Introduction

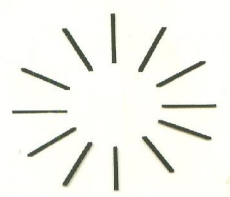
Artificial Neural Network

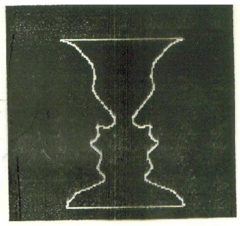
Why ANN?

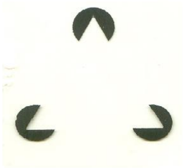
Some tasks can be done easily (effortlessly) by humans but are hard by conventional computers or Von Neumann machine with algorithmic approach

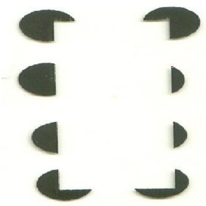
- Pattern recognition
- Content addressable recall
- Approximate, common sense reasoning (driving, playing piano, baseball player)

ILLUSORY BORDERS





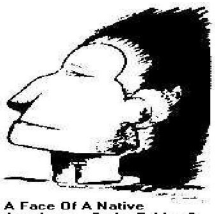




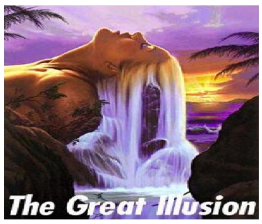
ILLUSIONS

Old Woman...Or Young Girl?
hint: the old woman's nose is the young girl's nose and chin

Woman In Vanity... Or Skull?
hint: moves farther a bit from the screen and blink to see the skull or the woman (looking at the mirror)



A Face Of A Native American... Or An Eskimo?



Introduction

Von Neumann machine

- One or a few high speed (ns) processors with considerable computing power
- One or a few shared high speed buses for communication
- Sequential memory access by address
- Hard to be adaptive

Human Brain

- Large # (10^{11}) of low speed processors (ms) with limited computing power
- Large # (10^{15}) of low speed connections. Massively parallel structure
- Content addressable recall (CAM)
- Adaptation by changing the connectivity

ANN usefulness and capabilities

- Nonlinearity
  - interconnection of nonlinear neurons
- Input output mapping
  - Learning mechanism helps(with a teacher)
  - Adjust parameters to have correct responses
- Adaptivity
  - Adapt parameters to the changes in the surrounding

Biological Neuron Model

The human brain consists of a large number, more than a billion of neural cells that process information. Each cell works like a simple processor. The massive interaction between all cells and their parallel processing only makes the brain's abilities possible.

Human Brain

- Human Brain has about 10^{11} Neuron
  (about the same number as the stars in our galaxy)
- A neuron has about 1000 to 10,000 synapses
- A Neural Network is more important than individual neurons
- Knowledge is acquired by the Neural Network through Learning Process
- Synaptic Weights are used to store the Knowledge.

Dendrites are branching fibers that extend from the cell body or soma.

Soma or cell body of a neuron contains the nucleus and other structures, support chemical processing and production of neurotransmitters.

Axon is a singular fiber carries information away from the soma to the synaptic sites of other neurons (dendrites and somas), muscles, or glands.

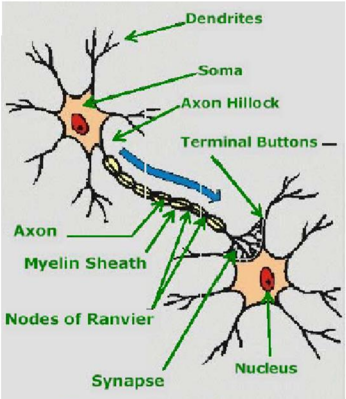


Fig. Structure of Neuron

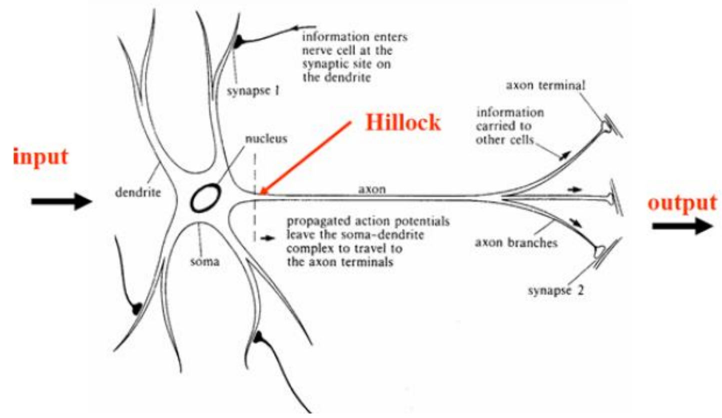
Synapse is the point of connection between two neurons or a neuron and a muscle or a gland. Electrochemical communication between neurons takes place at these junctions.

Similarity with brain expected in ANN

- Knowledge is acquired by learning process
- Interneuron connection strengths known as synaptic weights are used to store knowledge
- Process used to perform learning is called learning algorithm the function of which is to adjust weights.

Information flow in a Neural Cell

The input /output and the propagation of information are shown below.



Biological neuron

- Dendrites receive activation from other neurons.
- Soma processes the incoming activations and converts them into output activations.
- Axons act as transmission lines to send activation to other neurons.
- Synapses the junctions allow signal transmission between the axons and dendrites.
- The process of transmission is by diffusion of chemicals called neuro-transmitters.

McCulloch-Pitts introduced a simplified model of this real neurons.

Introduction ANN

- **What is an (artificial) neural network**
  - A set of **nodes** (units, neurons, processing elements)
    - Each node has input and output
    - Each node performs a simple computation by its **node function**
  - **Weighted connections** between nodes
    - Connectivity gives the structure/architecture of the net
    - What can be computed by a NN is primarily determined by the connections and their weights
  - A very much simplified version of networks of neurons is animal nerve systems

Artificial Neuron - Basic Elements

Neuron consists of three basic components - weights, thresholds, and a single activation function.

