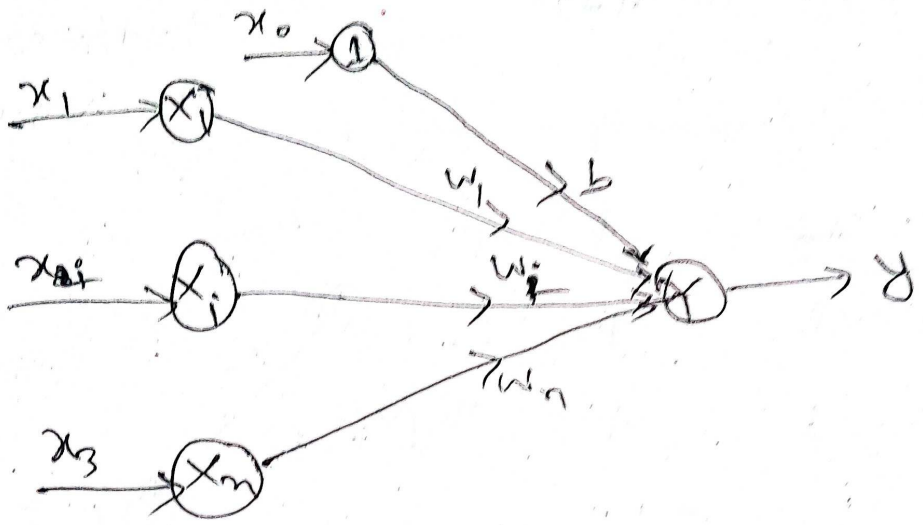
(1) Implement AND function using perceptron networks for bipolar inputs and targets



Perceptron learning rule

In case of the perceptron learning rule, the learning signal is the difference between the calculated output and actual (target) output of a neuron.

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

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

weights are updated using the formula

If $y \neq t$, then

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

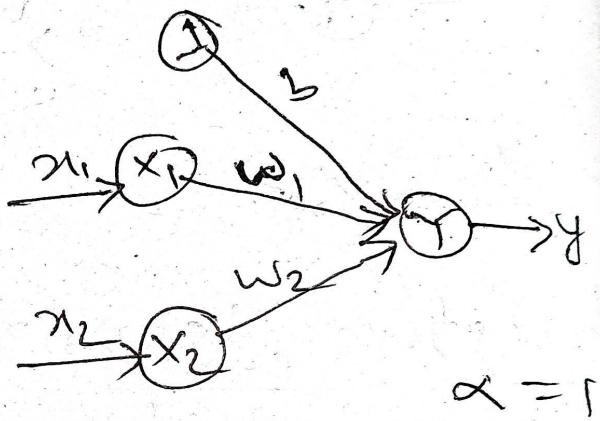
else,

$w(new) = w(old)$

t = target

x = input

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



$\alpha = 1$

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

$\Delta w_1 = \alpha t x_1$

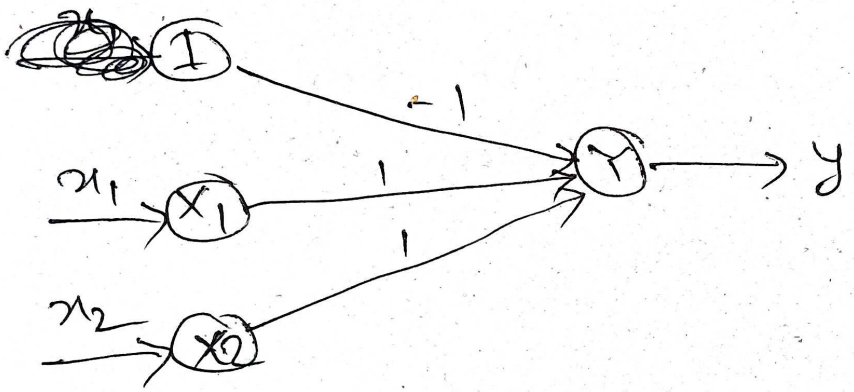
$\Delta w_2 = \alpha t x_2$

$\Delta b = \alpha t$

Input		target	Net input	Calculated output	weight changes			weights		
x_1	x_2	(t)	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1 (0)	w_2 (0)	b (0)
Epoch-1										
1	1	1	0	0	1	1	1	1	1	1
1	-1	-1	1	1	-1	1	-1	0	2	0
-1	1	-1	2	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	0	0	0	1	1	-1
1	-1	-1	-1	-1	0	0	0	1	1	-1
-1	1	-1	-1	-1	0	0	0	1	1	-1
-1	-1	-1	-3	-1	0	0	0	1	1	-1



