" or so. Thus here only critic information is available, not the exact information.

The learning based on critic information is called reinforcement learning and the feedback sent is called reinforcement signal.



The reInforcement Learning is a form of supervised learning because the network receives some feedback from its environment.

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

It is also called learing with critic.

§ Activation Functions →

Q. Explain different Activation functions (8)

To understand activation funcⁿ, let us assume a person is performing some work. To make the work more efficient and to obtain exact output, some force or activation may be given.

This activation helps in achieving the exact output.

In a similar way, the activation function is applied over the net input to calculate the output of an ANN.

The Information processing of a processing element has 2 major parts -

1. input
2. output

An integration function f is associated with the input of a processing element.

There are several Activation functions out of those following 5 are mostly used →

1- Identity function
2- Binary step function
3- Bipolar step function
4- Sigmoidal function
5- Ramp function

1- Identity Function →

It is a linear function and can be defined as

output → $y = f(y_{in}) = y_{in}$ ← input

$\forall\ y_{in}$

Here, output is same as input.

2- Binary Step function [Threshold function]

To generate the final output y, the total input y_{in} received by the soma of an ANN is passed on to a non-linear filter called as activation funct. f

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

A very commonly used Activation function is the Thresholding funct.

In this, the sum is compared with a threshold value θ.

If the value of y_{in} is greater than θ, then the output is 1 else it is 0.

$y = f(y_{in} - \theta)$

$y_{in} = w_1 x_1 + w_2 x_2 + \cdots w_n x_n$

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

$$y = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

where f is the step funct or Heaviside funct

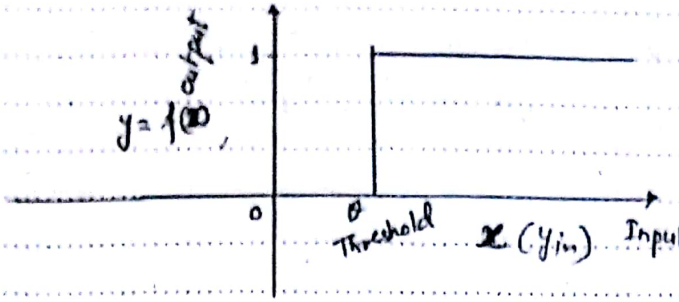
can be defined as

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

where θ = threshold value

This funct is mostly used in single-layer nets to convert the net input to an output that is a binary 0 or 1.

Threshold Activation function can be described by



It shows that the output signal is either 1 or 0
i.e. neuron is either On or Off.

3. Bipolar Step function →

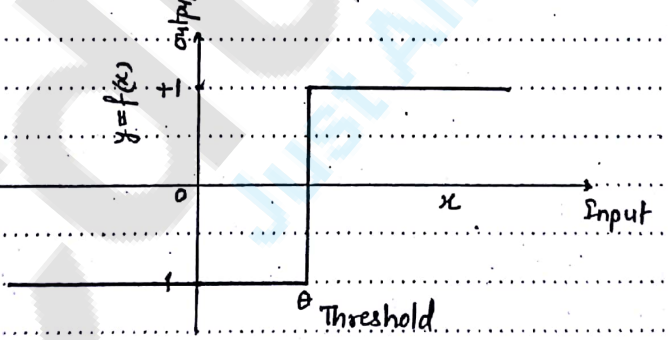
This function can be defined as

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

where θ = threshold value.

It is also known as signum function or quantizer function.

This funch can be described as →



4. Sigmoidal functions →

This is the most populer function used in ANN. Its graph is S-shaped. It is defined as strictly increasing function. And it is defined by →

$$f(x) = \frac{1}{1+e^{-\lambda x}}$$ where λ is the slope parameter

By varying the parameter λ, we obtain sigmoid functions of different slopes.

In the limit, as the slope parameter approaches infinity, the sigmoid function becomes simply a threshold function.

Note: The sigmoidal function is differentiable, where as the threshold function is not.

Differentiability is an important feature of neural network theory.

There are 2 types of sigmoidal funch-

a: Binary sigmoid function →

Here the range of the sigmoid function is from 0 to 1.

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

The derivative of this funch is

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

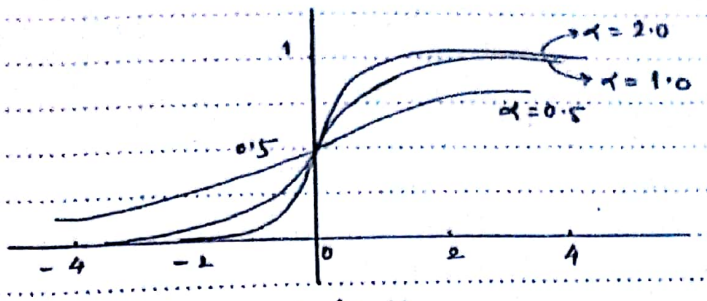
b: Bipolar Sigmoid function →

The sigmoid function range is between -1 & $+1$

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

The derivative of this funch can be-

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



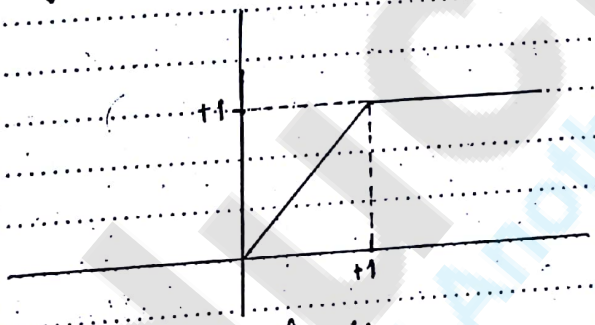
sigmoidal function

5. Ramp function →

It is defined as

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

The graphical representation is



Ramp function

§ Important Terminologies of ANN →

1. Weights →

In ANN, each neuron is connected with other neuron by means of directed communication links, and each communication link is associated with weights. It is also called as connection strength that can be either -ve or positive.

Specifically, a signal x_j at the input of synapse j

(02) (22)

connected to neuron k is multiplied by the synaptic weight w_{kj}.

first subscript refers to the neuron from where signal is coming.

second subscript refers to the input end of the synapse to which the weight refers.

The weight contains information about the input signal. This information is used by the net to solve the problem.

The weight can be represented in terms of matrix.

It is also called as connection matrix.

for mathematical notation -

Assuming that there are n processing elements in ANN and each processing element has exactly m adaptive weights.

$$W = \begin{bmatrix} w_1^T \\ w_2^T \\ \vdots \\ w_n^T \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1m} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2m} \\ w_{31} & w_{32} & w_{33} & \cdots & w_{3m} \\ \vdots & & & & \\ w_{n1} & w_{n2} & w_{n3} & \cdots & w_{nm} \end{bmatrix}$$

If the weight matrix W contains all the adaptive elements of an ANN, then the set of all W matrices will determine the set of all possible information processing configurations for this ANN.

The ANN can be realized by finding an appropriate matrix W.

Hence, the weight encode long-term memory (LTM) and The activation states of neuron encode short Term Memory (STM) in NN.

(12) (23)

Q. Bias →

Bias is included in network by adding a component $x_0 = 1$ to the input vector X.

so, the input vector becomes

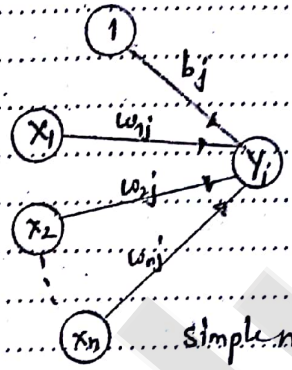
$$X = (1, x_1, x_2 \cdots x_n)$$

Bias has its impact in calculating the net input.

The bias is considered like another weight

i.e. $w_{0j} = b_j$

Consider a simple network →



simple net with bias

The net input to the output neuron y_j

$$= \sum_{i=0}^{n} x_i w_{ij}$$

$$= x_0 w_{0j} + x_1 w_{1j} + x_2 w_{2j} + \cdots x_n w_{nj}$$

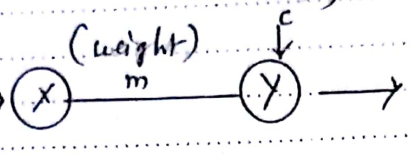
$$= x_0 w_{0j} + \sum_{j=1}^{n} x_i w_{ij}$$

$$= 1 \cdot w_{0j} + \sum_{j=1}^{n} x_i w_{ij}$$

$$= w_{0j} + \sum_{j=1}^{n} x_i w_{ij}$$

$$y_j = b_j + \sum_{i=1}^{n} x_i w_{ij}$$

Bias can also be explained in terms of straigh line Equation.



Thus, bias plays a major role in determining the output of the network.