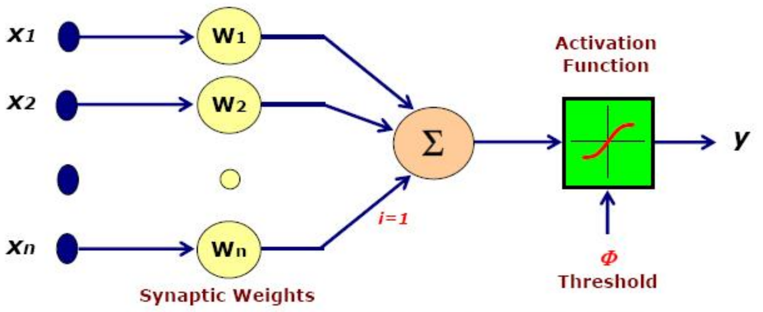


Fig Basic Elements of an Artificial Linear Neuron

- **Weighting Factors w**

The values $w_1, w_2, \ldots w_n$ are weights to determine the strength of input vector $X = [x_1, x_2, \ldots, x_n]^T$. Each input is multiplied by the associated weight of the neuron connection X^T **W**. The +ve weight excites and the -ve weight inhibits the node output.

$$I = X^T.W = x_1 w_1 + x_2 w_2 + \ldots + x_n w_n = \sum_{i=1}^{n} x_i w_i$$

- **Threshold Φ**

The node's internal threshold Φ is the magnitude offset. It affects the activation of the node output **y** as:

$$Y = f(I) = f\{\sum_{i=1}^{n} x_i w_i - \Phi_k\}$$

To generate the final output **Y** , the sum is passed on to a non-linear filter **f** called Activation Function or Transfer function or Squash function which releases the output **Y**.

- **Threshold for a Neuron**

In practice, neurons generally do not fire (produce an output) unless their total input goes above a threshold value.

The total input for each neuron is the sum of the weighted inputs to the neuron minus its threshold value. This is then passed through the sigmoid function. The equation for the transition in a neuron is :

$a = 1/(1 + \exp(-x))$ where

$x = \sum_i a_i w_i - Q$

a is the activation for the neuron

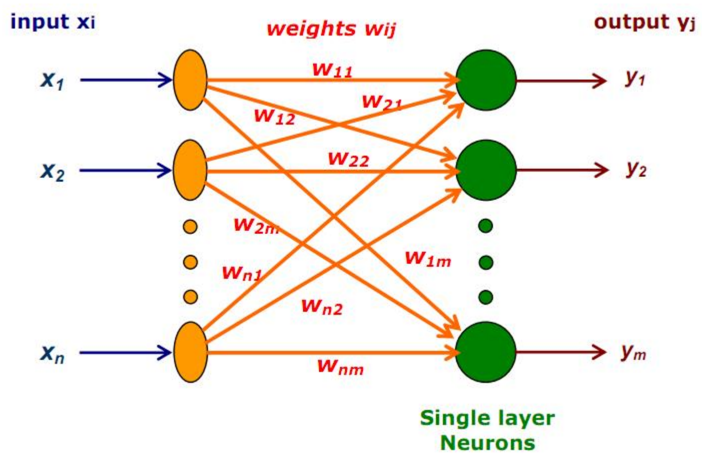
a_i is the activation for neuron ***i***

w_i is the weight

Q is the threshold subtracted

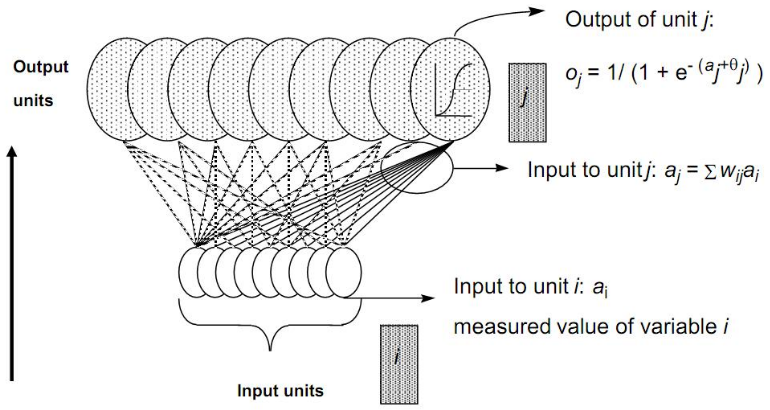
Neural Network Architectures

Single Layer Network



DJSCE/EXTC/VII/Nueral Network /Vishakha Kelkar

Single Layer Network



Multilayer Network

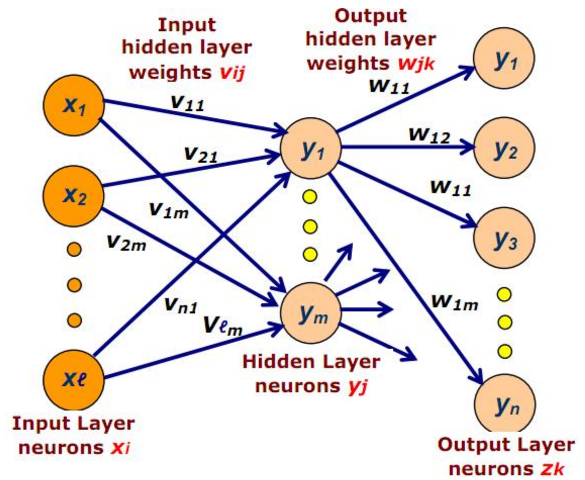
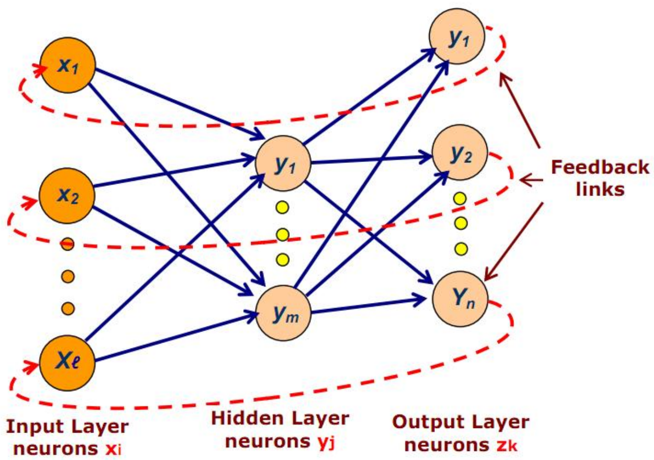


Fig. Multilayer feed-forward network in $(\ell - m - n)$ configuration.

