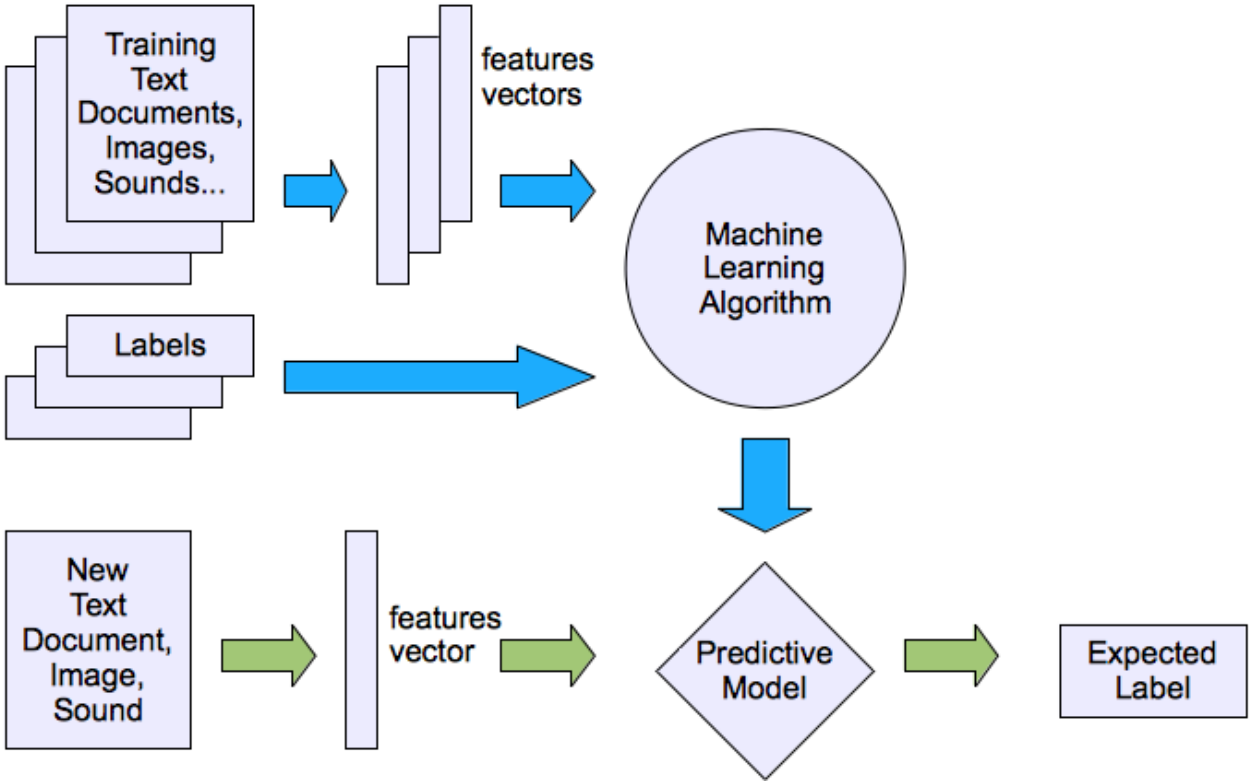
Module 1.2
Basic Model of NEURAL NETWORKS