The bias can be of 2 types -

* +ve bias - helps in increasing net i/p of network.
* -ve bias - helps in decreasing net i/p of network.

Thus, as a result of the bias effect, the output of the network, can be varied.

3. Threshold →

The threshold value is used in Activation funcⁿ.

A comparison is made between the calculated net input and the threshold to obtain the network output.

For each and every application, there is a threshold limit.

Consider a direct curren (Dc) motor. If its maximum speed is 1500 rpm then the threshold based on the speed is 1500 rpm.

If the motor is run on a speed higher than its set threshold, it may damage motor coils.

Similarly in Neural networks, based on the threshold value, the activation functions are defined and the output is calculated.

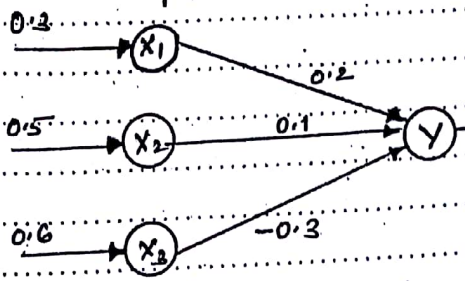
(U2) (25)

The activation function using Threshold can be defined as

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

where θ is the fixed Threshold value.

Q. For the network show in figure below, calculate the net input to the output neuron.



Solution: The given neural net consists of 3 input neurons & 1 output neuron.

The inputs are

$$[x_1, x_2, x_3] = [0.3, 0.5, 0.6]$$

and weights are

$$[w_1, w_2, w_3] = [0.2, 0.1, -0.3]$$

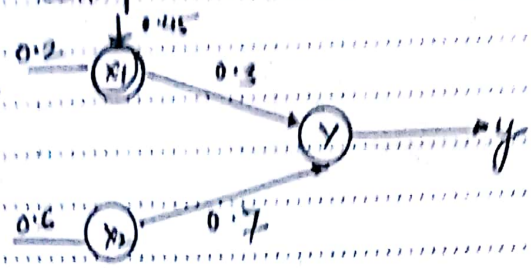
The net input can be calculated as

$$y = x_1 w_1 + x_2 w_2 + x_3 w_3$$
$$= 0.3 \times 0.2 + 0.5 \times 0.1 + 0.6 \times (-0.3)$$
$$= 0.06 + 0.05 + (-0.18)$$
$$= 0.11 - 0.18 = -0.07$$

∴ net output is -0.07

11

Q. Calculate the net input for the network shown in figure below with bias included in the network.



Soln → The given net consists of 2 input neurons
1 bias &
1 output neuron

The Inputs are

$[x_1, x_2] = [0.2, 0.6]$

weight are

$[w_1, w_2] = [0.3, 0.7]$

bias

$b = 0.45$

and we know that bias input $x_0 = 1$

∴ net input is

$y = b \cdot x_0 + x_1 w_1 + x_2 w_2$

$= 0.45 \times 1 + 0.2 \times 0.3 + 0.6 \times 0.7$

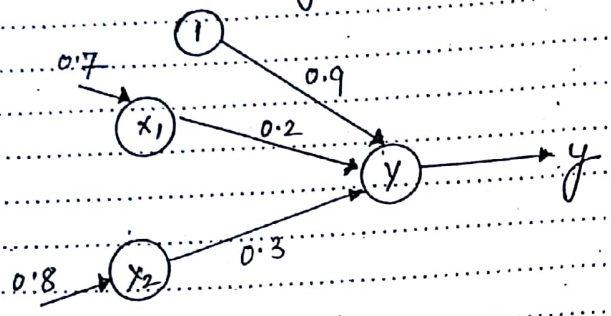
$= 0.45 + 0.06 + 0.42$

$= 0.93$

∴ The net input is 0.93

15

Q. Calculate the output of the neuron Y for the net given below. Use binary and bipolar sigmoidal funcn. (5)



Soln →

The given network has
- 2 input neurons
- 1 bias
- 1 output neuron

The given inputs are

$$[x_1, x_2] = [0.7, 0.8]$$

weights are

$$[w_1, w_2] = [0.2, 0.3]$$

with bias

$b = 0.9$ (Its input is always 1)

~~The net output of~~

The net input to the output neuron-

$$\text{net } y = b + \sum_{i=1}^{n} x_i w_i$$

$$= 0.9 + 0.7 \times 0.2 + 0.8 \times 0.3$$

$$= 0.9 + 0.14 + 0.24$$

$$\text{net } y = 1.28$$

i) For binary sigmoidal funcn →

$$y = f(\text{net}) = \frac{1}{1 + e^{-\lambda \text{net}}} = \frac{1}{1 + e^{1.28}} = 0.78$$

ii) for bipolar sigmoidal activation funcⁿ →

$$y = f(net) = \frac{2}{1 - e^{-\lambda \cdot net}} - 1$$

class assignment inside this sheet

$$= \frac{2}{1 - e^{-1 \cdot 28}} - 1 = 0.5649$$

(for basic neuron λ is always = 1)
It was the drawback in earlier neuron, so to get accurate value taken on they included λ in it)

Q. A neuron with 4 inputs has the weight vector $w = [1\ 2\ 3\ 4]$. The activation function is linear, that is, the activation funcⁿ is given by $f(net) = 2 * net$. If the input vector is $X = [5\ 6\ 7\ 8]^t$, then find the output of the neuron. (05)

given weigh vector is

$$w = [1\ 2\ 3\ 4]$$

Input vector

$$X = [5\ 6\ 7\ 8]$$

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

$$= 5*1 + 6*2 + 7*3 + 8*4$$

$$= 5 + 12 + 21 + 32$$

$$net = y_{in} = 70$$

Given linear activation funcⁿ is

$$f(net) = 2 * net$$

$$\therefore f(net) = 2 * 70$$

$$= 140$$

Ans = $f(net) = 140$

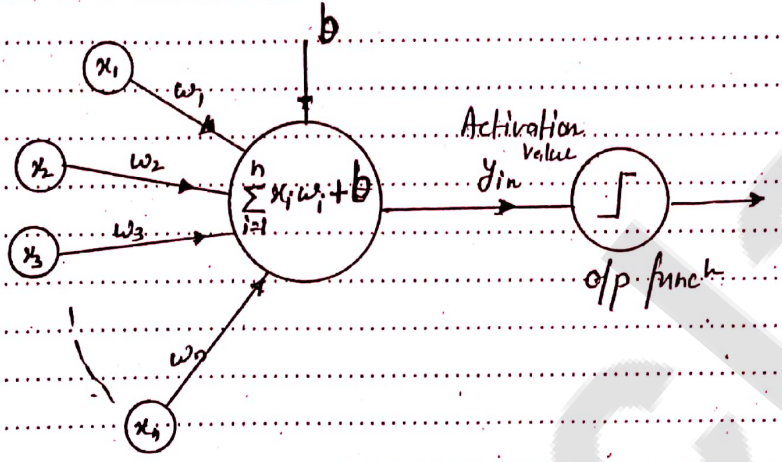
(U2) (28 a)

Different Models of Neuron →

There are 3 different models of neuron

1. Mc Cullock – Pitts Model
2. Perceptron
3. Adaline

1. Mc Cullock - Pitts Model →



According to the figure $x_1, x_2, \dots x_n$ are inputs and

$w_1, w_2, \dots w_n$ are weights

with bias term θ

And output signal y is obtained after applying activation funch on ~~input~~ Activation value y_{in}

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

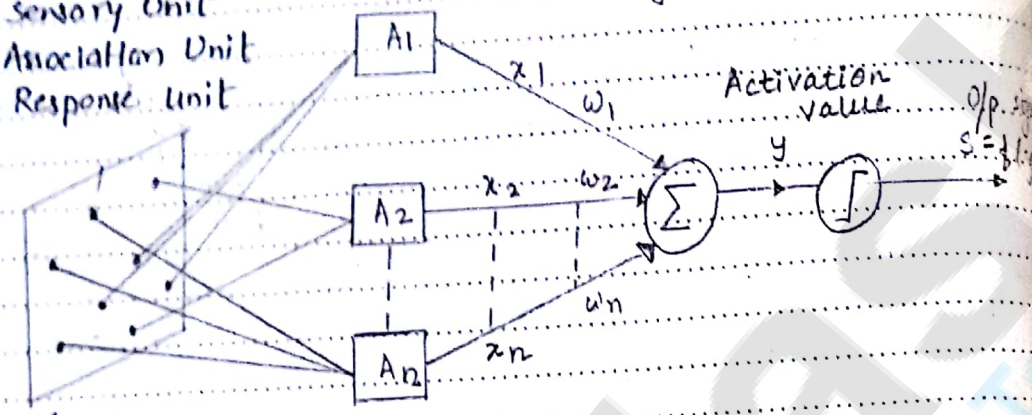
o/p signal $y = f(y_{in})$

In this model weights are fixed therefore the n/w which uses this model does not have the capability of learning.