Final rates of the particular network are 1 1 -1

Q Implement XOR function using McCulloch Pitts Neuron

x_1	x_2	Y
1	1	0
1	0	1
0	1	1
0	0	0

XOR function cannot be represented by simple and single logic functions. It is represented as,

$$Y = \overline{x_1}\, x_2 + x_1 \overline{x_2}$$

Now, we have to add the weights like:

$$y_{in} = \overline{x_1}\, x_2 w_1 + x_1 \overline{x_2}\, w_2$$

Assume, $w_1 = w_2 = 1$

x_1	w_1	x_2	w_2	y_{in}	Y
1	1	1	1	0	0
1	1	0	1	1	1
0	1	1	1	1	1
0	1	0	1	0	0

$y_{in} = \overline{x_1}\, x_2 w_1 + x_1 \overline{x_2} * w_2$

$y_{in} = 0 * 1 * 1 + 1 * 0 * 1 = 0$

$y_{in} = 0 * 0 * 1 + 1 * 1 * 1 = 1$

Calculating Threshold value.

We can simply say,

$\theta \geq 1$

XOR function using MP Neuron

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

(3) SGD method:

Stochastic Gradient Descent (training_examples, η)

Each training example is a pair of the form $(\vec{x}, t)$, where $\vec{x}$ is the vector of input values and t is the target output value; η is the learning rate (e.g. 0.05)

* Initialize each w_i to some small random value

* Until the termination condition is met, Do

~~- Initialize each Δw_i to zero.~~

- For each $(\vec{x}, t)$ in training example, Do
  * Input the instance $\vec{x}$ to the unit and compute the output o
  * For each linear unit weight w_i, Do

$$w_i := w_i + \eta(t-o)x_i$$

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(t_d - o_d)x_{id}$$

Gradient Descent	Stochastic Gradient descent
Error is summed over all examples before updating weights.	weights are updated examining each training example
summing over multiple examples require more computation per weight update step	Less computation as individual weights are updated

confusion matrix

TP	FP
FN	TN

⑨ SVM algorithm

Classification
Regression

Uses:
Face detection, image classification, text categorization.

Types:
① Linear ② non-linear

$z = x^2 + y^2$

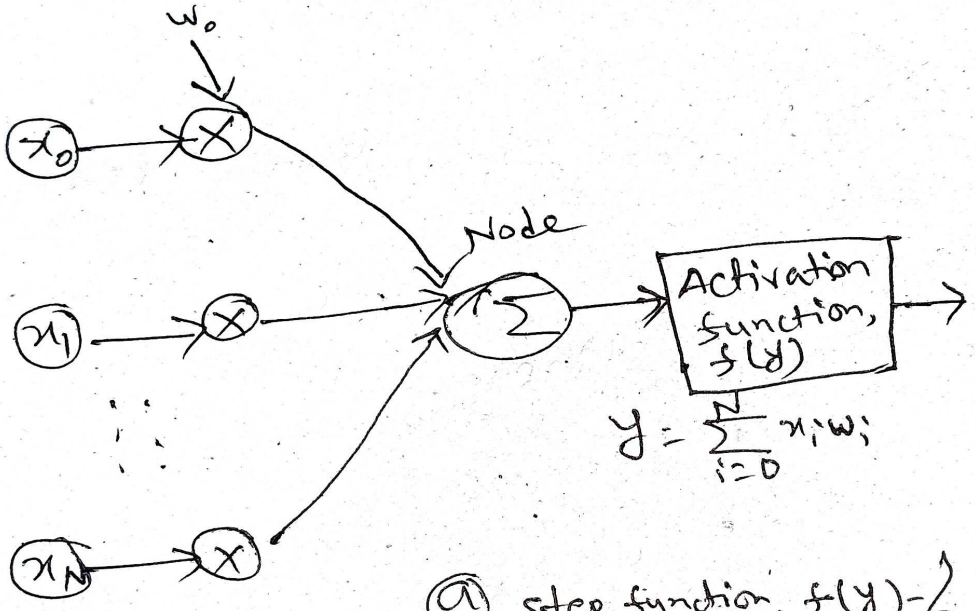
SVM:

Advantages:
① It is highly accurate
② It can handle many features

Disadvantages:
① Its speed is low
② Its requires more time to process.