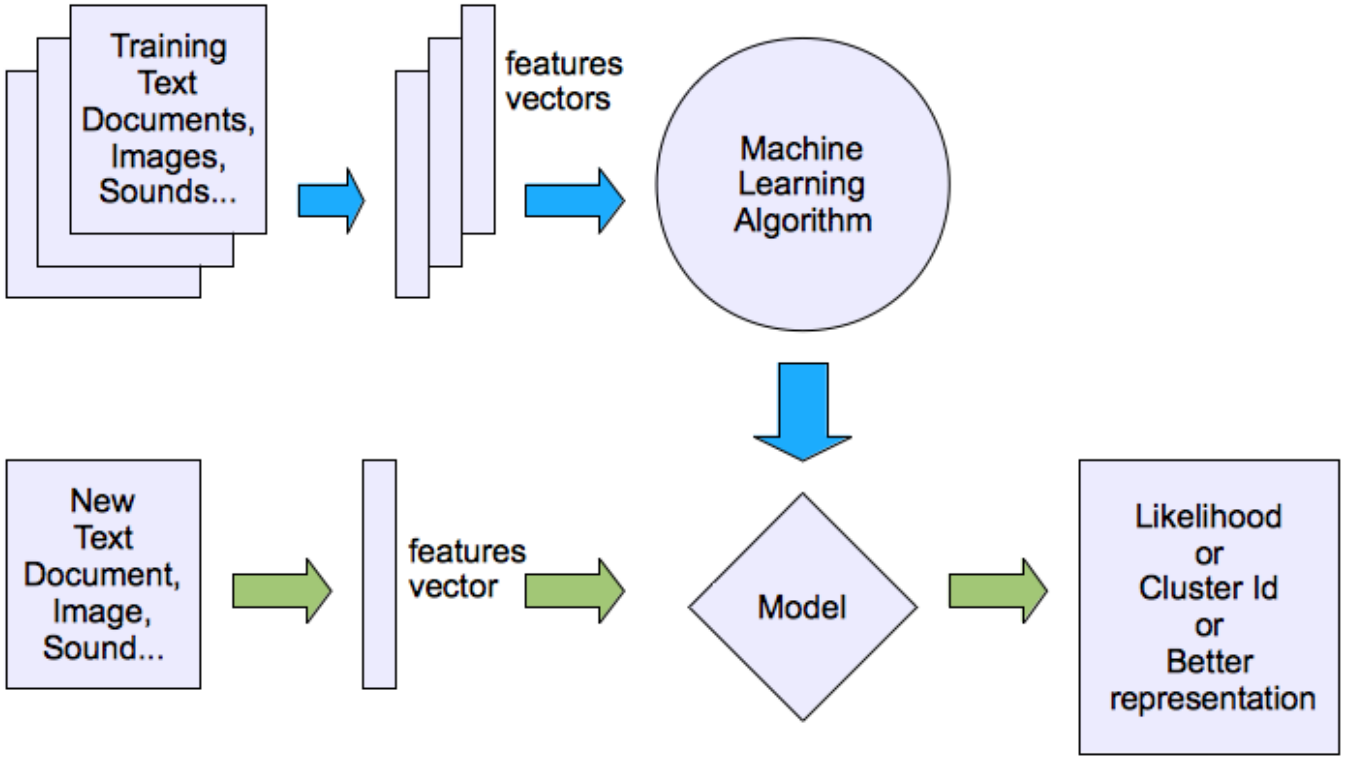
Machine learning structure

- Supervised learning

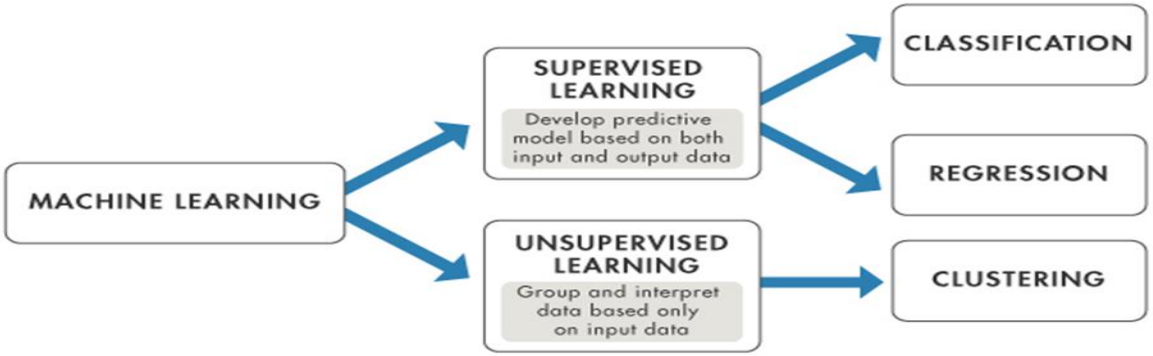


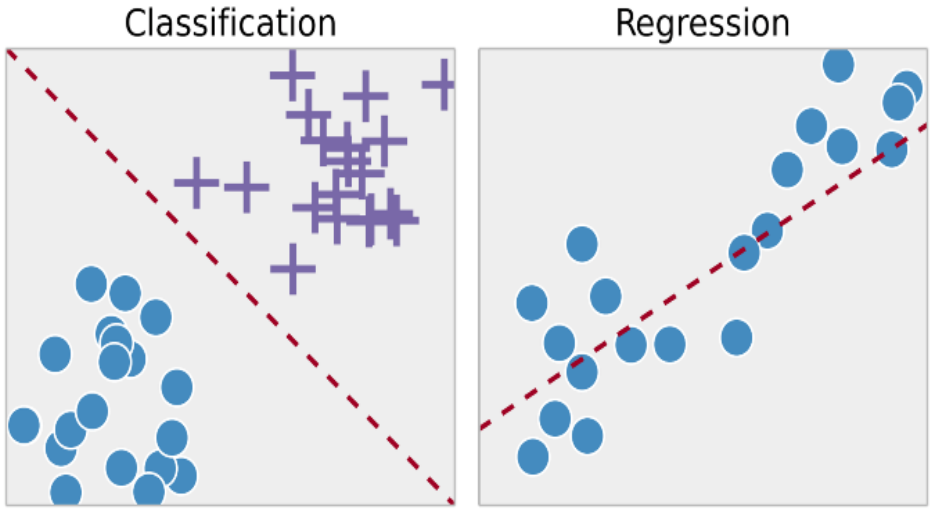
Machine learning structure

- Unsupervised learning



Supervised Vs Unsupervised learning





Reference: https://www.guru99.com/supervised-vs-unsupervised-learning.html

Basic Models of ANN

The models of ANN are specified as

- The models synaptic interconnections
  - ✓Single layer feed forward network
  - ✓Multilayer feed-forward network
  - ✓Single node with its own feedback
  - ✓Single layer recurrent network
  - ✓Multi layer recurrent network
- The training or learning rules adopted for updating and adjusting the connection weights
  - ✓Supervised learning
  - ✓Unsupervised learning
  - ✓Reinforcement learning
- Their activation function
  - ✓Identity function
  - ✓Binary step function
  - ✓Bipolar step function
  - ✓Sigmoidal function
  - ✓Ramp function

ANN Notations

x_i:	Activation of unit X_i, input signal.
y_j:	Activation of unit Y_j, $y_j = f(y_{inj})$
w_{ij}:	Weight on connection from unit X_i to unit Y_j.
b_j:	Bias acting on unit j. Bias has a constant activation of 1.
W:	Weight matrix, $W = \{w_{ij}\}$
y_{inj}:	Net input to unit Y_j given by $y_{inj} = b_j + \sum_i x_i w_{ij}$
$\|\|x\|\|$:	Norm of magnitude vector X.
θ_j:	Threshold for activation of neuron Y_j.

ANN Notations

S: Training input vector, $S = (s_1, \ldots, s_i, \ldots, s_n)$

T: Training output vector, $T = (t_1, \ldots, t_j, \ldots, t_n)$

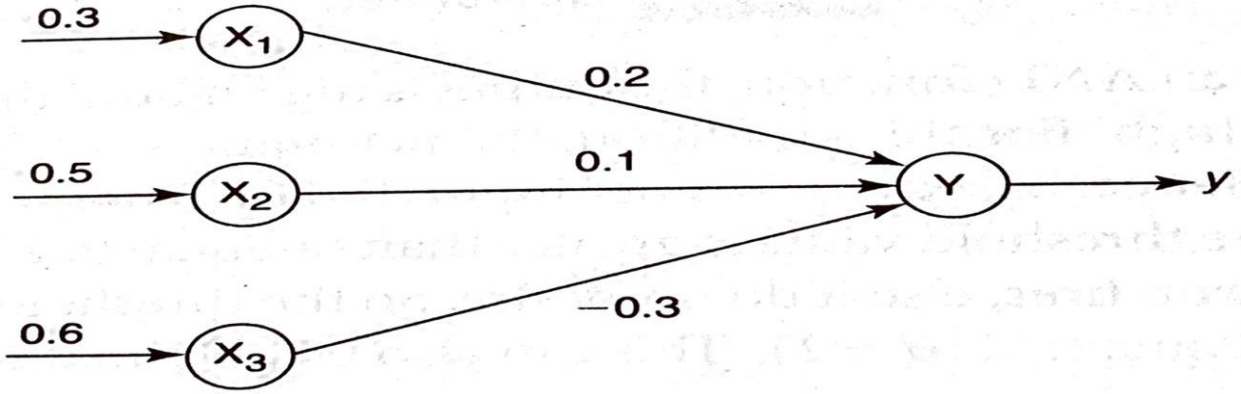
X: Input vector, $X = (x_1, \ldots, x_i, \ldots, x_n)$

Δw_{ij}: Change in weights given by $\Delta w_{ij} = w_{ij}(\text{new}) - w_{ij}(\text{old})$

α: Learning rate; it controls the amount of weight adjustment at each step of training.

ρ: Vigilance parameter, it controls the number of clusters in unsupervised networks.