Recurrent Neural Network

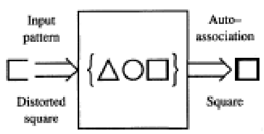


Applications

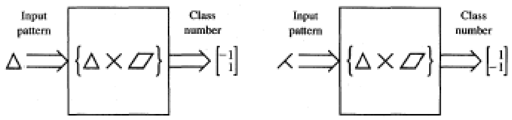
Classification

- Classify fruits: Apple and Orange
- I/Ps: Shape , texture , colour
- O/P: Apple , Orange

Association



Classification & recognition



Learning Methods In Neural Networks

Learning Methods

The learning methods in neural networks are classified into three basic types :

- Supervised Learning,
- Unsupervised Learning and
- Reinforced Learning

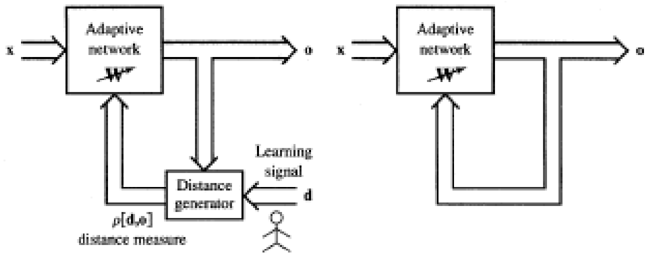
These three types are classified based on :

- presence or absence of **teacher** and
- the information provided for the system to learn.

These are further categorized, based on the **rules** used, as

- Hebbian,
- Gradient descent,
- Competitive and
- Stochastic learning.

Learning



- **Activation Function**

  An activation function **f** performs a mathematical operation on the signal output. The most common activation functions are:

  - Linear Function,
  - Threshold Function,
  - Piecewise Linear Function,
  - Sigmoidal (S shaped) function,
  - Tangent hyperbolic function

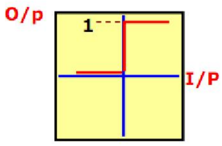
  The activation functions are chosen depending upon the type of problem to be solved by the network.

•Neuron in same layer have same Activation function .
•Linear and Nonlinear activation functions are used.
•Nonlinear functions are used in Multilayer neurons.

Threshold Function

A threshold (hard-limiter) activation function is either a binary type or a bipolar type as shown below.

binary threshold



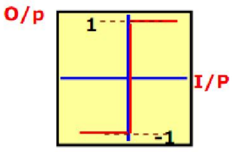
Output of a binary threshold function produces :

1 if the weighted sum of the inputs is +ve,

0 if the weighted sum of the inputs is -ve.

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

bipolar threshold



Output of a bipolar threshold function produces :

1 if the weighted sum of the inputs is +ve,

-1 if the weighted sum of the inputs is -ve.

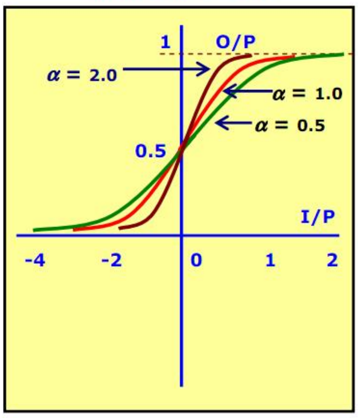
$$Y = f(I) = \begin{cases} 1 & \text{if } I \geq 0 \\ -1 & \text{if } I < 0 \end{cases}$$

Neuron with hard limiter activation function is called McCulloch-Pitts model.

Sigmoidal Function (S-shape function)

The nonlinear curved S-shape function is called the sigmoid function. This is most common type of activation used to construct the neural networks. It is mathematically well behaved, differentiable and strictly increasing function.

Sigmoidal function



A sigmoidal transfer function can be written in the form:

$$Y = f(I) = \frac{1}{1 + e^{-\alpha I}}, \quad 0 \leq f(I) \leq 1$$

$$= 1/(1 + \exp(-\alpha I)), \quad 0 \leq f(I) \leq 1$$

This is explained as

≈ **0** for large **-ve** input values,

1 for large **+ve** values, with a smooth transition between the two.

α is slope parameter also called shape parameter; symbol the **λ** is also used to represented this parameter.

Sigmoidal Functions

- Used in Multilayer Networks like back Propagation Networks
- Binary
- Bipolar Sigmoidal Function

$$Y = B(I) = 2 * f(I) - 1$$

$$= 2 * \frac{1}{1-\exp(-\alpha I)} - 1$$

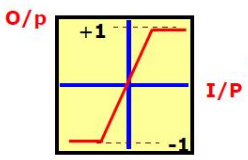
$$= \frac{2-1-\exp(-\alpha I)}{1-\exp(-\alpha I)}$$

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

Piecewise Linear Function

This activation function is also called saturating linear function and can have either a binary or bipolar range for the saturation limits of the output. The mathematical model for a symmetric saturation function is described below.