(7) ANN

Dendrite → Input
Soma → Node
Axon → output
synapse → weighting factor



(a) step function, $f(y) = \begin{cases} 1, & \text{if } y \geq 0 \\ 0, & \text{if } y \leq 0 \end{cases}$

(b) signum function,

$$f(y) = \begin{cases} +1, & \text{if } y > \tau \\ -1, & \text{if } y \leq \tau \end{cases}$$

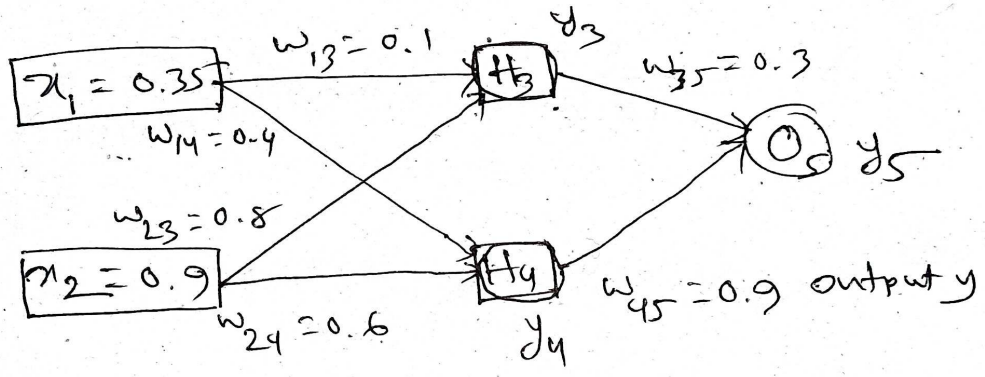
(c) sigmoid function,

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

where α is the slope parameter

Back propagation Example

Assume that the neurons have a sigmoid activation function, perform a forward pass, and a backward pass on the network. Assume that the actual output of y is 0.5 and learning rate is 1. perform another forward pass.



Forward pass: compute output for y_3, y_4 and y_5

$$a_j = \sum_i (w_{i,j} * x_i) \qquad y_j = F(a_j) = \frac{1}{1+e^{-a_j}}$$

$$a_1 = (w_{13} * x_1) + (w_{23} * x_2)$$

$$= (0.1 * 0.35) + (0.8 * 0.9) = 0.755$$

$$y_3 = f(a_1) = \frac{1}{1+e^{-0.755}} = 0.68$$

P.

$$a_2 = (w_{14} * x_1) + (w_{24} * x_2)$$
$$= (0.4 * 0.35) + (0.6 * 0.9) = 0.68$$
$$y_4 = f(a_2) = \frac{1}{1 + e^{-0.68}} = 0.6637$$

$$a_3 = (w_{35} * y_3) + (w_{45} * y_4)$$
$$= (0.3 * 0.68) + (0.9 * 0.6637) = 0.801$$
$$y_5 = f(a_3) = \frac{1}{1 + e^{-0.801}} = 0.69 \text{ (Network output)}$$

$$\text{Error} = y_{target} - y_5 = -0.19$$

* Each weight changed by:

$$\Delta w_{ji} = \eta \delta_j o_i$$

$$\delta_j = o_j (1 - o_j)(t_j - o_j) \quad \text{if } j \text{ is an output unit}$$

$$\delta_j = o_j (1 - o_j) \sum_k \delta_k w_{kj} \quad \text{if } j \text{ is a hidden unit}$$

* where η is a constant called the learning rate
* t_j is the correct teacher output for unit j
* δ_j is the error measure for unit j

Backward pass: compute δ_3, δ_4 and δ_5

For output unit:

$$\delta_5 = y(1-y)(y_{target} - y)$$

$$= 0.69(1-0.69)(0.5-0.69) = -0.0406$$

For hidden unit:

$$\delta_3 = y_3(1-y_3)w_{35} * \delta_5$$

$$= 0.68*(1-0.68)*(0.3)*(-0.0406) = -0.00265$$

$$\delta_4 = y_4(1-y_4)w_{45} * \delta_5$$

$$= 0.6637*(1-0.6637)*(0.9)*(-0.0406)$$

$$= -0.0082$$

$$\Delta w_{ij} = \eta \delta_j o_i$$

$$\Delta w_{45} = \eta \delta_5 y_4 = 1*(-0.0406)*0.6637$$

$$= -0.0269$$

$$w_{45}(new) = \Delta w_{45} + w_{45}(old)$$

$$= -0.0269 + 0.9 = 0.8731$$

$$\Delta w_{14} = \eta \delta_4 x_1 = 1*(-0.0082)*0.35 = -0.00287$$

$$w_{14}(new) = \Delta w_{14} + w_{14}(old) = -0.00287 + 0.4$$

$$= 0.3971$$

Similarly, update all other weights

i	j	w_{ij}	δ_j	x_i	r	updated w_{ij}
1	3	0.1	-0.00265	0.35	1	0.0991
2	3	0.8	-0.00265	0.9	1	0.7976
1	4	0.4	-0.0082	0.35	1	0.3971
2	4	0.6	-0.0082	0.9	1	0.5926
3	5	0.3	-0.0406	0.68	1	0.2724
4	5	0.9	-0406	0.6637	1	0.8731