Perceptron → The Perceptron is a computational model of the retina of the eye and hence, is named 'Perceptron'

The Perceptron Network consists of 3 units =
- Sensory Unit
- Association Unit
- Response Unit

Weights are adjustable



This model is given by Rosenblatt

According to the figure O/p of sensory unit are sent to Association units. And then O/ps of Association units are fed to the MP neuron [Mc-culloch Pitts]

The main difference b/w MP neurons and Perceptron is the adjustment of weights.

Here the desired o/p d is compared with the actual O/p s. and the error δ is calculated.

Activation $y = \sum_{i=1}^{n} x_i \omega_i + \theta$ → bias

O/p signal $S = f(y)$

//Error $\delta = d - s$

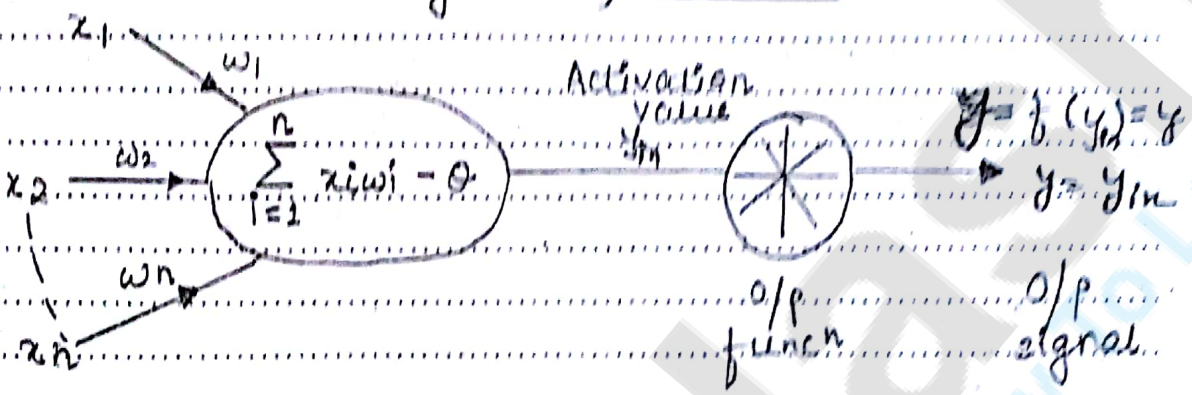
Weight change $\Delta \omega_i = \eta(\delta) x_i$

when η = learning

3. Adaline -

Its full form is ADAptive LINear Element [ADALINE]

This model is given by Widrow



The main difference b/w Perceptron and ADALINE is that, the o/p is a linear funch of the Activation value y.

$$\text{Activation } y = \sum_{i=1}^{n} x_i w_i = \theta$$

o/p signal $S = f(y) = y$

Error $\delta = d - s = d - y$

Weight change $\Delta w_i = \eta \delta x_i$ where η = learning rate

(02) (28d)

In short

MP neuron → 1) wt fix
2) O/p signal is non-linear func^n

1 factor is changed

Perception → 1) wts are not fixed
2) O/p signal is non-linear func^n

2 factor are changed

ADALINE → 1) wts are not fixed
2) O/p signal is linear func^n

Q. State the concept of linearly and non-linearly separable pattern classification (10)

OR

Q List the limitations of Perceptron (5)

Consider Iris data classification problem—

Here, features of flowers are measured. This measurement was given by Anderson and fisher.

It has 150 patterns each having 4 feature measurements

The Sepal Length
The Sepal Width
The Petal Length
The Petal Width

The data set is devided into 3 subsets of 50 patterns of each for 3 different Species of flowers.

1. Iris Sestosa : △
2. Iris Versicolor : +
3. Iris Verginica : o

The fig(a) shows that it is easy to separate the 3 datasets.

But some times it may happen that 2 datasets may overlapped and they can not be separated from each other as shown in fig (b)

fig(a)

Iris Data classification problem shows that finding a clearcut classification from a data sample is a complex process and drawing a decision boundary becomes a difficult problem to solve.

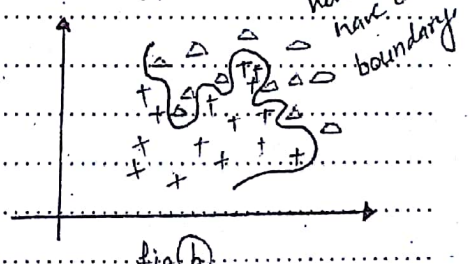
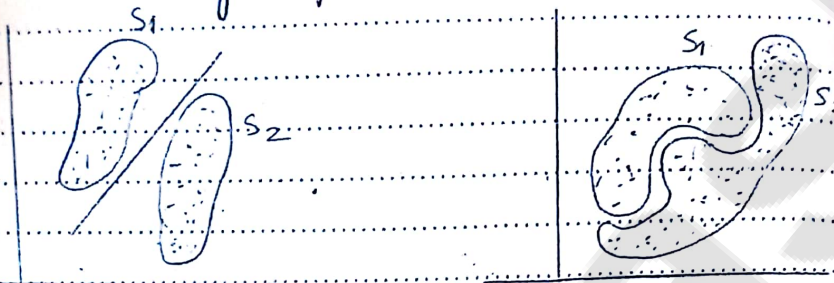


fig (b)

10 Limitation

Perceptron can not handle those tasks which are not linearly separable.

Or we can say a Perceptron func's properly when classes are linearly separable.



Linearly separable patterns

Non-linearly separable patterns

Limitation

For Example Perceptron is unable to solve XOR prob

Perceptron model with 2 inputs →

Consider a

Input calculated as

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

If we apply a bipolar step funct over net input y_{in}

then the value of func is 1 for +ve input

−1 for −ve input

so it shows there will be 2 region one for +ve and one for −ve values (i.e $y_{in} < 0$ & $y_{in} > 0$)

and at decision boundary

$$b + \sum_{i=1}^{n} x_i w_i = 0$$

∴ for above neural network the eqn will be

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

Before this just discuss about MP neuron, Perceptron + ADALINE from I-DTA notes.

6 Supervised Learning Network →

Perceptron Network →

The perceptron is a computational model of the retina of the eye and hence, is named 'perceptron'

The Perceptron Network consists of 3 units

- Sensory unit
- Association unit
- Response unit

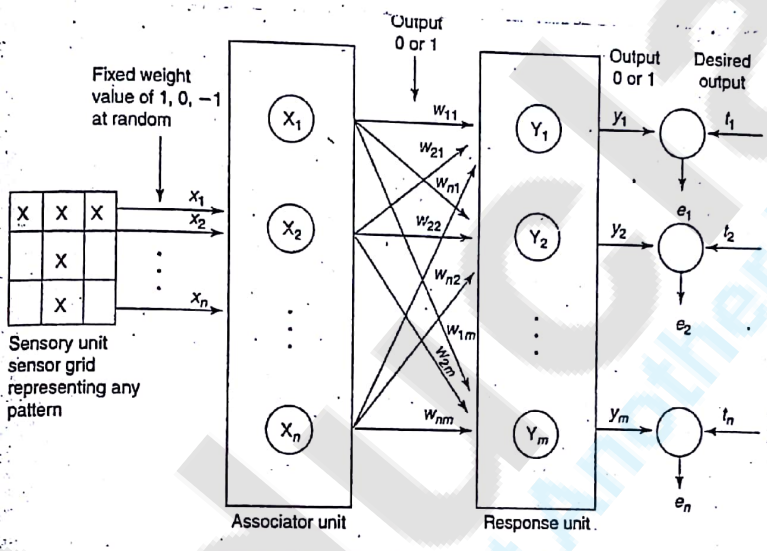


Figure 3-1 Original perceptron network.

The sensory units are connected to associator units with fixed weights 1, 0 or −1 which as assigned at random.

The binary activation function is used in sensory unit and associator unit.

The response unit has an activation of 1, 0 or −1, the binary step with fixed threshold θ is used as activation for ~~threshold~~ associator.

o/p signals that are sent from the associator unit to the response unit are only binary.

(U2) (30)

The output of the perceptron is given by

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

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

The perceptron learning rule is used in the weight updation b/w the associator unit and the response unit.

For each training input, the net will calculate the respon and it will determine whether or not an error ha occured.

The error is calculated on the basis of comparis b/w the values of target and values of the calculat outputs.

weights are adjusted for those connections which send the nonzero signal from associator unit to Response unit.

weights will be adjusted according to the learning rule.