Piecewise Linear

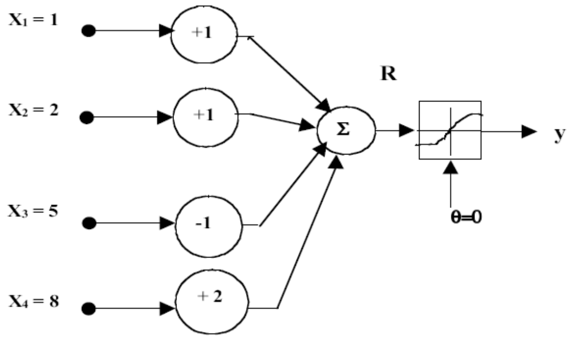


This is a sloping function that produces :

- **-1** for a -ve weighted sum of inputs,
- **1** for a +ve weighted sum of inputs.
- $\propto$ **I** proportional to input for values between **+1** and **-1** weighted sum,

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

Consider the following network consists of four inputs with the weights as shown



The output R of the network, prior to the activation function stage, is calculated as follows:

$$R = W^T.X = \begin{bmatrix} 1 & 1 & -1 & 2 \end{bmatrix}.\begin{bmatrix} 1 \\ 2 \\ 5 \\ 8 \end{bmatrix} = 14 \qquad (2.7)$$

With a binary activation function, and a sigmoid function, the outputs of the neuron are respectively as follow:

$\mathbf{y}(\mathit{Threshold}) = 1;$

$\mathbf{y}(\mathit{Sigmoid}) = 1.5*2^{-8}$

Biological neuron

- Dendrites receive activation from other neurons.
- Soma processes the incoming activations and converts them into output activations.
- Axons act as transmission lines to send activation to other neurons.
- Synapses the junctions allow signal transmission between the axons and dendrites.
- The process of transmission is by diffusion of chemicals called neuro-transmitters.

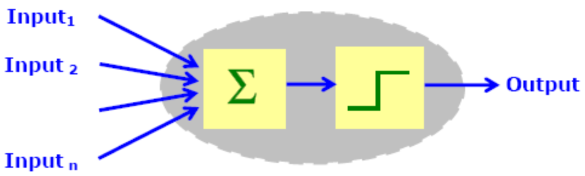
McCulloch-Pitts introduced a simplified model of this real neurons.

McCulloch Pitts Neuron

- The activation is binary
- Connection path is expiatory if the weight on the path is one and inhibitory if its zero
- If the net input to the neuron is greater than the threshold the neuron fires.
- Weights on the neurons are adjusted to perform simple logical functions.
- They could not adapt with application(can not be trained)
- Cant work with non binary inputs

- **The McCulloch-Pitts Neuron**

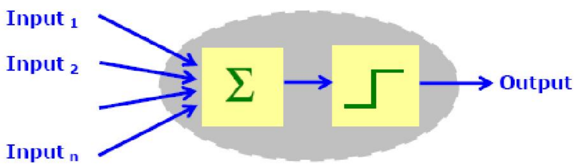
This is a simplified model of real neurons, known as a Threshold Logic Unit.



- A set of input connections brings in activations from other neurons.
- A processing unit sums the inputs, and then applies a non-linear activation function (i.e. squashing / transfer / threshold function).
- An output line transmits the result to other neurons.

McCulloch-Pitts (M-P) Neuron Equation

McCulloch-Pitts neuron is a simplified model of real biological neuron.



Simplified Model of Real Neuron
(Threshold Logic Unit)

The equation for the output of a McCulloch-Pitts neuron as a function of **1** to **n** inputs is written as

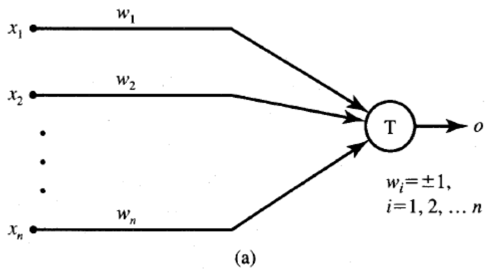
$$\text{Output} = \text{sgn}\left(\sum_{i=1}^{n} \text{Input}_i - \Phi \right)$$

where Φ is the neuron's activation threshold.

If $\sum_{i=1}^{n} \text{Input}_i \geq \Phi$ then Output = 1

If $\sum_{i=1}^{n} \text{Input}_i < \Phi$ then Output = 0

Simple neuron



DJSCE/EXTC/VII/Nueral Network /Vishakha Kelkar

Hebb Network

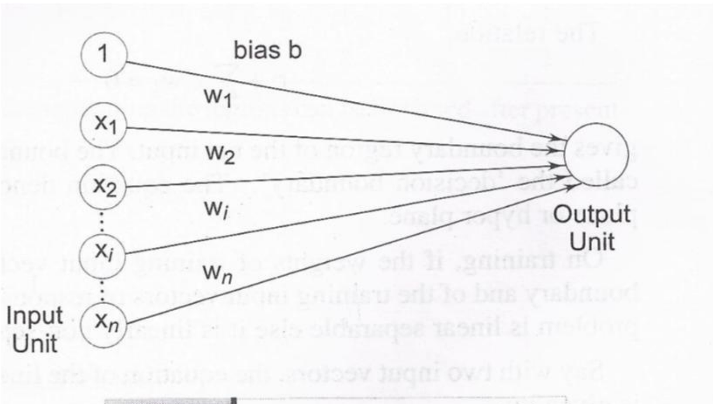


Fig. 3.8 *Architecture of a Hebb Net*

DJSCE/EXTC/VII/Nueral Network /Vishakha Kelkar

Hebb Rule

Step 1: Initialize all weights and bias to zero

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

Step 2: For each input training vector and target output pair (S, t) perform Steps 3 – 6.

Step 3: Set activations for input units with input vector.

$x_i = S_i$ ($i = 1$ to n)

Step 4: Set activation for output unit with the output neuron

$y = t$

Step 5: Adjust the weights by applying Hebb rule,

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

Step 6: Adjust the bias

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

This algorithm requires only one pass through the training set.