Example1: Calculate the net Input to the output neuron



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

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

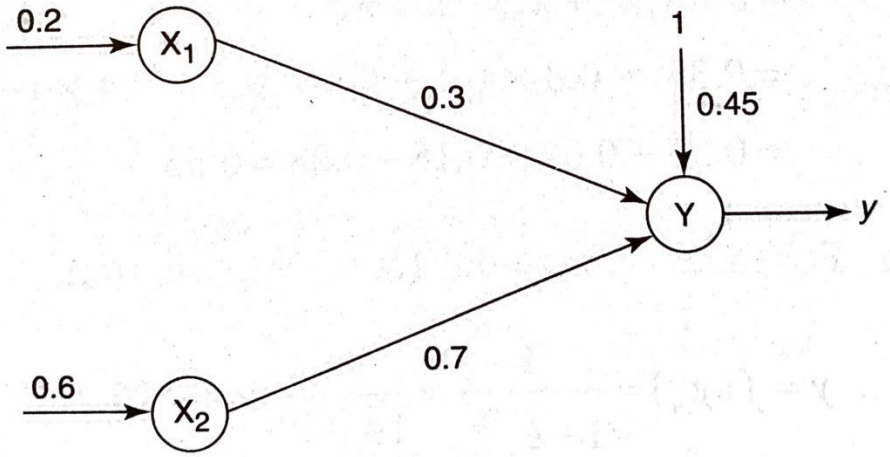
The net input can be calculated as

$$y_{in} = x_1 w_1 + x_2 w_2 + x_3 w_3$$

$$= 0.3 \times 0.2 + 0.5 \times 0.1 + 0.6 \times (-0.3)$$

$$= 0.06 + 0.05 - 0.18 = -0.07$$

Example2: Calculate the net Input to the output neuron



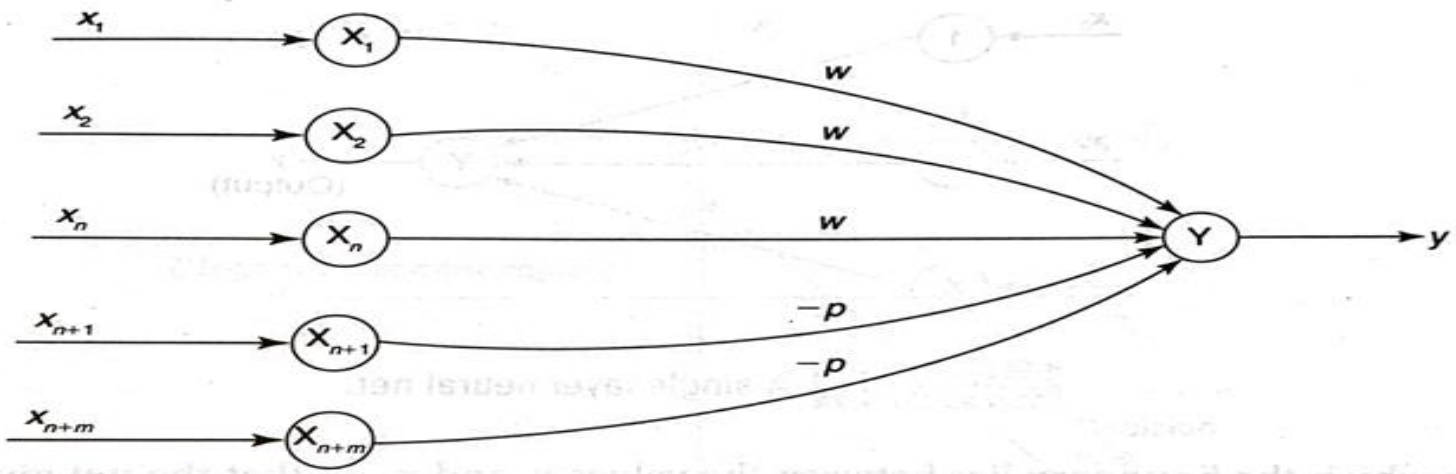
$[x_1, x_2] = [0.2, 0.6]$ and the weights are $[w_1, w_2] = [0.3, 0.7]$. Since the bias is included $b = 0.45$ and bias input x_0 is equal to 1, the net input is calculated as

$$\begin{aligned} y_{in} &= b + x_1 w_1 + x_2 w_2 \\ &= 0.45 + 0.2 \times 0.3 + 0.6 \times 0.7 \\ &= 0.45 + 0.06 + 0.42 = 0.93 \end{aligned}$$

1. McCulloch-Pitts Neuron Model

- ✓ Discovered in 1943 called as M-P neuron
- ✓ Activation of neuron is binary
- ✓ Threshold plays a major role
- ✓ The associated weights may be excitatory(+tive) and inhibitory
- ✓ excitatory with weight ($w>0$) and inhibitory with weight $-p(p<0)$
- ✓ Have same weights
- ✓ and will have a same weight for a particular neuron.
- ✓ Used in logic functions

McCulloch-Pitts Neuron Model



Firing of output neuron based upon the threshold, we need to define threshold *

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

Threshold should satisfy the following condition

$$\theta > nw - p$$

The output neuron will be fire if its receives K

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

Implementation of Mc Pitts neuron

- 2 input AND

1	1	1
1	0	0
0	1	0
0	0	0

- 2 input OR

1	1	1
1	0	1
0	1	1
0	0	0

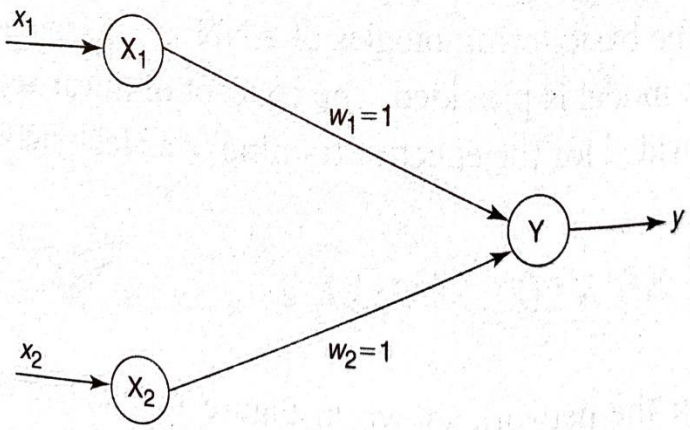
McCulloch-Pitts Neuron Model-AND

$$(1, 1),\ y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 1 + 1 \times 1 = 2$$

$$(1, 0),\ y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 1 + 0 \times 1 = 1$$

$$(0, 1),\ y_{in} = x_1 w_1 + x_2 w_2 = 0 \times 1 + 1 \times 1 = 1$$

$$(0, 0),\ y_{in} = x_1 w_1 + x_2 w_2 = 0 \times 1 + 0 \times 1 = 0$$



$$\theta \geq nw - p$$

Here, $n = 2$, $w = 1$ (excitatory weights) and $p = 0$ (no inhibitory weights). Substituting these values in the above-mentioned equation we get

$$\theta \geq 2 \times 1 - 0 \Rightarrow \theta \geq 2$$

Thus, the output of neuron Y can be written as

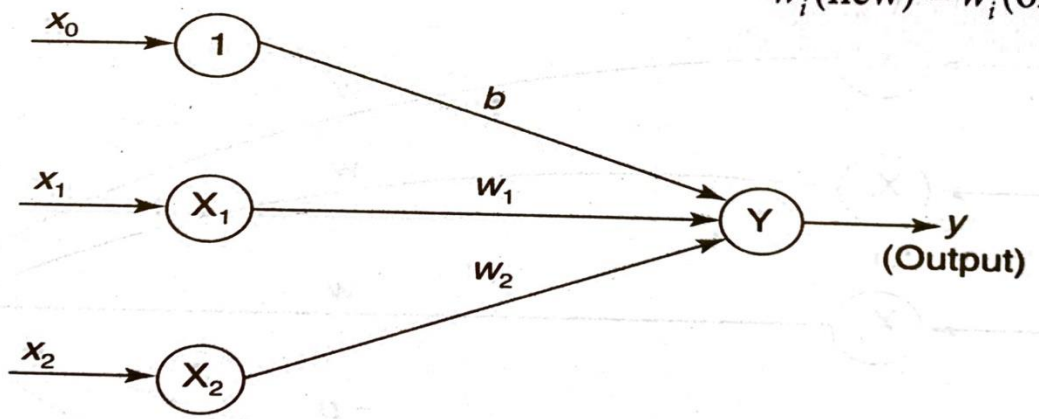
$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geq 2 \\ 0 & \text{if } y_{in} < 2 \end{cases}$$

where "2" represents the threshold value.