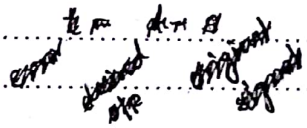
$$w_i(new) = w_i(old) + \alpha t x_i$$

this is change in bias $\rightarrow$ $b(new) = b(old) + \alpha t$

earlier they never used it but slowly they include it also to get the appropriate output

where α = learning rate

t = desired response



Q. What is Perceptron learning Rule OR Explain with example Perceptron learning Rule? 10

Here, the learing signal is the difference between the desired and actual response of a neuron.

The perceptron learning rule is explained as follows →

There are n number of input training vectors, with their associated target (desired) values t(n).
The target is either +1 or −1.

The output 'y' is obtained by applying activation function net input y_{in}.

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

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

If $y \neq t$, then

$$w(new) = w(old) + \alpha t x$$

where α = learning rate

else

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

weights can be initialized at any values in this method.

The perceptron rule convergence theorem states that "If there is a weight vector W, such that $f(x(n).W) = t(n)$ $\forall n$

then for any starting vector w_1, the perceptron learning rule will converge to a weight vector that gives

the correct response for all training patterns.

And this learning takes place within finite number of steps.

§ Architecture →

In the original perceptron network, the output obtained from the associator unit is a binary vector, and hence the output can be taken as input signal to the response unit, and classification can be performed

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

The associator unit behaves like the input unit.

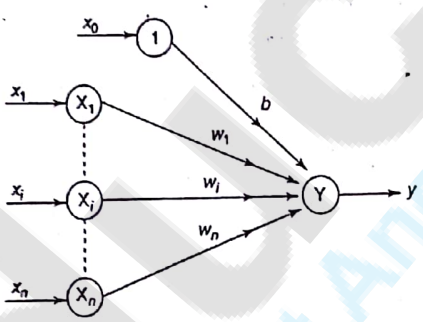


Figure 3-2 Single classification perceptron network

According to the figure, there are n input neurons,

1 output neuron &

1 bias

The input layer and output layer neurons are connected through a directed communication link, which is associated with weights.

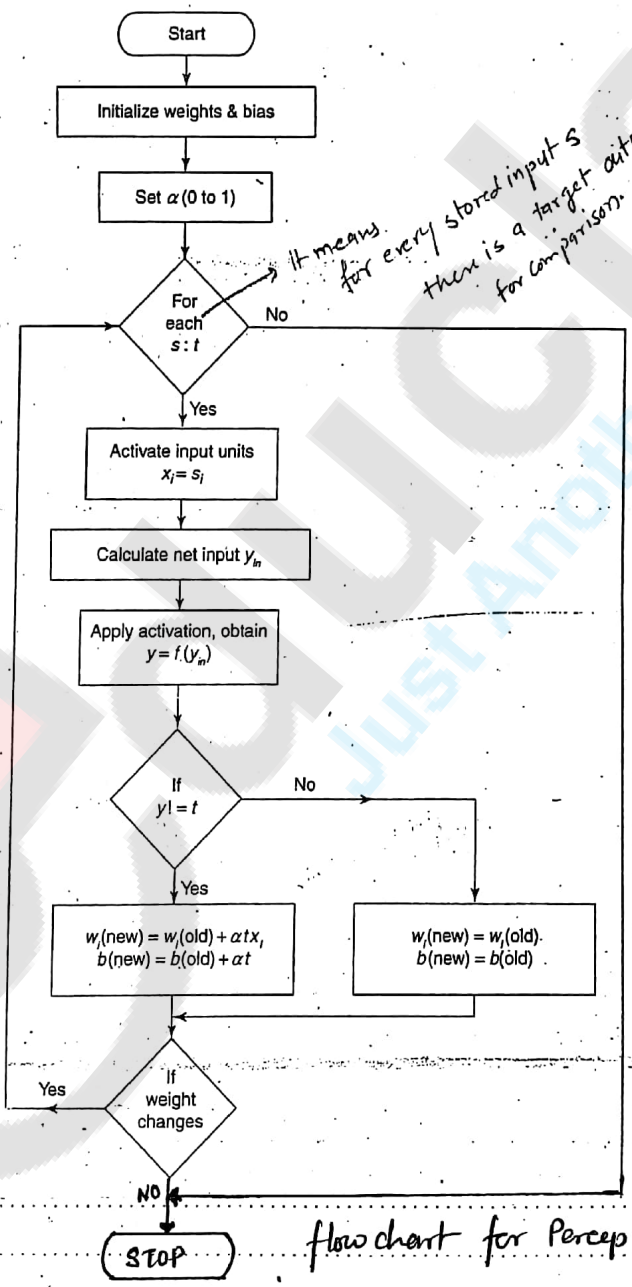
The goal of perceptron network is to classify the input pattern as a member or not a member to a particular class.

(U2) (33) 21

§ Flowchart for training process →

The following flowchart presents the flow of the training process -

1) The basic initialization is done to perform the training process.
2) The entire loop of the training process continues until the training input pair is presented to the network.
3) The weight updation is done on the basis of the comparison b/w the calculated and desired output.
4) The loop is terminated if there is no change in weight.



flowchart for Perceptron N/w with single output

Bernoulli

(U2) (34)

Perceptron Training Algorithm for single output class

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

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

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

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

$$x_i = s_i$$

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

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

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

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

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

If $y \neq t$, then

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

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

else, we have

$$w_i(\text{new}) = w_i(\text{old})$$

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

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

Architecture for Perceptron network for several output cl

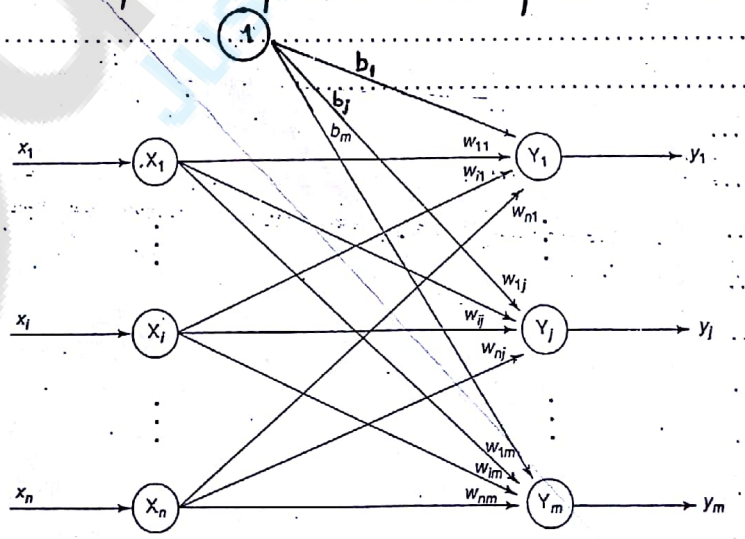


Figure 3-4 Network architecture for perceptron network for several output classes.

(02) (35)

§ Perceptron Training Algorithm for Multiple Output Classes

Step 0: Initialize the weights, biases and learning rate suitably.

Step 1: Check for stopping condition: if it is false, perform Steps 2–6.

Step 2: Perform Steps 3–5 for each bipolar or binary training vector pair *s:t.*

Step 3: Set activation (identity) of each input unit $i = 1$ to n:

$$x_i = s_i$$

Step 4: Calculate output response of each output unit $j = 1$ to m: First, the net input is calculated as

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

Then activations are applied over the net input to calculate the output response:

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

Step 5: Make adjustment in weights and bias for $j = 1$ to m and $i = 1$ to n.

If $t_j \neq y_j$, then

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

else, we have

$$w_{ij}(\text{new}) = w_{ij}(\text{old})$$
$$b_j(\text{new}) = b_j(\text{old})$$

Step 6: Test for the stopping condition, i.e., if there is no change in weights then stop the training process, else start again from Step 2.

§ Perceptron Network Testing Algorithm →

Once the training process is completed the network perfor-

3

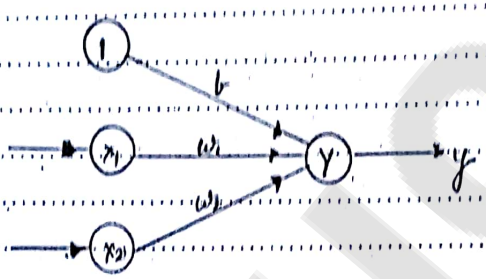
Q. Implement AND function using perceptron networks for bipolar inputs and targets.

The truth table for AND func^{n} with bipolar inputs and targets is →

x_1	x_2	t
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Solu^{n} → Perceptron

The network architecture for 2 inputs & 1 bias is



The input patterns are presented to the network one by one.