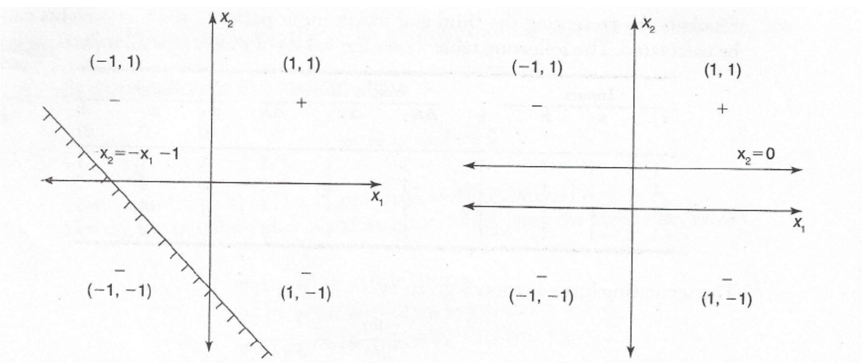
Hebb Rule

Design a Hebb net to implement logical AND function (use bipolar inputs and targets).
Solution: The training data for the AND function is

Inputs			Target
x_1	x_2	b	y
1	1	1	1
1	−1	1	−1
−1	1	1	−1
−1	−1	1	−1

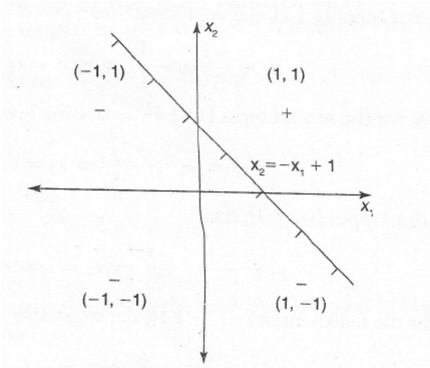
$$w_i(\text{new}) = w_i(\text{old}) + x_i y$$
$$w_1(\text{new}) = w_1(\text{old}) + x_1 y = 0 + 1 \times 1 = 1$$
$$w_2(\text{new}) = w_2(\text{old}) + x_2 y = 0 + 1 \times 1 = 1$$
$$b(\text{new}) = b(\text{old}) + y = 0 + 1 = 1$$

Inputs				Weight changes			Weights		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 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



(A) First input

(B) Second input



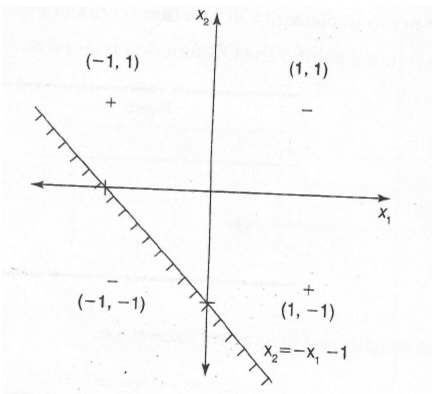
(C) Third and fourth inputs

Design a Hebb net to implement OR function (consider bipolar inputs and targets).

Solution: The training pair for the OR function is given as

Inputs			*Targets*
x_1	x_2	b	y
1	1	1	1
1	−1	1	1
−1	1	1	1
−1	−1	1	−1

Inputs				Weight changes			Weights		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 0)
1	1	1	1	1	1	1	1	1	1
1	−1	1	1	1	−1	1	2	0	2
−1	1	1	1	−1	1	1	1	1	3
−1	−1	1	−1	1	1	−1	2	2	2



Use the Hebb rule method to implement XOR function (take bipolar inputs and targets).

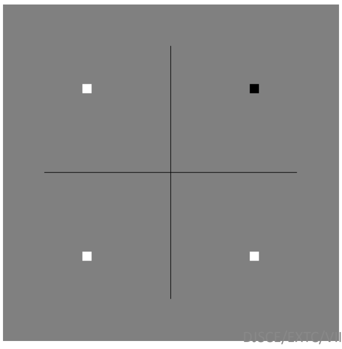
Solution: The training patterns for an XOR function are shown below:

Inputs			*Target*
x_1	x_2	b	y
1	1	1	−1
1	−1	1	1
−1	1	1	1
−1	−1	1	−1

Inputs				*Weight changes*			*Weights*		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 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

Example: AND

- Here is a representation of the AND function
- White means *false*, black means *true* for the output
- -1 means *false*, +1 means *true* for the input



-1 AND -1 = false

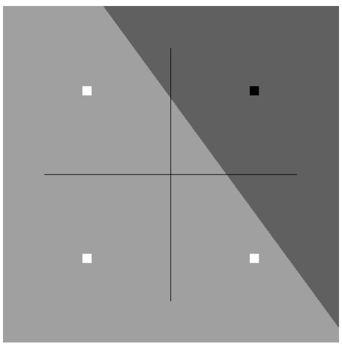
-1 AND +1 = false

+1 AND -1 = false

+1 AND +1 = true

Example: AND continued

- A linear decision surface separates *false* from *true* instances



Decision Boundaries for AND and OR

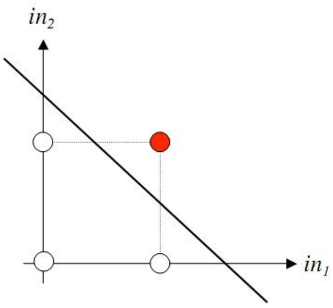
We can easily plot the decision boundaries we found by inspection last lecture:

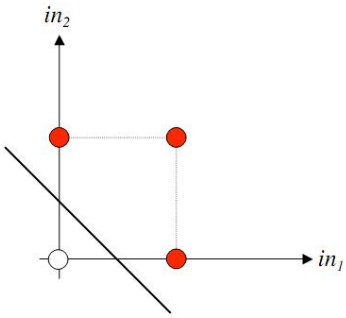
AND

$w_1 = 1,\ w_2 = 1,\ \theta = 1.5$

OR

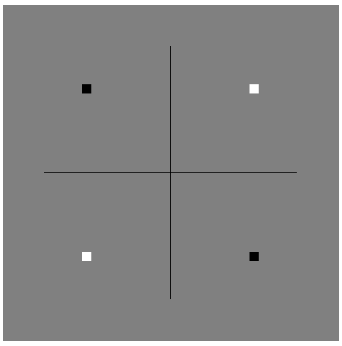
$w_1 = 1,\ w_2 = 1,\ \theta = 0.5$





Example: XOR

- Here's the XOR function:



-1 XOR -1 = *false*

-1 XOR +1 = *true*

+1 XOR -1 = *true*

+1 XOR +1 = *false*

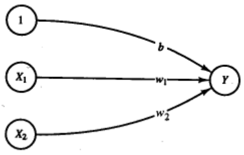
Perceptrons cannot learn such *linearly inseparable* functions

Bias and Threshold

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

where

$$\text{net} = b + \sum_i x_i w_i.$$



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

where

$$\text{net} = \sum_i x_i w_i.$$

Perceptron

- Weights and threshold can be determined analytically
- Continuous bipolar and multiple valued versions

Perceptron Learning Rule

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:

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

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

Perceptron Learning Rule

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.