2. HEBB Neuron Model

1. Started in 1949, learning is based on change in the synaptic gap
2. According to the Hebb rule, the weight vector is proportional to the product of the input and the learning signal.
3. The learning signal is the neuron output
4. The Hebb rule is more suited for bipolar data then binary data.
5. The training algorithm is used for the calculation and adjustment of weights
6. The weight update in Hebb rule is given by

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



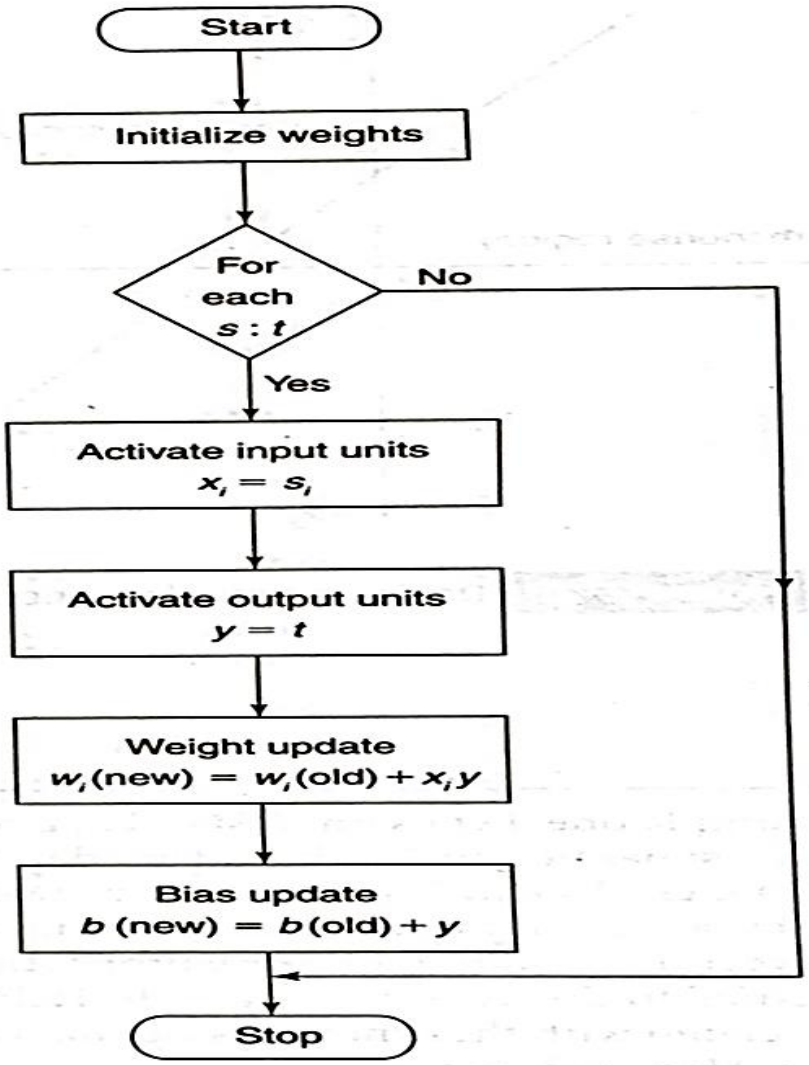
2. HEBB Neuron Model

The Hebb rule is more suited for bipolar data than binary data. If binary data is used, the above weight updation formula cannot distinguish two conditions namely:

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

Thus, there are limitations in Hebb rule application over binary data. Hence, the representation using bipolar data is advantageous.

HEBB Neuron Model



HEBB Training Algorithm

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

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

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

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

Step 4: Weight adjustments and bias adjustments are performed:

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

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

HEBB Training Algorithm

The above five steps complete the algorithmic process. In Step 4, the weight updation formula can also be given in vector form as

$$w(\text{new}) = w(\text{old}) + xy$$

Here the change in weight can be expressed as

$$\Delta w = xy$$

As a result,

$$w(\text{new}) = w(\text{old}) + \Delta w$$

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

HEBB Learning Problems

Design a Hebb net to implement logic AND function for bipolar input and target

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

HEBB Learning Problems

1. First Iteration initial weights

$$w_1 = w_2 = b = 0$$

First input $[x_1\ x_2\ b] = [1\,1\,1]$ and target = 1 [i.e., $y = 1$]: Setting the initial weights as old weights and applying the Hebb rule, we get

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

$$w_1(\text{new}) = w_1(\text{old}) + x_1 y = 0 + 1 \times 1 = 1$$

$$w_2(\text{new}) = w_2(\text{old}) + x_2 y = 0 + 1 \times 1 = 1$$

$$b(\text{new}) = b(\text{old}) + y = 0 + 1 = 1$$

The weights calculated above are the final weights that are obtained after presenting the first input. These weights are used as the initial weights when the second input pattern is presented. The weight change here is $\Delta w_i = x_i y$. Hence weight changes relating to the first input are

$$\Delta w_1 = x_1 y = 1 \times 1 = 1$$

$$\Delta w_2 = x_2 y = 1 \times 1 = 1$$

$$\Delta b = y = 1$$

HEBB Learning Problems

2. Second Iteration initial weights

$$\Delta w_1 = x_1 y = 1 \times 1 = 1$$
$$\Delta w_2 = x_2 y = 1 \times 1 = 1$$
$$\Delta b = y = 1$$

Second input $[x_1\ x_2\ b] = [1\ -1\ 1]$ and $y = -1$: The initial or old weights here are the final (new) weights obtained by presenting the first input pattern, i.e.,

$$[w_1\ w_2\ b] = [1\ 1\ 1]$$

HEBB Learning Problems

The weight change here is

$$\Delta w_1 = x_1 y = 1 \times -1 = -1$$
$$\Delta w_2 = x_2 y = -1 \times -1 = 1$$
$$\Delta b = y = -1$$

The new weights here are

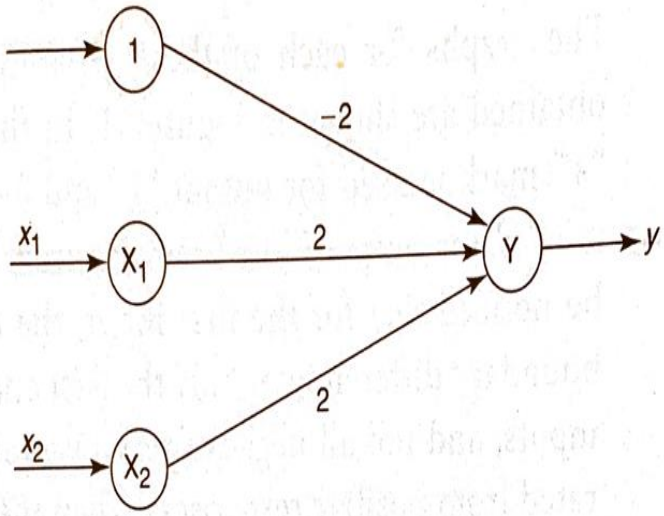
$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 1 - 1 = 0$$
$$w_2(\text{new}) = w_2(\text{old}) + \Delta w_2 = 1 + 1 = 2$$
$$b(\text{new}) = b(\text{old}) + \Delta b = 1 - 1 = 0$$