When all 4 input patterns are presented, then 1 epoch (cycle) (duration of time) is said to be completed.

The initial weights and threshold are set to zero i.e. $w_1 = w_2 = b = 0$ and $\theta = 0$

The learing rate $\alpha = 1$

Step1 → for 1st Input pattern $x_1 = 1$ $x_2 = 1$ calculate the net input

$$b + \sum x_i w_i$$

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

$$= 0 + 1 \times 0 + 1 \times 0$$

$$y_{in} = 0$$

Step2 → To get output y apply activation func^{n} over y_{in} →

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

Here we have taken $\theta = 0$

$\therefore y_{in} = 0$

$\therefore y = 0$

Step3 → check whether $t = y$

Here $t = 1$ and $y = 0$

so $t \neq y$

$\therefore$ weight updation takes place.

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

$w_1(new) = w_1(old) + \alpha t x_1 = 0 + 1*1*1 = 1$

$w_2(new) = w_2(old) + \alpha t x_2 = 0 + 1*1*1 = 1$

$b(new) = b(old) + \alpha t = 0 + 1*1 = 1$

$\therefore$ change in weights are

$\Delta w_1 = \alpha t x_1 = 1$

$\Delta w_2 = \alpha t x_2 = 1$

$\Delta b = \alpha t = 1$

$\therefore$ weight $w_1 = 1$, $w_2 = 1$ &
bias $b = 1$
are the final weights after 1st input pattern is presented.

The same process is repeated for all the input patterns.

12

Step 4 → The process can be stopped when at

Stopping Criteria for NN Training

① When all the target becomes equal to the calculated output i.e. $t = y$

OR

② When a separating line is obtained using the final weights for separating the +ve response from -ve response.

Input				Weights			Net input	Calculated o/p	Weight changes			Target (t)
	x_1	x_2	1	w_1 (a)	w_2 (a)	b (a)	y_{in}	y	Δw_1	Δw_2	Δb	
EPOCH – I	1	1	1	1	1	1	0	0	1	1	1	1
	1	-1	1	0	2	0	1	1	-1	1	-1	-1
	-1	1	1	1	1	-1	2	1	-1	-1	-1	-1
	-1	-1	1	0	0	-2	-3	-1	1	1	1	-1
EPOCH – II	1	1	1	1	1	-1	1	1	0	0	0	1
	1	-1	1	1	1	-1	-1	-1	0	0	0	-1
	-1	1	1	1	1	-1	-1	-1	0	0	0	-1
	-1	-1	1	1	1	-1	-3	-1	0	0	0	-1

Hence $t = y$ ∴ no change in weight ∴ stop process.

(12) (41)

Q. Determine the weights after four steps of training for Perceptron learning rule of a single neuron network starting with initial weights $w = [0\ 0]^t$, Inputs as $X_1 = [2\ 2]^t$, $X_2 = [1\ -2]^t$, $X_3 = [-2\ 2]^t$ $X_4 = [-1\ 1]^t$, $d_1 = 0$, $d_2 = 1$, $d_3 = 0$, $d_4 = 1$ and $c = 1$. (12)

Sol^n → Given weights are $w_1 = 0$, $w_2 = 0$

∵ bias is not given ∴ $b = 0$

also threshold is not given ∴ $\theta = 0$.

d_1, d_2, d_3 & d_4 are targets.

c is α i.e. learning rate that is 1.

Input signals and targets →

x_1	x_2	t
2	2	0
1	−2	1
−2	2	0
−1	1	1

∵ Activation funct^n is also not given.

So, as we know if it is said Perceptron then we know that in Perceptron binary step with fixed threshold θ is used.

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

Now, we will draw the table for 4 EPOCHS →

Inputs or training pairs are –

x_1	x_2	x_3	x_4	t
1	-2	3	-1	-1
0	-1	2	-1	1
-2	0	-3	-1	-1

given learning rate $c = 1$ (α)

$y_{in} = b + x_1w_1 + x_2w_2 + x_3w_3 + x_4w_4$, $f(y_{in}) = \begin{cases} 1 & y_{in} > 0 \\ 0 & y_{in} = 0 \\ -1 & y_{in} < 0 \end{cases}$

x_1	x_2	x_3	x_4	Target t	Input y_{in}	y	$\alpha t x_1$ Δw_1	$\alpha t x_2$ Δw_2	$\alpha t x_3$ Δw_3	$\alpha t x_4$ Δw_4	αt Δb	w_1 (0)	w_2 (0)	w_3 (0)	w_4 (0)	b (0)
EPOCH-I																
1	-2	5	-1	-1	0	0	-1	2	-3	1	-1	-1	2	-3	1	-1
0	-1	2	-1	+1	-10	-1	0	-1	2	-1	1	-1	1	-1	0	0
-2	0	-3	-1	-1	5	1	2	0	3	1	-1	1	1	2	1	-1 → w_1
EPOCH-II																
1	-2	3	-1	-1	3	1	-1	2	-3	1	-1	0	3	-1	2	-2
0	-1	2	-1	1	-9	-1	0	-1	2	-1	1	0	2	1	1	-1
-2	0	-3	-1	-1	-4	-1	0	0	0	0	0	0	2	1	1	-1 → w_2
EPOCH-III																
1	-2	3	-1	-1	-3	-1	0	0	0	0	0	0	2	1	1	-1
0	-1	2	-1	1	-2	-1	0	1	-2	1	1	0	3	-1	2	0
-2	0	-3	-1	-1	1	1	2	0	3	1	-1	2	3	2	3	-1 → w_3
EPOCH-IV																
1	-2	3	-1	-1	-2	-1	0	0	0	0	0	2	3	2	3	-1
0	-1	2	-1	1	-3	-1	0	-1	2	-1	1	2	2	4	2	0
-2	0	-3	-1	-1	-18	-1	0	0	0	0	0	2	2	4	2	0

→ Is this w_4 i.e. $[3\ 2\ 6\ 1]^t$ here I am not getting same [illegible]

13

Q. Implement the Perceptron rule training using $f(net) = sgn(net)$, $c = 1$, and the following data specifying the initial weights $W1$, and the two training pairs.

it is a learning rate

$W1 = [0\ 1\ 0]^t$ → d → desired output = t (target)

$X1 = [2\ 1\ -1]^t$ $d1 = -1$;

$X2 = [0\ 1\ -1]^t$ $d2 = 1$;

Repeat the training sequence until two correct responses in a row are achieved. (10)

2 EPOCHS

Soln → Given $\theta = 0$ $w_2 = 1$ & $w_3 = 0$

Inputs and Targets are

x_1	x_2	x_3	t
2	1	-1	-1
0	1	-1	1

and learning rate $\alpha = 1$.

∵ bias is not given so, we assume it as 0.

threshold $\theta = 0$.

∵ $f(net) = sgn(net)$ $f(a) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$

↳ 'sgn' is the step (bipolar step) function whose outputs are 1 and -1.

Now, we will prepare table →

Input x_1	Input x_2	Target t	net i/p y_{in}	Calculated o/p y	weigh changes Δw_1	Δw_2	Δw_3	Δb	weights w_1	w_2	w_3	b
EPOCH-I												
2	1	-1	-1	1	-2	-1	1	-1	-2	0	1	-1
0	1	-1	1	0	1	0	0	0	-2	1	2	0
EPOCH-II												
2	1	-1	-1	-4	-1	0	0	0	-2	2	1	-1
0	1	-1	1	0	1	0	0	0	-2	2	1	-1

Here all $t = y$ ∴ no change in weight ∴ stop process.

In 2nd EPOCH, all the calculated o/p's become equal to targets and the n/ has converged.

(12) (13)

Q. Find the weight required to perform the following classification using Perceptron network. The vectors (1, 1, 1, 1) and (−1, 1, −1, −1) are belonging to the class with target o/p 1 and vectors (1, 1, 1, −1) and (1, −1, −1, 1) are belonging to the class with target output 0. Assume learning rate to be 1 and initial weights to be zero (Calculate upto three epochs)
This is Ex 4 page 85 with little change here (16)

§ ADAptive LInear NEuron (ADALINE) →

In Adaline, the input-output relationship is linear.

Adaline uses bipolar activation for its input signals.

The weights b/w the input and the output are adjustable.

The bias in Adaline acts like an adjustable weight, whose connection is from a unit with activations being always 1.

Adaline is a net which has only one o/p unit.

The Adaline network is trained for using delta rule.

The delta rule is also called as Least Mean Square rule or Widrow-Hoff rule.

This learning rule is found to minimize the mean-squared error b/w the activation and the target value.

Q. Explain the Delta Learning Rule (4)

§ Delta Rule for Single output Unit →

This rule is very similar to Perceptron learning rule. only their origins are different.