Solved example

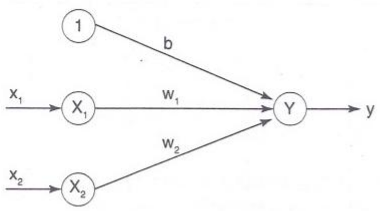
1. Implement AND function using perceptron networks for bipolar inputs and targets.

Solution: The truth table for AND function with bipolar inputs and targets is given below:

x_1	x_2	t
1	1	1
1	−1	−1
−1	1	−1
−1	−1	−1

Solved example

The initial weights and threshold are set to zero, i.e., $w_1 = w_2 = b = 0$ and $\theta = 0$. The learning rate α is set equal to 1.



Solved example

For the first input pattern, $x_1 = 1$, $x_2 = 1$ and $t = 1$, with weights and bias, $w_1 = 0$, $w_2 = 0$ and $b = 0$:

- Calculate the net input

$$y_{in} = b + x_1 w_1 + x_2 w_2$$
$$= 0 + 1 \times 0 + 1 \times 0$$
$$y_{in} = 0$$

- The output y is computed by applying activations over the net input calculated:

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

Here we have taken $\theta = 0$. Hence, when, $y_{in} = 0$, $y = 0$.

Solved example

- Check whether $t = y$. Here, $t = 1$ and $y = 0$, so $t \neq y$, hence no weight updation takes place:

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$
$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_1 = 0 + 1 \times 1 \times 1 = 1$$
$$w_2(\text{new}) = w_2(\text{old}) + \alpha t x_2 = 0 + 1 \times 1 \times 1 = 1$$
$$b(\text{new}) = b(\text{old}) + \alpha t = 0 + 1 \times 1 = 1$$

Here, the change in weights are

$$\Delta w_1 = \alpha t x_1$$
$$\Delta w_2 = \alpha t x_2$$
$$\Delta b = \alpha t$$

The weights $w_1 = 1$, $w_2 = 1$, $b = 1$ are the final weights after first input pattern is presented. The same process is repeated for all the input patterns.

Solved example

- Check whether $t = y$. Here, $t = 1$ and $y = 0$, so $t \neq y$, hence no weight updation takes place:

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$
$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_1 = 0 + 1 \times 1 \times 1 = 1$$
$$w_2(\text{new}) = w_2(\text{old}) + \alpha t x_2 = 0 + 1 \times 1 \times 1 = 1$$
$$b(\text{new}) = b(\text{old}) + \alpha t = 0 + 1 \times 1 = 1$$

Here, the change in weights are

$$\Delta w_1 = \alpha t x_1$$
$$\Delta w_2 = \alpha t x_2$$
$$\Delta b = \alpha t$$

The weights $w_1 = 1$, $w_2 = 1$, $b = 1$ are the final weights after first input pattern is presented. The same process is repeated for all the input patterns.

Solved example

Input			Target	Net input	Calculated output	Weight changes			Weights		
x_1	x_2	1	(t)	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 0)
EPOCH-1											
1	1	1	1	0	0	1	1	1	1	1	1
1	−1	1	−1	1	1	−1	1	−1	0	2	0
−1	1	1	−1	2	1	−1	−1	−1	1	1	−1
−1	−1	1	−1	−3	−1	0	0	0	1	1	−1
EPOCH-2											
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	1	−1

Implement OR function with binary inputs and bipolar targets using perceptron trainin algorithm upto 3 epochs.

Solution: The truth table for OR function with binary inputs and bipolar targets is give below.

x_1	x_2	t
1	1	1
1	0	1
0	1	1
0	0	−1

The perceptron network, which uses perceptron learning rule, is used to train the OF function. The network architecture is shown in Figure 3.

The initial values of the weights and bias are taken as zero, i.e.,

$$w_1 = w_2 = b = 0$$

Also learning rate is 1 and threshold is 0.2. So, the activation function becomes

$$y = \begin{cases} 1 & \text{if } y_{in} > 0.2 \\ 0 & \text{if } -0.2 \le y_{in} \le 0.2 \\ -1 & \text{if } y_{in} < -0.2 \end{cases}$$

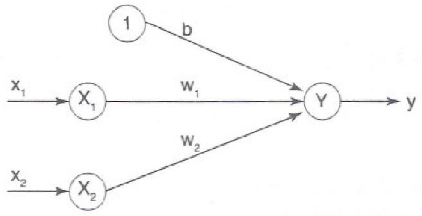


Figure 3 Perceptron network for OR function.

Input			Target	Net input	Calculated output	Weight changes			Weights		
x_1	x_2	1	(t)	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 0)
EPOCH-1											
1	1	1	1	0	0	1	1	1	1	1	[illegible]
1	0	1	1	2	1	0	0	0	1	1	[illegible]
0	1	1	1	2	1	0	0	0	1	1	[illegible]
0	0	1	−1	1	1	0	0	−1	1	1	[illegible]
EPOCH-2											
1	1	1	1	2	1	0	0	0	1	1	[illegible]
1	0	1	1	1	1	0	0	0	1	1	[illegible]
0	1	1	1	1	1	0	0	0	1	1	[illegible]
0	0	1	−1	0	0	0	0	0	1	1	[illegible]
EPOCH-3											
1	1	1	1	1	1	0	0	0	1	1	[illegible]
1	0	1	1	0	0	1	0	1	2	1	[illegible]
0	1	1	1	1	1	0	0	0	2	1	[illegible]
0	0	1	−1	0	0	0	0	−1	2	1	[illegible]

Classification Problem

Find the weights 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 (so have target value 1), vectors (1, 1, 1, −1) and (1, −1, −1, 1) are not belonging to the class (so have target value −1). Assume learning rate as 1 and initial weights as 0.