These weights will be the initial weights for the next iteration

HEBB Learning Problems

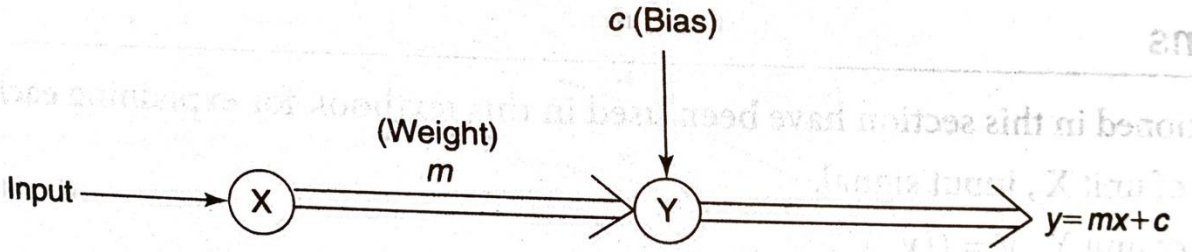
Similarly, by presenting the third and fourth input pattern, the new weights can be calculated. Below table the result of one complete epoch.

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



HEBB Learning Problems

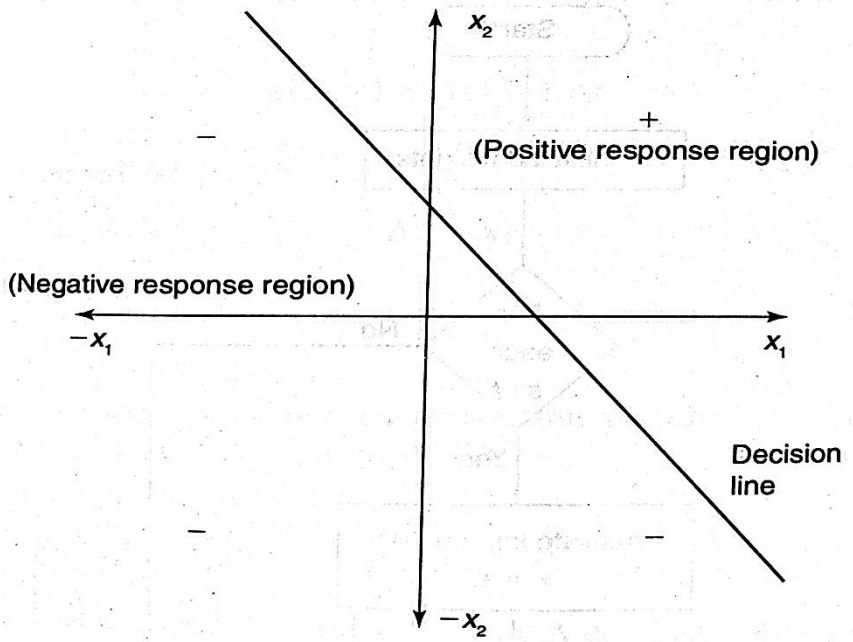
Decision boundary for AND function using Hebb rule



equation of straight line,

$$y = mx + c$$

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



HEBB Learning Problems

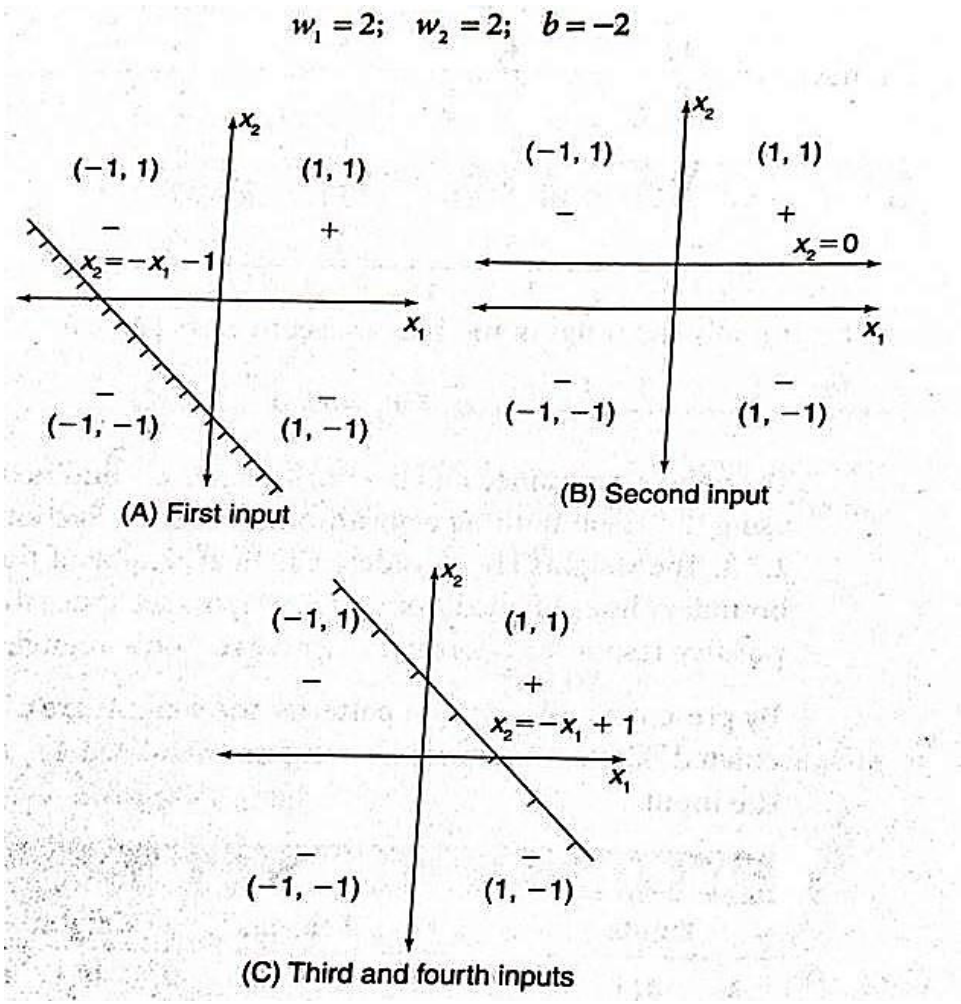
Decision boundary for AND function using Hebb rule for each pair