The Perceptron learning rule originates from the Hebbian assumption while
The Delta rule is derived from the gradient-descent method.

The Perceptron learning rule stops after a finite number of learning steps but the gradient-descent approach continues forever, converging only asymptotically to the solution.

The Delta rule updates the weights between the connection to minimize the difference b/w the desired (target) output and actual output (i/p).

The major aim is to minimize the error over all training patterns.

This is done by reducing the error for each pattern one at a time.

The Delta rule for adjusting the weight of ith pattern

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

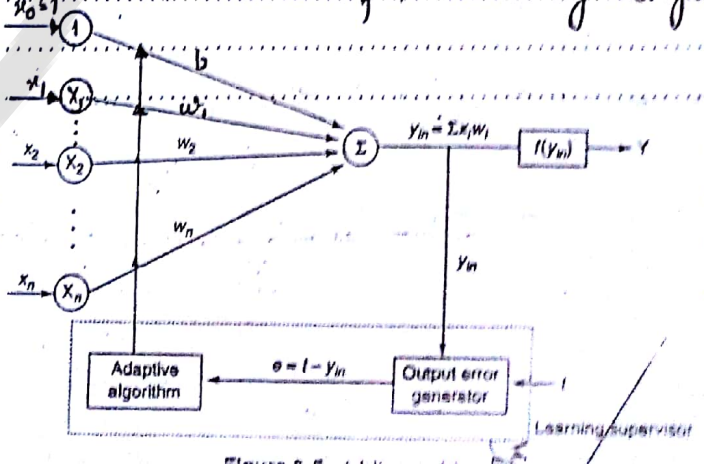
where Δw_i = weight change

α = learning rate.

y_{in} = net i/p to o/p unit
i.e. $y = \sum_{i=1}^{n} x_i w_i$

t = target o/p

§ Architecture → The Adaline model compares the actual o/p with the target o/p and on the basis of the training algo



(22) (45)

§ Training Algo →

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

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

Step 2: Perform Steps 3–5 for each bipolar training pair $s:t$.

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

$$x_i = s_i$$

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

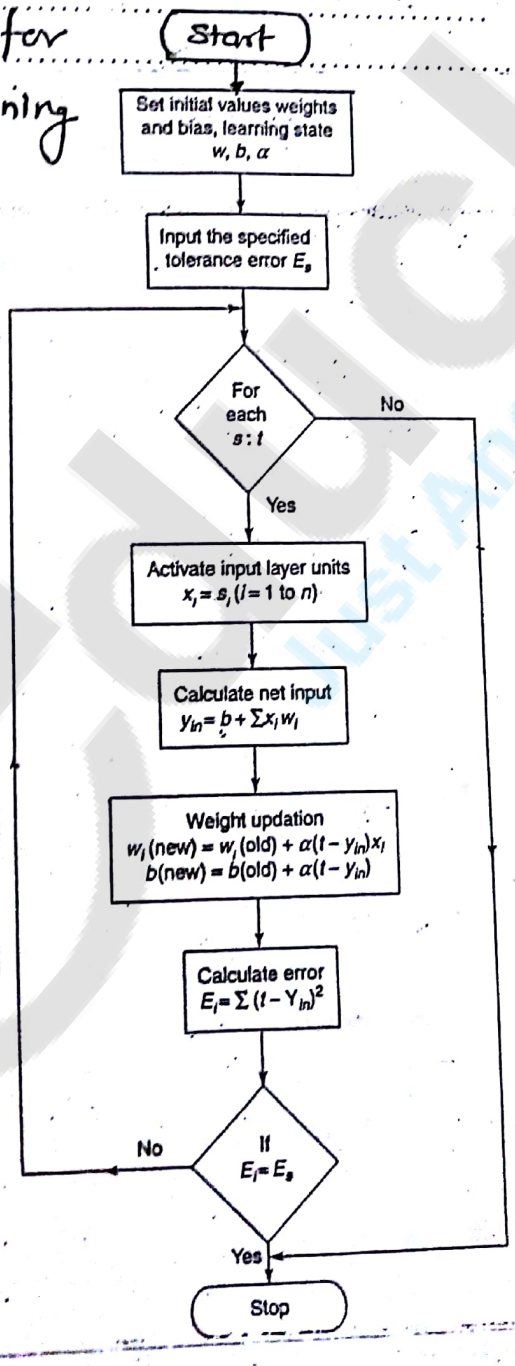
$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

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

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

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

§ Flowchart for ADALINE Training process →



(02) (16)

Q. Implement OR function with bipolar inputs and targ using Adaline Network–

The truth table for OR funct with bipolar inputs & Target

x_1	x_2	1	t
1	1	1	1
1	-1	1	1
-1	1	1	1
-1	-1	1	-1

Soluⁿ →

Initially all the weights and links are assumed to be small random values like 0·1 and the learning rate is also set to 0·1.

The weights are calculated until the least mean square error is obtained.

The initial weights are taken to be $w_1 = w_2 = b = 0·1$

and the learning rate $\alpha = 0·1$

For the first input sample

$x_1 = 1 \quad x_2 = 1 \quad t = 1$

we calculate the net input as

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

$$= b + \sum_{i=1}^{2} x_i w_i$$

$$= 0·1 + x_1 w_1 + x_2 w_2$$

$$= 0·1 + 1 * 0·1 + 1 * 0·1$$

$$= 0·1 + 0·1 + 0·1$$

$$y_{in} = 0·3$$

Now will compute the $(t - y_{in})$

$t - y_{in}$

$= 1 - 0.3$

$= 0.7$

∴ weight updation -

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

Δw_i = weight change

$\therefore w_1(new) = w_1(old) + \alpha(t - y_{in}) x_1$

$= 0.1 + 0.1 * 0.7 * 1$ $\quad [\Delta w_1 = \alpha(t - y_{in}) x_1]$

$= 0.1 + 0.07$

$\therefore w_1(new) = 0.17$

$w_2(new) = w_2(old) + \Delta w_i$

$= 0.1 + 0.1 * 0.7 * 1$ $\quad [\Delta w_2 = \alpha(t - y_{in}) x_2]$

$= 0.1 + 0.07$

$= 0.17$

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

$= 0.1 + 0.1 * 0.7$ $\quad [\Delta b = \alpha(t - y_{in})]$

$= 0.1 + 0.07$

$= 0.17$

Now we compute the error -

$E = (t - y_{in})^2 = (0.7)^2 = 0.49$

So final weights after presenting first input to the network are

$w = [0.17 \quad 0.17 \quad 0.17]$

& $E = 0.49$

These calculations are performed for all the input samples and the error is calculated.

Once EPOCH is completed when all the input patterns are presented.

Summing up all the errors obtained for each input sample during 1 epoch will give the total mean square error of that epoch.

The network training is continued until this error is minimized to a very small value.

EPOCH	Total Mean Square Error
Epoch1	3.02
Epoch2	1.938
Epoch3	1.5506
Epoch4	1.417
Epoch5	1.377

Inputs			Target	Net input		Weight changes			Weights			Error
x_1	x_2	1	t	y_{in}	$(t-y_{in})$	Δw_1	Δw_2	Δb	w_1 (0.1	0.1	0.1)	$(t-y_{in})^2$
EPOCH-1												
1	1	1	1	0.3	0.7	0.07	0.07	0.07	0.17	0.17	0.17	0.49
1	−1	1	1	0.17	0.83	0.083	−0.083	0.083	0.253	0.087	0.253	0.69
−1	1	1	1	0.087	0.913	−0.0913	0.0913	0.0913	0.1617	0.1783	0.3443	0.83
−1	−1	1	−1	0.0043	−1.0043	0.1004	0.1004	−0.1004	0.2621	0.2787	0.2439	1.01
EPOCH-2												
1	1	1	1	0.7847	0.2153	0.0215	0.0215	0.0215	0.2837	0.3003	0.2654	0.046
1	−1	1	1	0.2488	0.7512	0.7512	−0.0751	0.0751	0.3588	0.2251	0.3405	0.564
−1	1	1	1	0.2069	0.7931	−0.7931	0.0793	0.0793	0.2795	0.3044	0.4198	0.629
−1	−1	1	−1	−0.1641	−0.8359	0.0836	0.0836	−0.0836	0.3631	0.388	0.336	0.699
EPOCH-3												
1	1	1	1	1.0873	−0.0873	−0.087	−0.087	−0.087	0.3543	0.3793	0.3275	0.0076
1	−1	1	1	0.3025	+0.6975	0.0697	−0.0697	0.0697	0.4241	0.3096	0.3973	0.487
−1	1	1	1	0.2827	0.7173	−0.0717	0.0717	0.0717	0.3523	0.3813	0.469	0.515
−1	−1	1	−1	−0.2647	−0.7353	0.0735	0.0735	−0.0735	0.4259	0.4548	0.3954	0.541
EPOCH-4												
1	1	1	1	1.2761	−0.2761	−0.0276	−0.0276	−0.0276	0.3983	0.4272	0.3678	0.076
1	−1	1	1	0.3389	0.6611	0.0661	−0.0661	0.0661	0.4644	0.3611	0.4339	0.437
−1	1	1	1	0.3307	0.6693	−0.0669	0.0669	0.0699	0.3974	0.428	0.5009	0.448
−1	−1	1	−1	−0.3246	−0.6754	0.0675	0.0675	−0.0675	0.465	0.4956	0.4333	0.456
EPOCH-5												
1	1	1	1	1.3939	−0.3939	−0.0394	−0.0394	−0.0394	0.4256	0.4562	0.393	0.155
1	−1	1	1	0.3634	0.6366	0.0637	−0.0637	0.0637	0.4893	0.3925	0.457	0.405
−1	1	1	1	0.3609	0.6391	−0.0639	0.0639	0.0639	0.4253	0.4654	0.5215	0.408
−1	−1	1	−1	−0.3603	−0.6397	0.064	0.064	−0.064	0.4893	0.5204	0.4575	0.409

