SINGLE LAYER NEURAL NETWORK

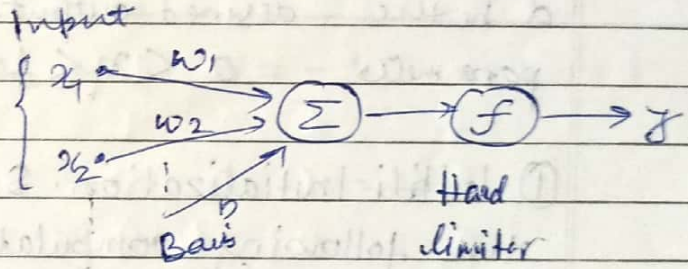
DATE: / /

Perceptron's, convergence theorem, Linear Separability, Bayes theorem, & Relationship b/w Perceptron & Bayes classifier, ADALINE,

McCulloch and Pitts (1943) for introducing the idea of neural networks as computing m/c.
Hebb (1949) for postulating the first rule for self-organized learning.

⇒ PERCEPTRON :- Rosenblatt (1958)

It is based one McCulloch-Pitts model of the neuron with hard limitation activation ~~fn~~ function.
The perceptron has a single output whose values determine to which of the two classes each input patterns belongs.
The neuron produces an output equal to +1 if the hard limiter input is positive, and -1 if it is negative.



$$y_i = f(a) = \sum_{i=} w_i x_i + b$$

The goal of the the perceptron is to correctly classify the set of input into one of two classes y_1 or y_2.

The linear equation

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

defines the decision boundary (a hyperplane in the n-dimensional input space) that ~~R~~ Halves the space.

Be Positive.....

DATE: / /

Rosenblatt developed a learning procedure to determine the weight & threshold in a perceptron, given a set of training patterns.

Rosenblatt proved that when training patterns are drawn from two linearly separable classes, the perceptron learning procedure converges after a finite no. of iterations.

This is perceptron convergence theorem.

A single layer perceptron can only separate linearly separable patterns as long as a monotonic activation function is used.

Perceptron learning Convergence Theorem:-

Let $x(n)$ be the input vectors & $w(n)$ be the weight vectors, b is bais, y be the actual response d be the desired output & η be the learning rate parameter $0 < \eta \leq 1$.

(1) Initialization: Set $w(0) = 0$. Then perform the following computations for time step $n = 1, 2, \ldots$

(2) Activation: At time step n, activate the perceptron by applying continuous valued input vector $x(n)$ & desired response $d(n)$.

(3) Computation of Actual response of perceptron

$$y(n) = \text{sgn}[w^T(n)\, x(n)]$$

Signum function.

(4) Adaptation of Weight Vector: update the weight vector of perceptron:

$$w(n+1) = w_{ij}(n) + \eta[d(n) - y(n)]\, x(n)$$

$$b(n+1) = b(n) + \eta[d(n) - y(n)]$$

Be Positive.....

