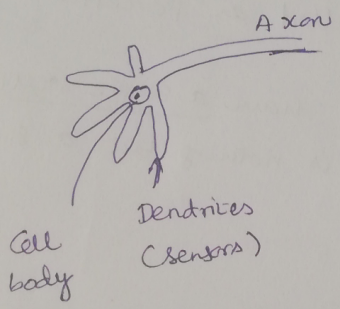
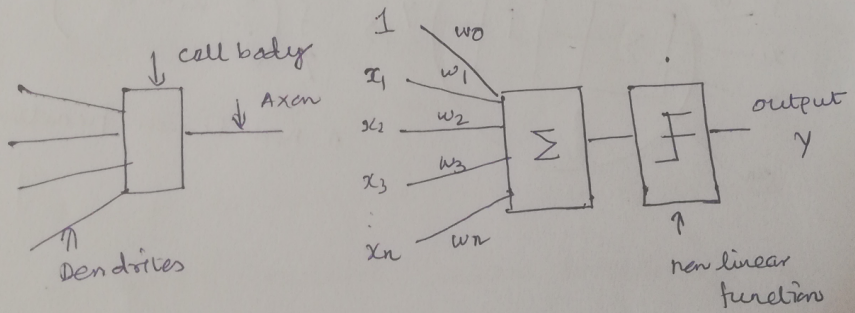
14/3/2020 :

Neural networks for PR :

It actually immitates the actual brain which consists of number of cells, whole thing is called cell body



A neuron model for machine intelligence is



$$y = f\left(\sum_{i=1}^{n} x_i w_i + w_0\right) \quad > T \cdot \quad \boxed{y = 1}$$

$$\leq T \quad \boxed{y = 0}$$

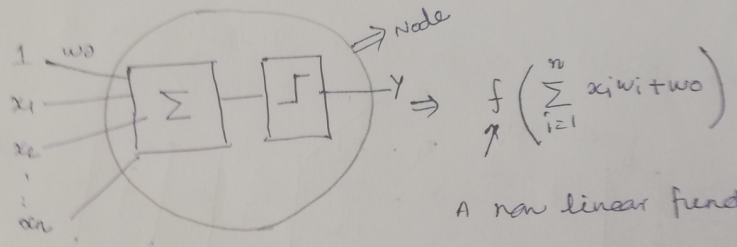
The model is closely similar to what we have followed in Linear Discriminant function.

(2)

The aim is to select proper weight vectors, which can classify the classes ⇒ this can be done during training.

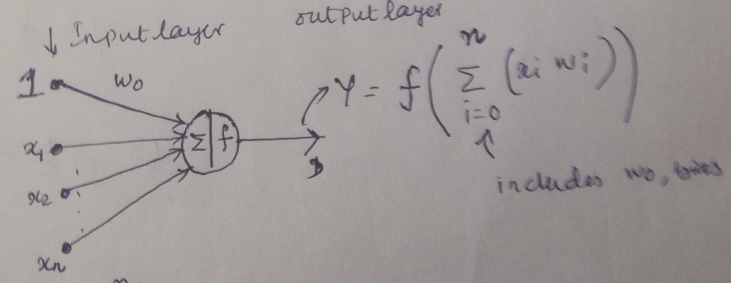
As, this model is very close to perceptron criterion, A neural networks formed out of such neuron model, they are also called as perceptron.

In the simplest case, we can have a single output single layer perceptron. which is nothing but a single neuron in neural network.



$$f\left(\sum_{i=1}^{n} x_i w_i + w_0\right)$$

A single layer single output perceptron: Fig. 1



$$Y = f\left(\sum_{i=0}^{n} (x_i w_i)\right)$$

includes w0, bias

$$g_i(x) = \sum_{i=0}^{n} w_i x_i + w_0 \begin{cases} > 0 \Rightarrow x \in \omega_1 \\ \leq 0 \Rightarrow x \in \omega_2 \end{cases}$$

let the target output is d_p, and the actual output is $D_p = \sum_{i=0}^{n} w_i x_i^p$

So the error 'e' is $= e = D - d$ (actual ↓, target ↙)

And hence sum of squared error

$$E = \frac{1}{2} \sum_{p=0}^{n} (D_p - d_p)^2 \quad -(1)$$

over, every sample we have an error $(D-d)$ and sum for all samples gives us $E = \sum_{p=0}^{n} (D_p - d_p)^2$

(*) This sum of squared error is actually a function of 'w', where 'w' is weight.