(U2) (49)

§ Multiple Adaptive Linear Neurons → [Madaline]

Madaline model consists of many Adalines in parallel with a single output unit.

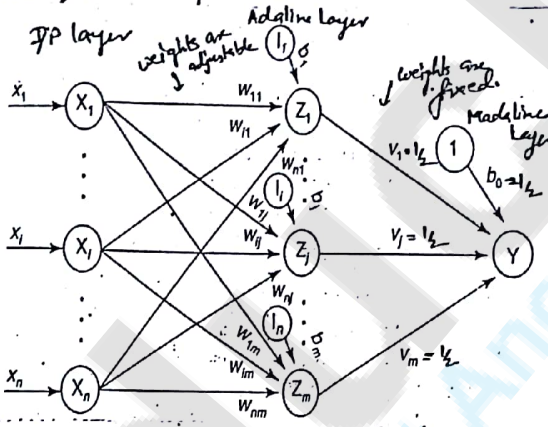
Its o/p value is based on certain selection rules.

If it is majority vote rule then
o/p have answer as either True or False

If it is AND rule then

o/p have answer as True if and only if both the i/ps are true.

The weights that are connected from Adaline layer to Madaline layer are fixed, +ve and have equal values.



The weights between the input layer and the Adaline layer are adjusted during the training process.

The Adaline and Madaline layer neuron has bias as 1

The training process for a Madaline system is similar to Adaline.

§ Architecture →

A simple Madaline has "n" units of i/p layer
"m" units of Adaline layer &
1 unit of the Madaline layer.

Each neuron of Adaline & Madaline has bias as 1.

Adaline layer is present between the input layer and the Madaline (output) layer. Hence, Adaline layer can be considered as a hidden layer.

This hidden layer gives the net computational capability.

The Adaline & Madaline models are widely used in communication system of adaptive equalizers and adaptive noise cancellation circuits.

Training Algorithm —

Only the weights b/w hidden layer & i/p layer are adjustable.

weights for the o/p units are fixed.

Weights for o/p units are denoted by $v_1, v_2 \dots v_m = \frac{1}{2}$

and bias $b_0 = \frac{1}{2}$

The activation for Adaline and Madaline units is given by —

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

(02) (51) 31

Algo →

Step 0: Initialize the weights. The weights entering the output unit are set as above. Set initial small random values for Adaline weights. Also set initial learning rate α.

Step 1: When stopping condition is false, perform Steps 2–3.

Step 2: For each bipolar training pair $s:t$, perform Steps 3–7.

Step 3: Activate input layer units. For $i = 1$ to n,

$$x_i = s_i$$

Step 4: Calculate net input to each hidden Adaline unit:

$$z_{inj} = b_j + \sum_{i=1}^{n} x_i w_{ij}, \quad j = 1 \text{ to } m$$

Step 5: Calculate output of each hidden unit:

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

Step 6: Find the output of the net:

$$y_{in} = b_0 + \sum_{j=1}^{m} z_j v_j$$

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

Step 7: Calculate the error and update the weights.

1. If $t = y$, no weight updation is required.
2. If $t \neq y$ and $t = +1$, update weights on z_j, where net input is closest to 0 (zero):

$$b_j(\text{new}) = b_j(\text{old}) + \alpha(1 - z_{inj})$$
$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha(1 - z_{inj})x_i$$

3. If $t \neq y$ and $t = -1$, update weights on units z_k whose net input is positive:

$$w_{ik}(\text{new}) = w_{ik}(\text{old}) + \alpha(-1 - z_{ink})x_i$$
$$b_k(\text{new}) = b_k(\text{old}) + \alpha(-1 - z_{ink})$$

Step 8: Test for the stopping condition. (If there is no weight change or weight reaches a satisfactory level, or if a specified maximum number of iterations of weight updation have been performed then stop; or else continue).

14

Q. Using Madaline network, implement XOR funct with bipolar inputs and targets. Assume the required parameters for training of the network.

The training pattern for XOR funct is

x_1	x_2	1	t
1	1	1	-1
1	-1	1	1
-1	1	1	1
-1	-1	1	-1

Initial weights and bias are

$[w_{11} \ w_{21} \ b_1] = [0.05 \ 0.2 \ 0.3]$

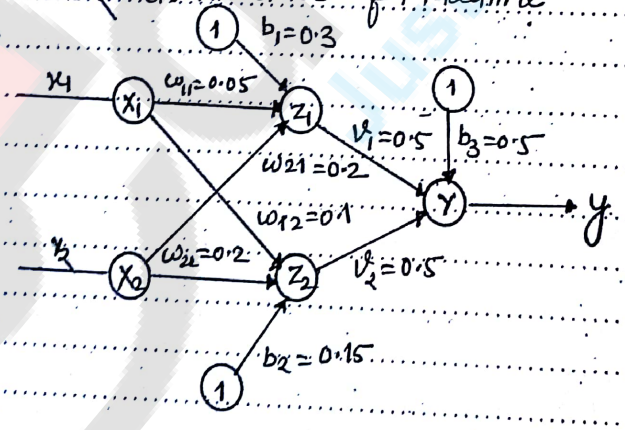
$[w_{12} \ w_{22} \ b_2] = [0.1 \ 0.2 \ 0.15]$ and

$[v_1 \ v_2 \ b_3] = [0.5 \ 0.5 \ 0.5]$

learning rate $\alpha = 0.5$

sol^n →

Network architecture of Madaline



→ Calculate net input to the hidden units:

$$z_{in1} = b_1 + x_1 w_{11} + x_2 w_{21}$$
$$= 0.3 + 1 \times 0.05 + 1 \times 0.2 = 0.55$$
$$z_{in2} = b_2 + x_1 w_{12} + x_2 w_{22}$$
$$= 0.15 + 1 \times 0.1 + 1 \times 0.2 = 0.45$$

Calculate the output z_1, z_2 by applying the activation over the net input computed.

The activation function is given by

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

$\therefore z_1 = f(z_{in}1) = f(0.55) = 1$

$\therefore z_2 = f(z_{in}2) = f(0.45) = 1$

After computing the output of the hidden units, find the net input entering into the output unit:

$$y_{in} = b_3 + z_1 v_1 + z_2 v_2$$
$$= 0.5 + 1 \times 0.5 + 1 \times 0.5$$
$$= 0.5 + 0.5 + 0.5$$
$$= 1.5$$

Now apply the activation function over the net input y_{in} to calculate the output y.

$$y = f(y_{in}) = f(1.5) = 1$$

$\because t \neq y$, weight updation has to be performed.

$$w_{11}(new) = w_{11}(old) + \alpha (t - z_{in}1)\, x_1$$
$$= 0.05 + 0.5\,(-1 - 0.55) \times 1 = -0.725$$

$$w_{12}(new) = w_{12}(old) + \alpha (t - z_{in}2)\, x_1$$
$$= 0.1 + 0.5 \times (-1 - 0.45) \times 1 = -0.625$$

$$b_1(new) = b_1(old) + \alpha (t - z_{in}1)$$
$$= 0.3 + (0.5)(-1 - 0.55) = -0.475$$

(U2) (56)

$w_{21}(new) = w_{21}(old) + \alpha(t - z_{in}^1)\, x_2$

$= 0.2 + 0.5\,(-1 - 0.55) \times 1 = -0.575$

$w_{22}(new) = w_{22}(old) + \alpha(t - z_{in}^2)\, x_2$

$= 0.2 + 0.5\,(-1 - 0.45) \times 1 = -0.525$

$b_2(new) = b_2(old) + \alpha(t - z_{in}^2)$

$= 0.15 + 0.5\,(-1 - 0.45) = -0.575$

All the weights and bias between the input layer a
hidden layer are adjusted.