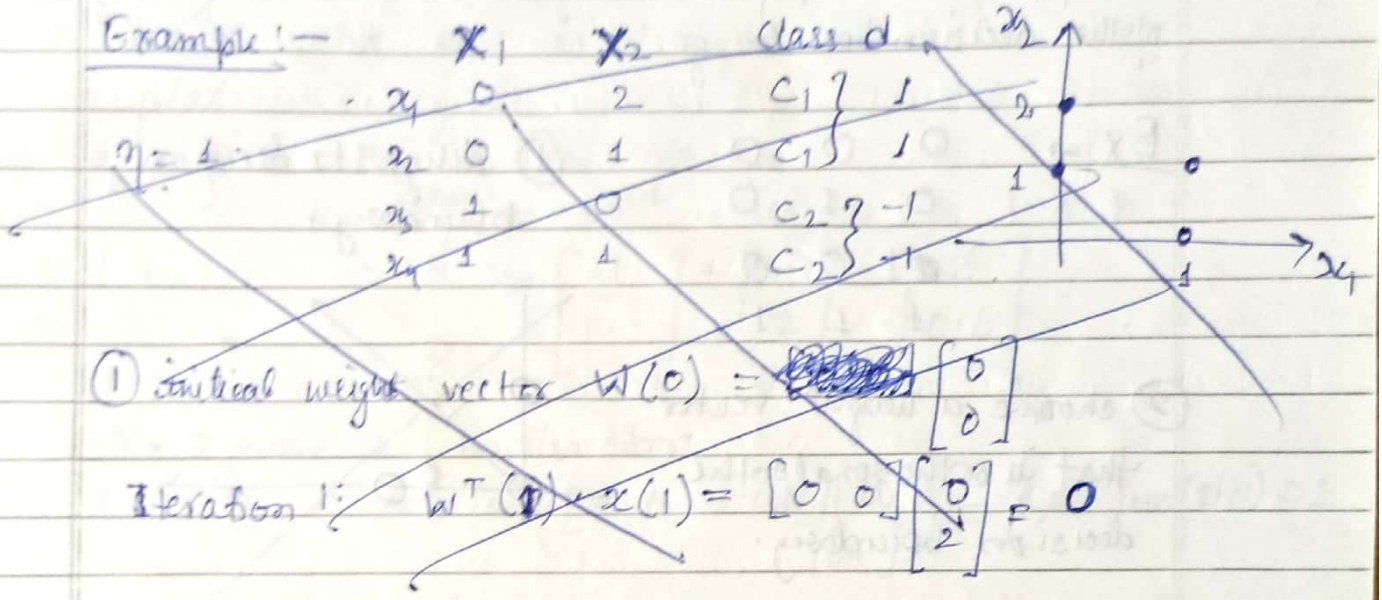
DATE: / /

Where, $d(n) = \begin{cases} +1 & \text{if } x(n) \text{ belongs to class I} \\ -1 & \text{if } x(n) \text{ belongs to class II} \end{cases}$

(5) Continuation: Increment time step n by one & go back to (2).

Example:-

	x_1	x_2	class d
x_1	0	2	C_1 } 1
x_2	0	1	C_1 } 1
x_3	1	0	C_2 } -1
x_4	1	1	C_2 } -1



(1) initial weight vector $w(0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Iteration 1: $w^T(1)\, x(1) = \begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = 0$

Ex → $w_{11} = 1, \quad w_{12} = 1, \quad b = -1$

$$a = w_i^T x_i + b$$

$$1 \cdot x_1 + 1 \cdot x_2 + b = 0$$

$$x_1 + x_2 - 1 = 0$$

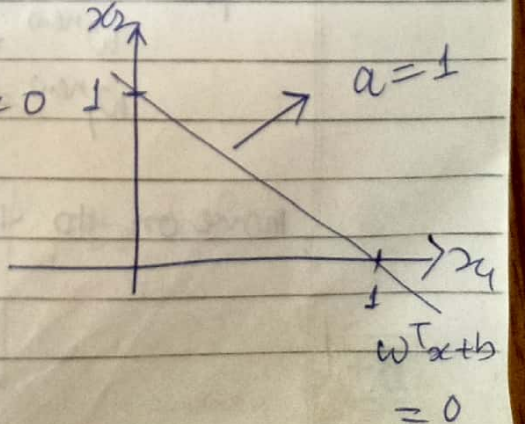
To draw the line, we can find the points, where it intersects the x_1 & x_2

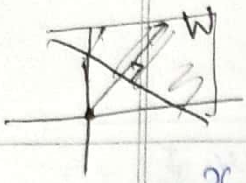
if $x_1 = 0$, $\Rightarrow \quad 0 + x_2 - 1 = 0$

$$x_2 = 1$$

if $x_2 = 0$ $\Rightarrow \quad x_1 + 0 - 1 = 0$

$$x_1 = 1$$





DATE: / /

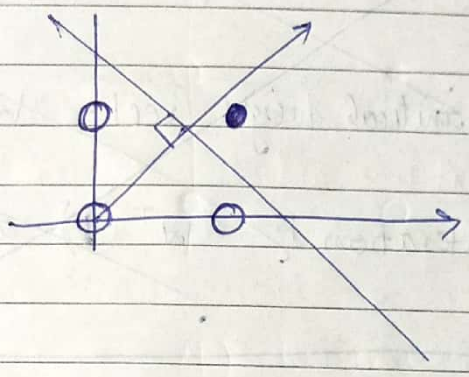
$x = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$, $a = hardlim(w^T x + b)$

$= hardlim\left(\begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} 2 \\ 0 \end{bmatrix} - 1\right) = 1$

output will be 1, for the region above and to the right of the decision boundary.

Ex → 0 0 0
0 1 0
1 0 0
1 1 1

① design the decision boundary.



② choose a weight vector that is orthogonal to the decision boundary.

Example:-

			b
p_1 →	1	2	1
p_2 →	−1	2	0
p_3 →	0	−1	0

$w^* = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix}$

start with x_4

$a = hardlim(w_1^T x_1) = hardlim\left(\begin{bmatrix} 1.0 & -0.8 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix}\right)$

$= hardlim(-0.6) = 0$

it's output is not correct.

update the weight vector.

$w^{new} = w^{old} + x$

$w_1^{new} = w_1^{old} + x_4 = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 1.2 \end{bmatrix}$

move on to the next input vector.