The truth table for the given vectors is given below.

Input					
x_1	x_2	x_3	x_4	b	Target (t)
1	1	1	1	1	1
−1	1	−1	−1	1	1
1	1	1	−1	1	−1
1	−1	−1	1	1	−1

Classification Problem

Let $w_1 = w_2 = w_3 = w_4 = b = 0$ and the learning rate $\alpha = 1$. Since the threshold $\theta = 0.2$, so the activation function is

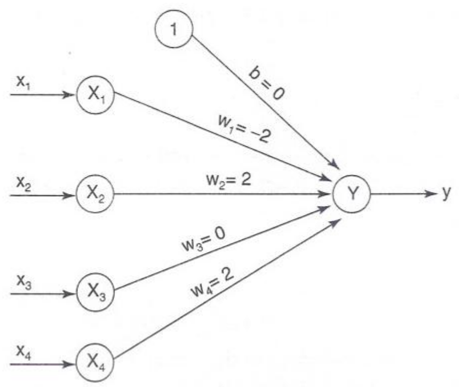
$$y = \begin{cases} 1 & \text{if } y_{in} > 0.2 \\ 0 & \text{if } -0.2 \le y_{in} \le 0.2 \\ -1 & \text{if } y_{in} < -0.2 \end{cases}$$

Input $(x_1\ x_2\ x_3\ x_4\ b)$	Target t	Net input y_{in}	Output Y	Weight changes $(\Delta w_1\ \Delta w_2\ \Delta w_3\ \Delta w_4\ \Delta b)$	Weights $(w_1\ w_2\ w_3\ w_4\ b)$ (0 0 0 0 0)
EPOCH-1					
(1 1 1 1 1)	1	0	0	1 1 1 1 1	1 1 1 1 1
(−1 1 −1 1 1)	1	−1	−1	−1 1 −1 −1 1	0 2 0 0 2
(1 1 1 −1 1)	−1	4	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

Classification Problem

Input $(x_1\ x_2\ x_3\ x_4\ b)$	Target t	Net input y_{in}	Output Y	Weight changes $(\Delta w_1\ \Delta w_2\ \Delta w_3\ \Delta w_4\ \Delta b)$	Weights $(w_1\ w_2\ w_3\ w_4\ b)$ (0 0 0 0 0)
EPOCH-2					
(1 1 1 1 1)	1	0	0	1 1 1 1 1	−1 3 1 1 1
(−1 1 −1 1 1)	1	3	1	0 0 0 0 0	−1 3 1 1 1
(1 1 1 −1 1)	−1	4	1	−1 −1 −1 1 −1	−2 2 0 2 0
(1 −1 −1 1 1)	−1	−2	−1	0 0 0 0 0	−2 2 0 2 0
EPOCH-3					
(1 1 1 1 1)	1	2	1	0 0 0 0 0	−2 2 0 2 0
(−1 1 −1 1 1)	1	2	1	0 0 0 0 0	−2 2 0 2 0
(1 1 1 −1 1)	−1	−2	−1	0 0 0 0 0	−2 2 0 2 0
(1 −1 −1 1 1)	−1	−2	−1	0 0 0 0 0	−2 2 0 2 0

Classification Problem



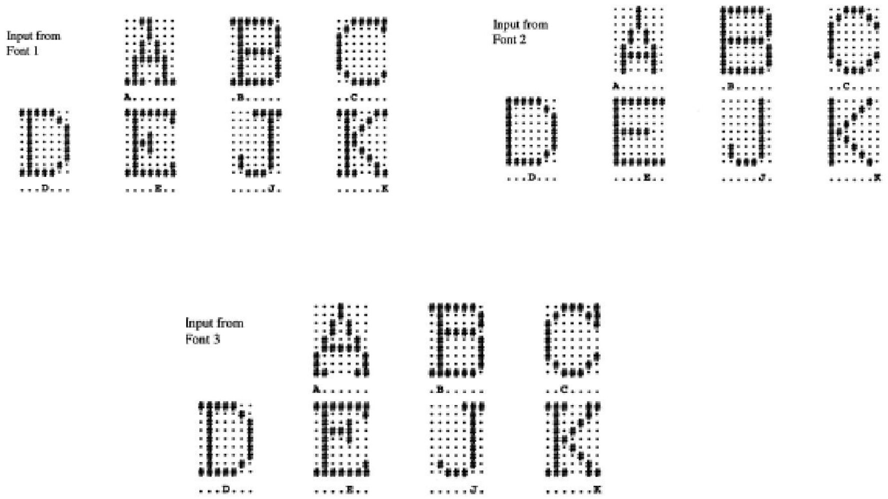
Neuron as classifier

Assume that a set of eight points, P_0, P_1, ..., P_7, in three-dimensional space is available. The set consists of all vertices of a three-dimensional cube as follows:

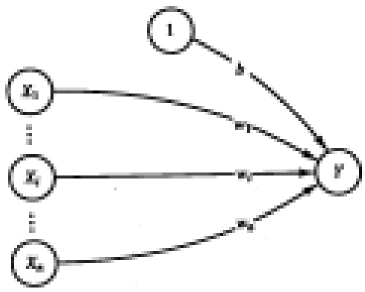
$$\{P_0(-1,-1,-1), P_1(-1,-1,1), P_2(-1,1,-1), P_3(-1,1,1), P_4(1,-1,-1), P_5(1,-1,1), P_6(1,1,-1), P_7(1,1,1)\}$$

Elements of this set need to be classified into two categories. The first category is defined as containing points with two or more positive ones; the second category contains all the remaining points that do not belong to the first category. Accordingly, points P_3, P_5, P_6, and P_7 belong to the first category, and the remaining points to the second category.