$$x_2 = \frac{-2}{2} x_1 + \frac{2}{2} \Rightarrow x_2 = -x_1 + 1$$

The graphs for each of these separating lines

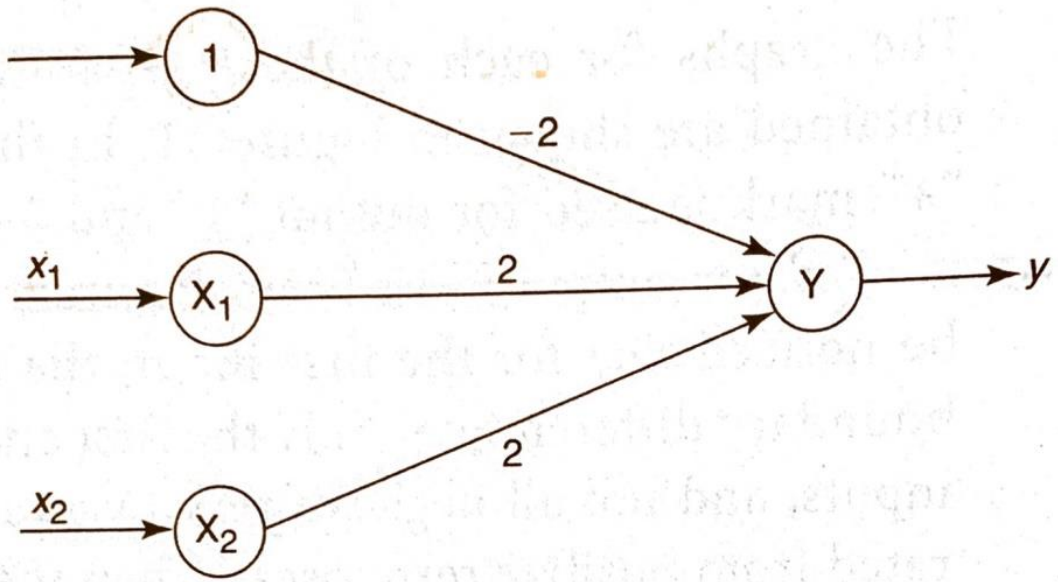
HEBB Learning Problems

Hebb net to implement logic AND function for bipolar input and target with calculated weight after one epochs



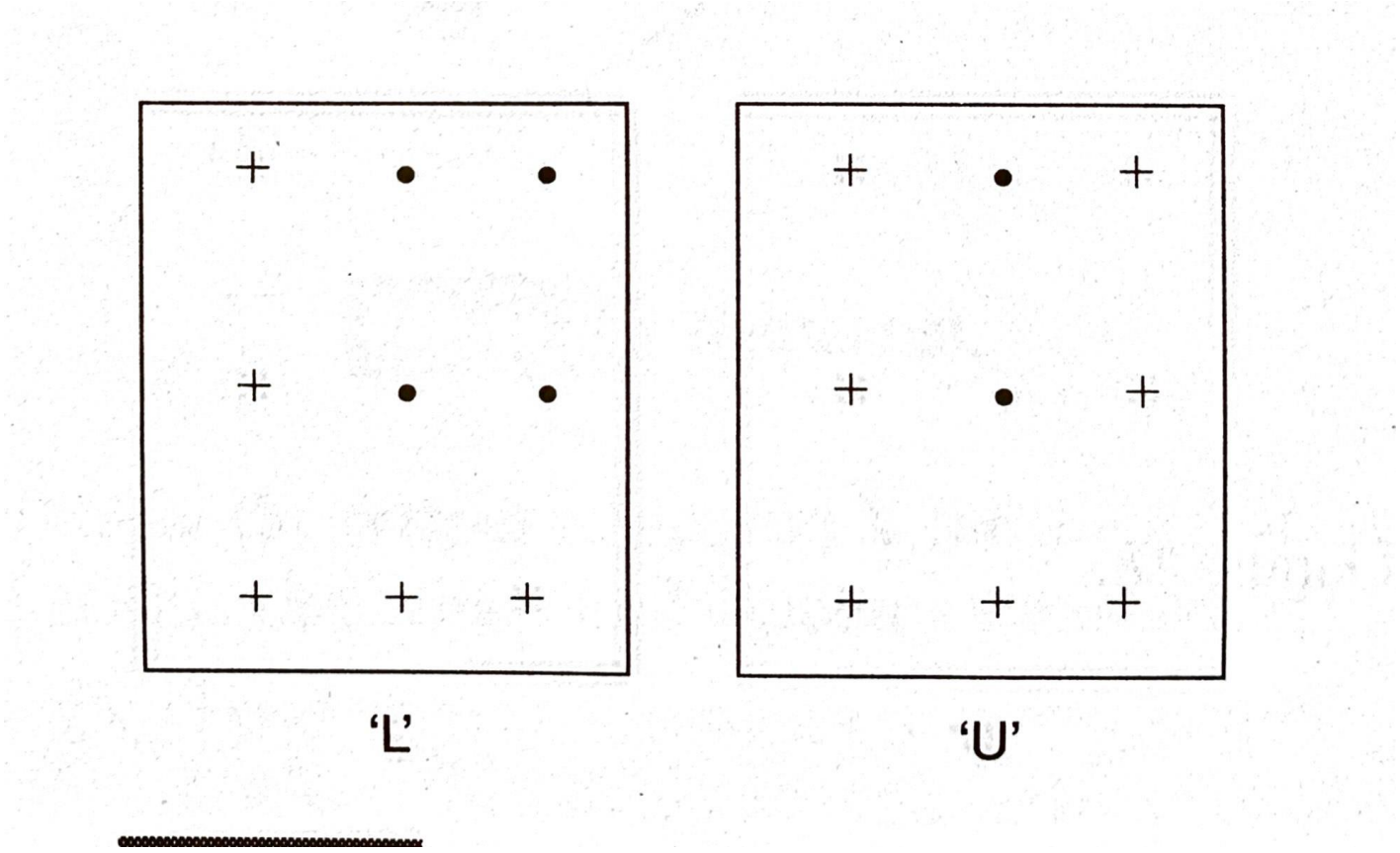
HEBB Learning Problems

Design a Hebb net to implement logic AND function for bipolar input and target



HEBB Learning Problems-2

Find the weights required to perform the following classifications of the input patterns using Hebbs rule



HEBB Learning Problems-2

Pattern	Inputs										Target
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	b	y
L	1	-1	-1	1	-1	-1	1	1	1	1	1
U	1	-1	1	1	-1	1	1	1	1	1	-1

HEBB Learning Problems-2

A single-layer network with nine input neurons, one bias and one output neuron is formed. Set the initial weights and bias to zero, i.e.,