DATE: / /

$$a = hardlim(w_1^T x_2) = hardlim\left(\begin{bmatrix}2 & 1.2\end{bmatrix}\begin{bmatrix}-1\\2\end{bmatrix}\right)$$

$$= hardlim(0.4) = 1$$

target associated with x_2 is 0 & the output a is 1. A class 0 vector was misclassified as a 1.

update the weight

if $t = 0$ and $a = 1$ then

$$w_1^{new} = w_1^{old} - x_2$$

$$= \begin{bmatrix}2\\1.2\end{bmatrix} - \begin{bmatrix}-1\\2\end{bmatrix} = \begin{bmatrix}3\\-0.8\end{bmatrix}$$

Now move to next vector

$$a = hardlim\left(\begin{bmatrix}3.0 & -0.8\end{bmatrix}\begin{bmatrix}0\\-1\end{bmatrix}\right) = hardlim(0.8) = 1$$

It is also misclassifies x_3

update weight

$$w_1^{new} = w_1^{old} - p x_3 = \begin{bmatrix}3.0\\-0.8\end{bmatrix} - \begin{bmatrix}0\\-1\end{bmatrix} = \begin{bmatrix}3.0\\0.2\end{bmatrix}$$

continue with iterations you will find that both i/p values will now be correctly classified.

perceptron error $e = target - actual = t - a$

we can update bias

$b^{new} = b^{old} + e$ The algo has converged to a solution.

Ex → $|a| = \begin{bmatrix}0.5\\-1\\0.5\end{bmatrix}$, $b = 0.5$

$$x_1 = \begin{bmatrix}+1\\-1\\-1\end{bmatrix}, t_1 = [0], p_2 = \begin{bmatrix}1\\1\\-1\end{bmatrix}, t_2 = [1]$$

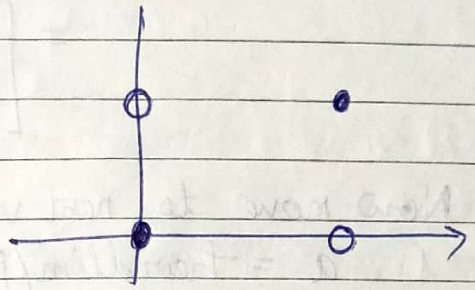
Be Positive.....

DATE: / /

Limitation :- The perceptron learning rule is guaranteed to converge to a solution in a finite number of steps, so long as a solution exists.

The perceptron can be used to classify input vectors that can be separated by a linear boundary. We call such vectors linearly separable.
Unfortunately, many problems are not linearly separable.
e.g. XOR gate problem.

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



Ex → $\left\{p_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, t_1 = 0\right\}$

$\left\{p_2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, t_2 = 1\right\}$

$\left\{p_3 = \begin{bmatrix} -2 \\ 2 \end{bmatrix}, t_3 = 0\right\}$

$\left\{p_4 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, t_4 = 1\right\}$

(1) Initial weights & bias.

$W(0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$b(0) = 0$

(2) Calculating perceptron output a for first input vector p_1,

(3) $W = \begin{bmatrix} -2 \\ -3 \end{bmatrix}, b = 1$

(4) Draw Decision Boundary.

Be Positive.....

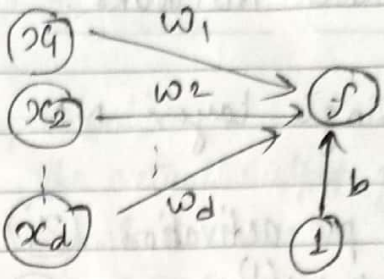
⇒ BIOLOGICAL MOTIVATION AND CONNECTION

The basic computational unit of the brain is neuron.
The neurons are connected with approximately $10^{14} - 10^{15}$ synapses.
Each neuron receives inputs signals from its dendrites and produces output signals along its (single) ~~neuron~~ axon.
The axon eventually branches out and connects via synapses to dendrites of other neurons.
- In the computational model of a neuron, the signals that travel along the axons (eg. x_1) interact multiplicatively (eg. $w_1 x_1$) with the dendrites of other neurons based on the synaptic strength at that synapse (e.g. w_1)

ॐ गणेशाय नमः

DATE: __/__/____

ARTIFICIAL NEURON:



Neuron pre-activation (or input activation):

$$a(x) = b + \sum_i w_i x_i$$
$$= b + w^T x$$

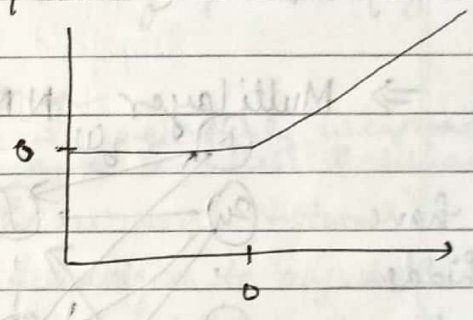
$W \rightarrow$ connection weights
$b \rightarrow$ bias
$g(\cdot) \rightarrow$ activation function

Neuron (output) activation
$$h(x) = g(a(x)) = g\left(b + \sum_i w_i x_i\right)$$

ACTIVATION FUNCTION:

→ Rectified linear activation function

$$g(a) = \text{rectlin}(a)$$
$$= \max(0, a)$$



bounded below by 0 (always non-negative)
Not upper bound
strictly increasing
tends to give neurons with sparse activities

→ Linear activation function
$$g(a) = a$$

→ Sigmoid activation function
$$g(a) = \text{sigm}(a) = \frac{1}{1+e^{-a}}$$

always positive, bounded (0, 1)
strictly increasing

→ hyperbolic tangent
$$g(a) = \tanh(a) = \frac{e^{2a} - 1}{e^{2a} + 1}$$

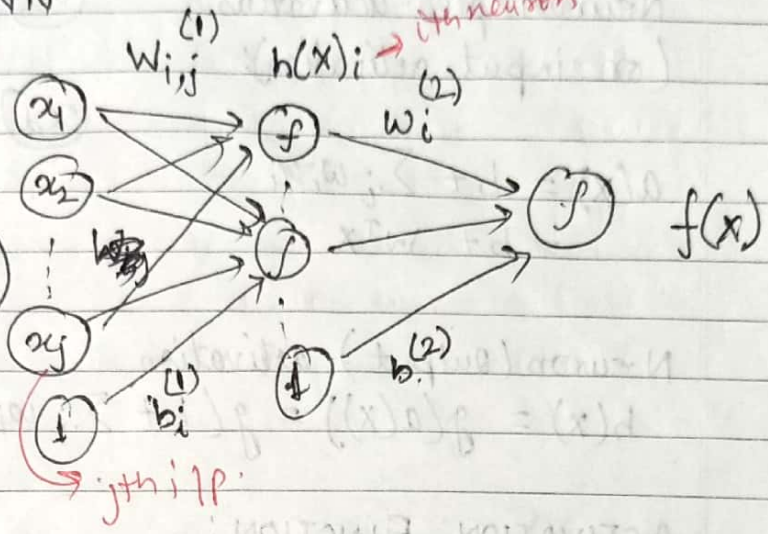
Can be positive or negative
Bounded, strictly increasing

Be Positive.....

DATE: __/__/____

NEURAL NETWORK :-

⇒ Single layer NN



Hidden layer pre-activation

$a(x) = b^{(1)} + W^{(1)} x$

$\left(a(x)_i = b_i^{(1)} + \sum_j W_{i,j}^{(1)} x_j\right)$

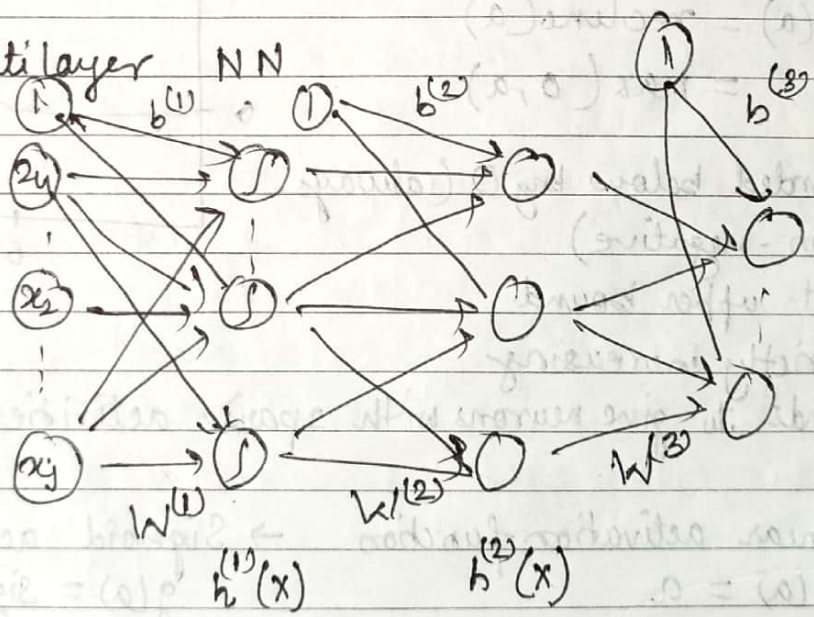
Hidden layer activation

$h(x) = g(a(x))$

Output layer activation:

$f(x) = o(b^{(2)} + W^{(2)T} h^{(1)} x)$

⇒ Multi layer NN



Could have L hidden layers:

Layer preactivation for $k > 0$

$(h^{(0)}(x) = x)$

$a^{(k)}(x) = b^{(k)} + W^{(k)} h^{(k-1)}(x)$

Hidden layer activation (k from 1 to L):

$h^{(k)}(x) = g(a^{(k)}(x))$

Output layer activation $(k = L+1)$:

$h^{(L+1)}(x) = o(a^{(L+1)}(x)) = f(x)$

Be Positive.....