Perceptron for character recognition



Perceptron



Perceptron for character recognition

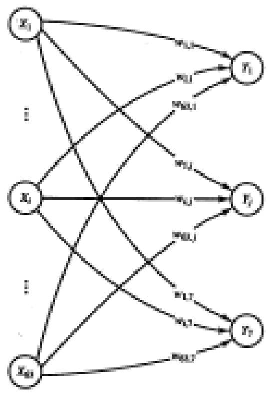
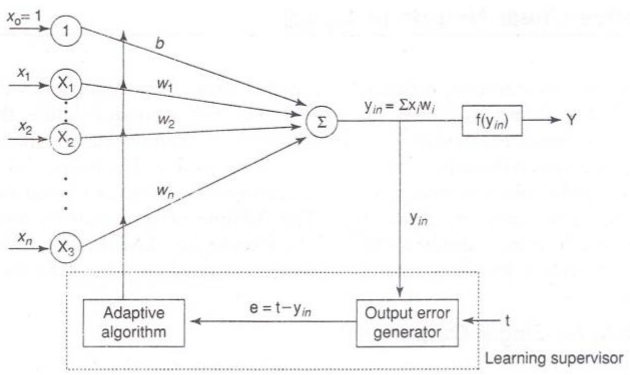


Figure 2.21 Perceptron to classify input into seven categories.

ADALINE MODEL



Delta Rule(Widrow-Hoff Rule)

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.

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

Delta Rule(Widrow-Hoff Rule)

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 a network.

The range of learning rate can be between 0.1 to 1.0.

Widrow-Hoff learning rule

Apple/Banana Example

Training set:

$$\left\{\mathbf{p}_1 = \begin{bmatrix}-1\\1\\-1\end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix}-1\end{bmatrix}\right\} \qquad \left\{\mathbf{p}_2 = \begin{bmatrix}1\\1\\-1\end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix}1\end{bmatrix}\right\}$$

Banana **Apple**

Learning rate: $\eta = 0.4$

Widrow-Hoff learning rule

Learning rate: $\eta = 0.4$

First iteration - $\mathbf{p}_1$ (banana):

$$a(0) = \mathbf{W}(0)\mathbf{p}(0) = \mathbf{W}(0)\mathbf{p}_1 = \begin{bmatrix}0 & 0 & 0\end{bmatrix}\begin{bmatrix}-1\\1\\-1\end{bmatrix} = 0$$

$$e(0) = t(0) - a(0) = t_1 - a(0) = -1 - 0 = -1$$

$$\mathbf{W}(1) = \mathbf{W}(0) + \eta e(0)\mathbf{p}^T(0) = [0\ 0\ 0] + 0.4(-1)\begin{bmatrix}-1\\1\\-1\end{bmatrix}^T = [0.4\ -0.4\ 0.4]$$

Widrow-Hoff learning rule

Second iteration - $\mathbf{p}_2$ (apple):

$$a(1) = \mathbf{W}(1)\mathbf{p}(1) = \mathbf{W}(1)\mathbf{p}_2 = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = -0.4$$

$$e(1) = t(1) - a(1) = t_2 - a(1) = 1 - (-0.4) = 1.4$$

$$\mathbf{W}(2) = [0.4 \ -0.4 \ 0.4] + (0.4)(1.4) \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}^T = [0.96 \ 0.16 \ -0.16]$$

Widrow-Hoff learning rule

Third iteration - $\mathbf{p}_1$ (banana):

$$a(2) = \mathbf{W}(2)\mathbf{p}(2) = \mathbf{W}(2)\mathbf{p}_1 = \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = -0.64$$

$$e(2) = t(2) - a(2) = t_1 - a(2) = -1 - (-0.64) = -0.36$$

$$\mathbf{W}(3) == [1.104 \ 0.016 \ -0.016]$$

Delta Rule(Widrow-Hoff Rule)

Implement OR function with bipolar inputs and targets using Adaline network.

Solution: The truth table for OR function with bipolar inputs and targets is shown below.

x_1	x_2	1	t
1	1	1	1
1	−1	1	1
−1	1	1	1
−1	−1	1	−1

(Learning rate=0.1, Initial weights =all 0.1)

Delta Rule(Widrow-Hoff Rule)

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, $x_1 = 1, x_2 = 1, t = 1$, we calculate the net input as

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

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

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

$$y_{in} = 0.1 + 1 \times 0.1 + 1 \times 0.1$$

$$y_{in} = 0.3$$

Delta Rule(Widrow-Hoff Rule)

Now compute $(t - y_{in}) = (1 - 0.3) = 0.7$.
Updating the weights we obtain,

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

where $\alpha(t - y_{in})x_i$ is called as weight change Δw_i. The new weights are obtained as

$$\begin{aligned} w_1(\text{new}) &= w_1(\text{old}) + \Delta w_1 \\ &= 0.1 + 0.1 \times 0.7 \times 1 \\ &= 0.1 + 0.07 \\ w_1(\text{new}) &= 0.17 \end{aligned}$$

Delta Rule(Widrow-Hoff Rule)

$$\begin{aligned} w_2(\text{new}) &= w_2(\text{old}) + \Delta w_2 \\ &= 0.1 + 0.1 \times 0.7 \times 1 \\ w_2(\text{new}) &= 0.17 \\ b(\text{new}) &= b(\text{old}) + \Delta b \\ &= 0.1 + 0.1 \times 0.7 \\ b(\text{new}) &= 0.17 \end{aligned}$$

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

The final weights after presenting first input sample are

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

and error $E = 0.49$.

Delta Rule(Widrow-Hoff Rule)

Inputs x_1 x_2 1	Target t	Net input y_{in}	$t-y_{in}$	Weight changes Δw_1	Δw_2	Δb	Weights w_1 (0.1	w_2 0.1	b 0.1)	Error $(t-y_{in})^2$
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

Delta Rule(Widrow-Hoff Rule)

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

Delta Rule(Widrow-Hoff Rule)

Epoch	*Total mean square error*
Epoch 1	3.02
Epoch 2	1.938
Epoch 3	1.5506
Epoch 4	1.417
Epoch 5	1.377