$$w_1 = w_2 = w_3 = w_4 = w_5$$
$$= w_6 = w_7 = w_8 = w_9 = b = 0$$

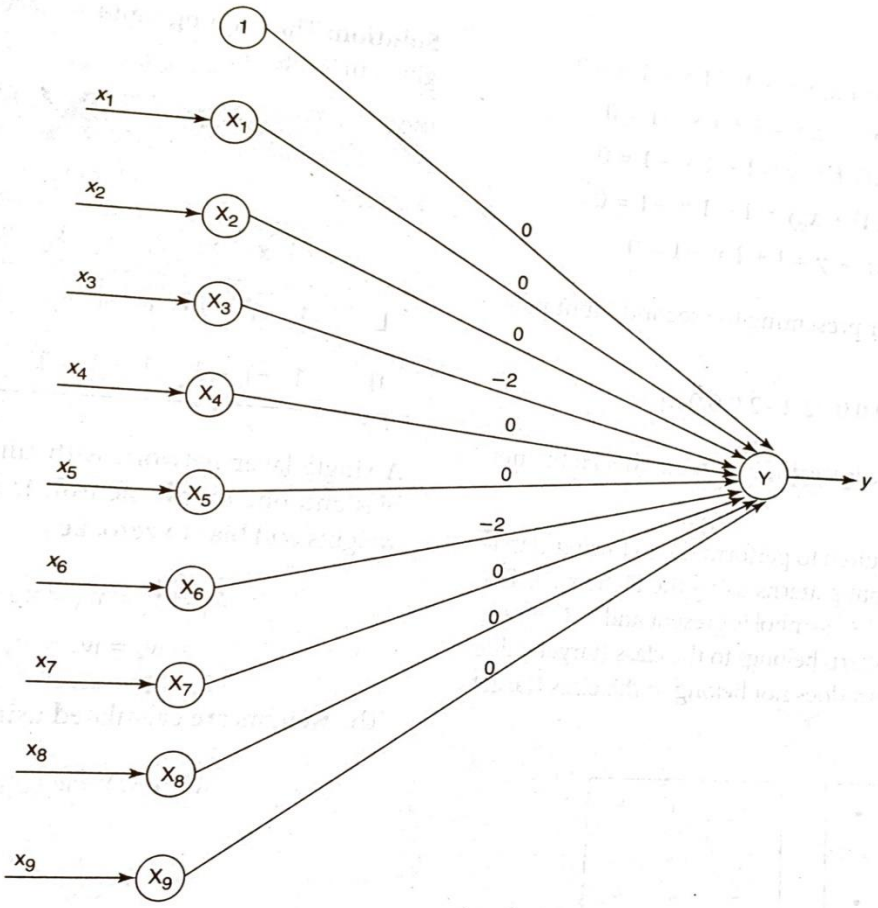
The weights are calculated using

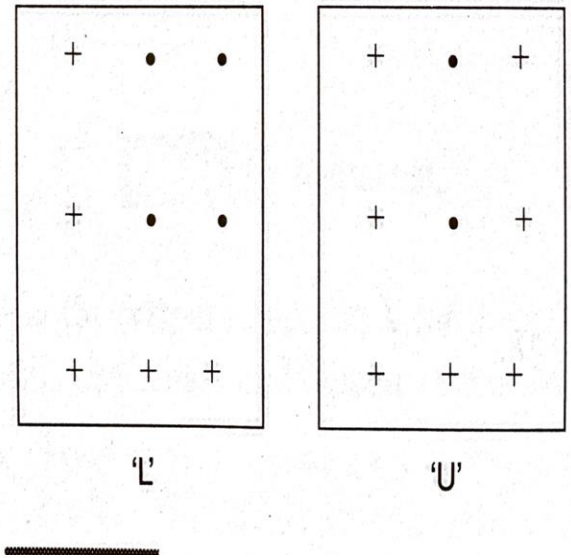
$$w_i(\text{new}) = w_i(\text{old}) + x_i y$$

HEBB Learning Problems-2

Inputs										Target	Weights									
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	b	y	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	b
											(0	0	0	0	0	0	0	0	0	0)
1	-1	-1	1	-1	-1	1	1	1	1	1	1	-1	-1	1	-1	-1	1	1	1	1
1	-1	1	1	-1	1	1	1	1	1	-1	0	0	-2	0	0	-2	0	0	0	0

HEBB Learning Problems-2





Try for following logic function

NOT

x	x'
0	1
1	0

AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

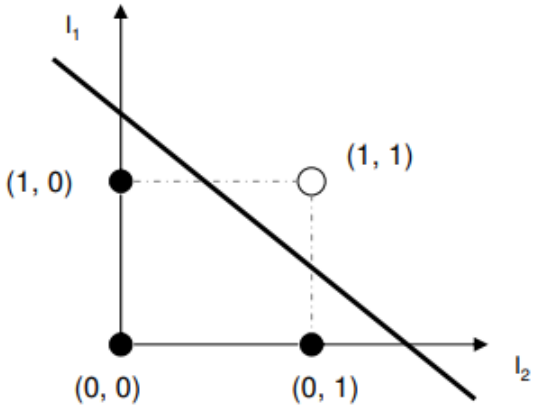
x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

XOR

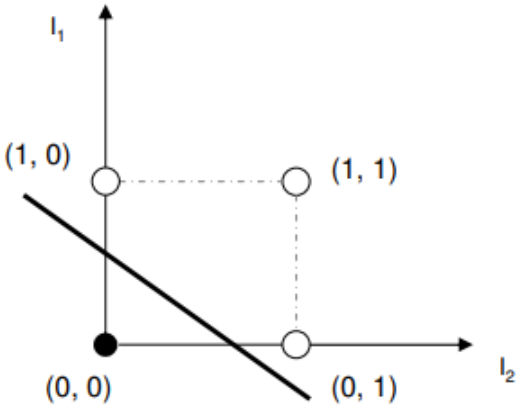
x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Concept of Linear and Nonlinear separability in ANN

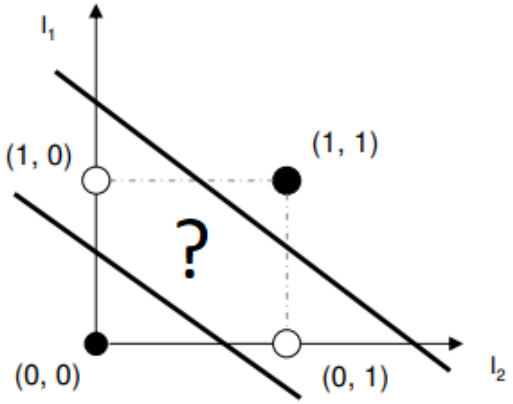
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0



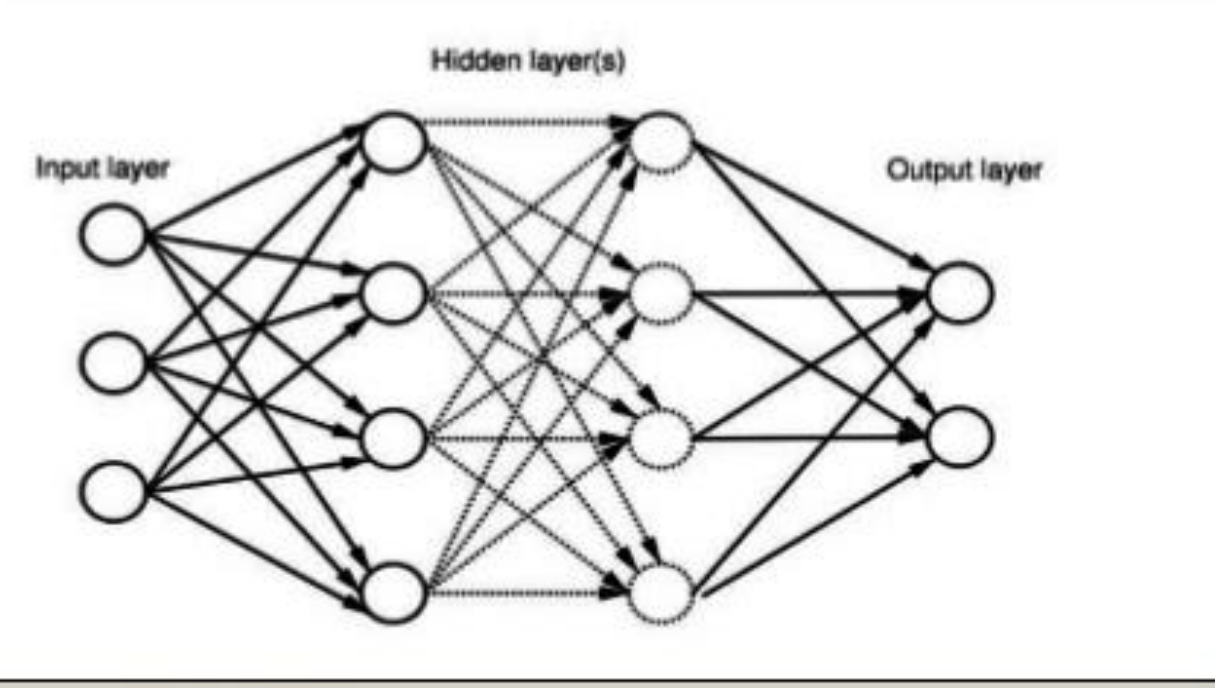
Concept of Linear and Nonlinear separability in ANN