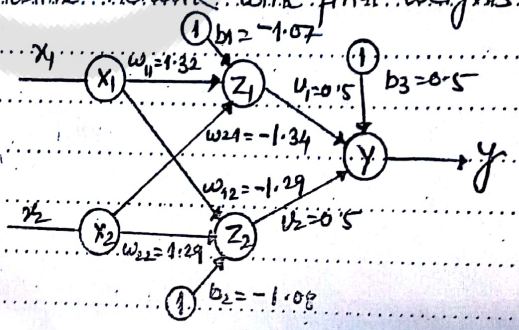
This completes the training for the first EPOCH.

The same process is repeated until the weight converg

Here, weight converges at the end of 3 EPOCH.

Inputs x_1	x_2	1	Target (t)	z_{in1}	z_{in2}	z_1	z_2	y_{in}	y	w_{11}	w_{21}	b_1	w_{12}	w_{22}	b_2
EPOCH-1															
1	1	1	−1	0.55	0.45	1	1	1.5	1	−0.725	−0.58	−0.475	−0.625	−0.525	−0.575
1	−1	1	1	−0.625	−0.675	−1	−1	−0.5	−1	0.0875	−1.39	0.34	−0.625	−0.525	−0.575
−1	1	1	1	−1.1375	−0.475	−1	−1	−0.5	−1	0.0875	−1.39	0.34	−1.3625	0.2125	0.1625
−1	−1	1	−1	1.6375	1.3125	1	1	1.5	1	1.4065	−0.069	−0.98	−0.207	1.369	−0.994
EPOCH-2															
1	1	1	−1	0.3565	0.168	1	1	1.5	1	0.7285	−0.75	−1.66	−0.791	−0.207	−1.58
1	−1	1	1	−0.1845	−3.154	−1	−1	−0.5	−1	1.3205	−1.34	−1.068	−0.791	0.785	−1.58
−1	1	1	1	−3.728	−0.002	−1	−1	−0.5	−1	1.3205	−1.34	−1.068	−1.29	0.785	−1.08
−1	−1	1	−1	−1.0495	−1.071	−1	−1	−0.5	−1	1.3205	−1.34	−1.068	−1.29	1.29	−1.08
EPOCH-3															
1	1	1	−1	−1.0865	−1.083	−1	−1	−0.5	−1	1.32	−1.34	−1.07	−1.29	1.29	−1.08
1	−1	1	1	1.5915	−3.655	1	−1	0.5	1	1.32	−1.34	−1.07	−1.29	1.29	−1.08
−1	1	1	1	−3.728	1.501	−1	1	0.5	1	1.32	−1.34	−1.07	−1.29	1.29	−1.08
−1	−1	1	−1	−1.0495	−1.701	−1	−1	−0.5	−1	1.32	−1.34	−1.07	−1.29	1.29	−1.08

Madaline network with final weights — weights are coming same, i.e. weights are converging



5 Back Propagation Network → (BPN)

This learning algorithm is one of the most important development in Neural Network.

This learning algorithm is applied to multilayer feed-forward networks.

This is a method where the error is propagated back to the hidden unit.

When the hidden layers are increased the network training becomes more complex.

To update weights, the error must be calculated.

The error, which is the difference between the actual (calculated) and the desired (target) output, is easily measured at the output layer.

At the hidden layer, there is no direct information of the error. Therefore, other technique should be used to calculate an error at the hidden layer, which will cause minimization of the o/p error, and this is the ultimate goal.

The training of the BPN is done in 3 stages —

1. The feed-forward of the input training pattern,
2. The calculation and back-propagation of the error.
3. Updation of weights.

(V2) (58)

§ Architecture →

A back-propagation neural network is a multilayer feed-forward neural network consists of

- an i/p layer
- a hidden layer
- an o/p layer

The neurons present in the hidden and output layers have biases, whose activation is always 1.

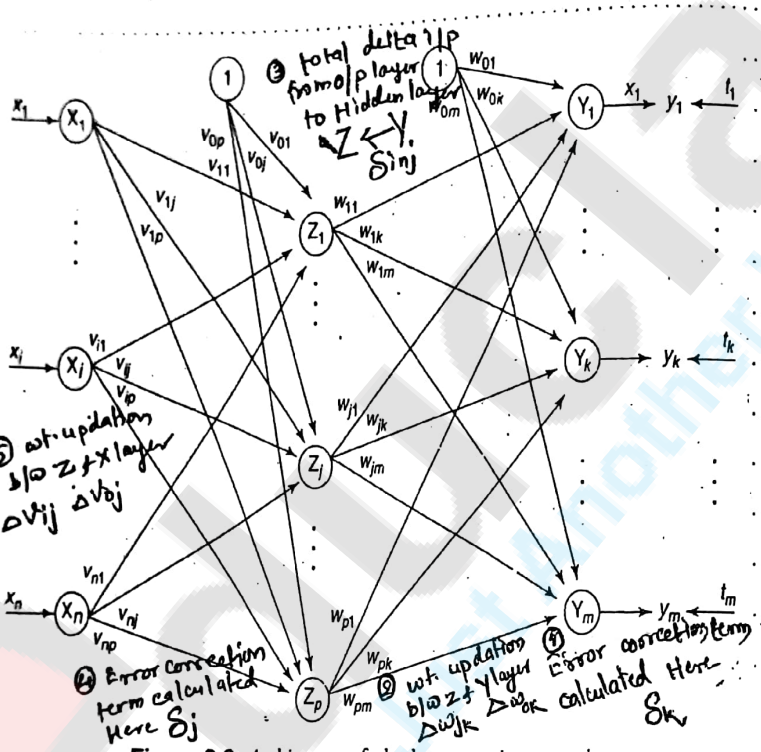


Figure 3-9 Architecture of a back-propagation network.

During the back-propagation phase of learning, signals are sent in the reverse direction.

The outputs obtained from the net could be either binary (0, 1) or

bipolar (−1, +1)

The activation function could be any funct which Increases monotonically and is also differentiable

Back Propagation Explaination →

During the back-propagation phase of learning, signals are sent in the reverse direction.

1. Feed forward Phase I
2. Weight updation Phase III

After this explain

3. Back Propagation of error Phase II

1. Computation of error correction term of o/p layer

δ_k at o/p layer ∵ o/p unit is receiving Targe response t_k
∴ o/p unit will compute error correction term

$$\delta_k = \underbrace{(t_k - y_k)}_{\text{error}} \times \underbrace{f'(y_{ink})}_{\text{o/p of the layer}}$$

2. wt. updation — On the basis of δ_k, will calculate Δw_{jk} & Δw_{ok}

$$\Delta w_{jk} = \alpha \delta_k z_j \qquad \Delta w_{ok} = \alpha \delta_k$$

3. Again this δ_k has to be sent backward layer who is sending & o/p layer Y_k to whome Hidden layer Z

So, sum of all δ_k inputs to Z layer is

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k \underbrace{w_{jk}}_{\text{with wts}}$$

∵ every signal moves with weight

∵ δ_{inj} is comming from o/p layer to Hidden layer
i.e. k^{th} layer → j^{th} layer.

δ_j at Hidden layer

4. Computation of error correction term at hidden layer

$$\delta_j = \underbrace{\delta_{inj}}_{\text{error}} \times \underbrace{f'(z_{inj})}_{\text{o/p of the layer}}$$

5. wt. updation — On the basis of δ_w, will calculate Δv_{ij} & Δv_{oj}

$$\Delta v_{ij} = \alpha \delta_j x_i \quad , \quad \Delta v_{oj} = \alpha \delta_j$$

(V2) (59) 36

Flowchart for Training Process → Back Propagation Algorithm

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

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

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

Feed-forward phase (Phase I):

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

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

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

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

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

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

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

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

and apply the activation function to compute output signal

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

Back-propagation of error (Phase II):

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

Error computation

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

The derivative $f'(y_{ink})$ can be calculated as in Section 2.3.3. On the basis of the calculated error correction term, update the change in weights and bias:

wt. updation blw Z & Y layer

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

Also, send δ_k to the hidden layer backwards.

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

Error summation

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

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

Error computation

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

The derivative $f'(z_{inj})$ can be calculated as discussed in Section 2.3.3 depending on whether binary or bipolar sigmoidal function is used. On the basis of the calculated δ_j, update the change in weights and bias:

wt. updation

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

algorithm (10)