- Non Linearly separable functions: XOR gate implementation
  - MLP data transformation
  - mapping implementation
  - graphical solution

Sourse: https://www.slideshare.net/Randa_Elanwar/lecture-6-44124142

Concept of Linear and Nonlinear separability in ANN

To implement the nonlinearity we need to insert one or more extra layer of nodes between the input layer and the output layer that is Hidden layer



Concept of Linear and Nonlinear separability in ANN

MLP data transformation and mapping implementation

- **Need for hidden units:**
- If there is one layer of enough hidden units, the input can be recoded (memorized) → multilayer perceptron (MLP)
- This recoding allows any problem to be mapped/represented (e.g., x^2, x^3, etc.)
- Question: how can the weights of the hidden units be trained?
- Answer: Learning algorithms e.g., back propagation
- The word 'Back propagation' is meant to the error propagation for weight adaptation of layers beginning from the last hidden layer back to the first i.e. weights of last layer are computed before the previous layers

Concept of Linear and Nonlinear separability in ANN

- Back propagation tries to transform training patterns to make them almost linearly separable and use linear network

- In other words, if we need more than 1 straight line to separate +ve and –ve patterns, we solve the problem in two phases:
  - *In phase 1:* we first represent each straight line with a single perceptron and classify/map the training patterns (output)
  - *In phase 2:* these outputs are transformed to new patterns which are now linearly separable and can be classified by an additional perceptron giving the final result.

Concept of Linear and Nonlinear separability in ANN

Multi-layer Networks and Perceptrons

- Have one or more layers of hidden units.

- With two possibly very large hidden layers, it is possible to implement any function.

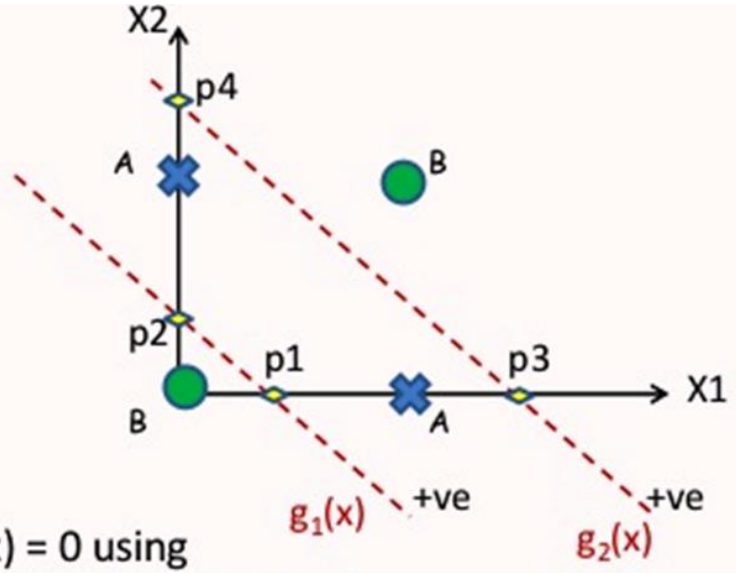
- Networks without hidden layer are called perceptrons.

- Perceptrons are very limited in what they can represent, but this makes their learning problem much simpler.

Concept of Linear and Nonlinear separability in ANN

- **Example:** XOR problem

x1	x2	XOR	Class
0	0	0	B
0	1	1	A
1	0	1	A
1	1	0	B



- **Phase 1:** we draw arbitrary lines
- We find the line equations $g_1(x) = 0$ and $g_2(x) = 0$ using arbitrary intersections on the axes (*yellow points p1, p2, p3, p4*).
- We assume the +ve and –ve directions for each line.
- We classify the given patterns as +ve/-ve with respect to both $g_1(x)$ & $g_2(x)$
- **Phase 2:** we transform the patterns we have
- Let the patterns that are +ve/-ve with respect to both $g_1(x)$ and $g_2(x)$ belong to class B (similar signs), otherwise belong to class A (different signs).
- We find the line equations $g(y) = 0$ using arbitrary intersections on the new axes

Concept of Linear and Nonlinear separability in ANN

- Let p1 = (0.5,0), p2 = (0,0.5), p3(1.5,0), p4(0,1.5)
- Constructing $g_1(x) = 0$

$$\frac{x2 - p1(2)}{x1 - p1(1)} = \frac{p2(2) - p1(2)}{p2(1) - p1(1)}$$

$$\frac{x2 - 0}{x1 - 0.5} = \frac{0.5 - 0}{0 - 0.5}$$

- $g_1(x) = x1 + x2 - 0.5 = 0$
- Constructing $g_2(x) = 0$

$$\frac{x2 - p3(2)}{x1 - p3(1)} = \frac{p4(2) - p3(2)}{p4(1) - p3(1)}$$

$$\frac{x2 - 0}{x1 - 1.5} = \frac{1.5 - 0}{0 - 1.5}$$

- $g_2(x) = x1 + x2 - 1.5 = 0$

Concept of Linear and Nonlinear separability in ANN

- Assume x1>p1(1) is the positive direction for $g_1(x)$
- Assume x1>p3(1) is the positive direction for $g_2(x)$
- Classifying the given patterns with respect to $g_1(x)$ and $g_2(x)$:

x1	x2	$g_1(x)$	$g_2(x)$	y1	y2	Class
0	0	-ve	-ve	0	0	B
0	1	+ve	-ve	1	0	A
1	0	+ve	-ve	1	0	A
1	1	+ve	+ve	1	1	B

- If we represent +ve and –ve values as a result of step function
  i.e., y1 =f($g_1(x)$)= 0 if $g_1(x)$ is –ve and 1 otherwise
  and y2 =f($g_2(x)$)= 0 if $g_2(x)$ is –ve and 1 otherwise
- We now have only three patterns that can be linearly separable and we got rid of the extra pattern causing the problem (since 2 patterns coincide)

Concept of Linear and Nonlinear separability in ANN

- Let p1 = (0.5,0), p2 = (0,-0.25)
- Constructing g(y) = 0

$$\frac{y2 - p1(2)}{y1 - p1(1)} = \frac{p2(2) - p1(2)}{p2(1) - p1(1)}$$

$$\frac{y2 - 0}{y1 - 0.5} = \frac{-0.25 - 0}{0 - 0.5}$$

- g(y) = y1 – 2 y2 – 0.5 = 0