(*) our aim is to minimize this sum of squared error 'E' using gradient descent procedure.

$$\frac{\partial E}{\partial w_i} = (D-d)\, x_i \quad \text{from } (E) - (1)$$

Accordingly, weight updation rule is

$$w(k+1) = w(k) - \eta\,(D-d)\, x_i$$

In perceptron criteria

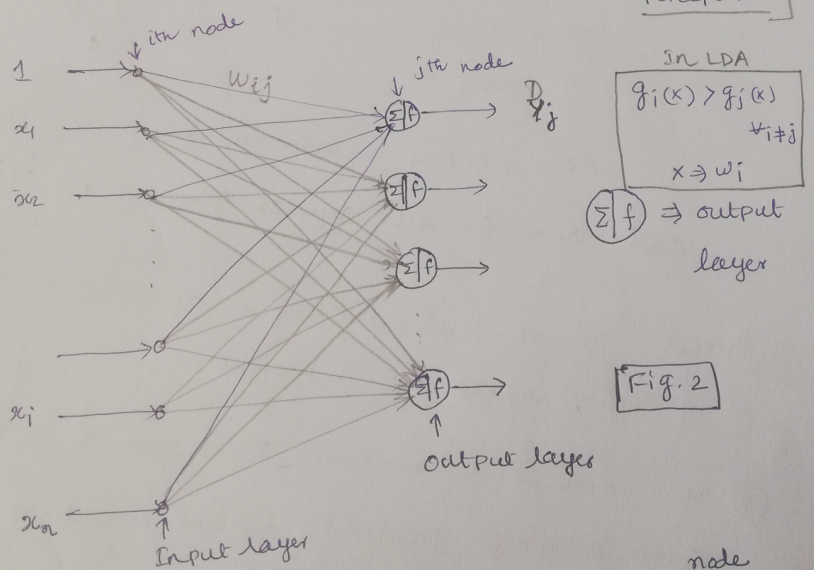
$$w(k+1) = w(k) + \eta\, Y$$

misclassified

④

* This single output perceptron actually neuron, takes care of only two class problem.

* But, If we have multiple class problem, then we should have neural network having multiple number of outputs. [A single layer multiple output perceptron]



In LDA

$g_i(x) > g_j(x)$

$\forall i \neq j$

$x \Rightarrow w_i$

$\Sigma | f \Rightarrow$ output layer

Fig. 2

w_{ij} → weight of the connection from ith node of the input layer to jth node of the output layer

Dy_j → ~~is the output of the jth node~~ jth node in the output layer.

$$Dy_j = f\left(\sum_{i=0}^{n} w_{ij} \cdot x_i\right)$$

↑ bias

f - some non-linear function, which could be sigmoidal function (or) threshold function

(*) In single layer single output NN, the linear equation $\sum_{i=0}^{n} w_i x_i$ is nothing but equation of a single st. line.

It actually divides the feature space into two half planes.

(*) Extending this, an SLMO NN, each of the output neuron in the output layer $\hat{y}_j$ actually defines the equation of the st. line

$$Y_j = f\left(\sum_{i=0}^{n} w_{ij} \cdot x_i\right)$$

If we have 'c' class problem, we can have 'c' number of nodes in the output layer. Every node corresponds to one class.

(*) Each of the node defines equation of the st. line. (or) Each of them gives us a linear equation which is the discriminant function.

individual classes
for an ~~binary classification~~, for a given input which of the output layer node gives the maximum o/p, and the sample is classified into that class. which is nothing but a discriminant function.

So, It is assumed that the classes are linearly separable. what ~~is~~ we do if the classes are not linearly separable? (both SO, MO) single o/p multiple o/p

Then, this single layer NN is not sufficient, we have to go for multiple layer in Neural network. ~~In~~ when we go for multiple layers, what we get is a 'non-linear boundary' is actually broken into linear pieces. Piece wise approximation of non linear boundary.

Lect 25

A Single layer single output perceptron:

Refer fig. 1 on Page 1

$$D = \sum_{i=0}^{n} w_i x_i$$

we know, what is expected output is the target output, and what we actually get is the actual output and the difference between these two is the error.

let expected / Target output is 'd'

and the actual output is 'D' which is given by $D = \sum_{i=0}^{n} w_i x_i$

$\downarrow$ a.o $\quad$ $\swarrow$ T.o

$e = D - d$ and the squared error is

$E = \frac{1}{2}(D-d)^2$ and our aim is to minimize this sum of squared error 'E'.

use gradient descent procedure,

$$\frac{\partial E}{\partial w_i} = \frac{2}{2}(D-d) \cdot \frac{\partial (D-d)}{\partial w_i}$$

$$\boxed{\frac{\partial E}{\partial w_i} = (D-d) \cdot x_i}$$

(8)

$\frac{\partial (D-d)}{\partial w_i}$

Target o/p is fixed which is not depend upon w_i

$$\frac{\partial}{\partial w_i}(D-d) = \frac{\partial}{\partial w_i} D = \frac{\partial}{\partial w_i}\left[\sum_{i=0}^{n} w_i x_i\right] = x_i$$

(*)

weight updation rule:

$w_i(0) \leftarrow$ randomly

$$w_i(k+1) = w_i(k) - \eta \frac{\partial E}{\partial w_i}$$

$$\boxed{w_i(k+1) = w_i(k) - \eta (D-d) \cdot x_i}$$

The convergence of the algorithm is until all the samples are correctly classified (or) the sum of squared error (E) is minimum. This is the termination | convergence criteria for this training algorithm.

15/3/2020 lect 25

Single layer multiple output neural network:

It is similar to linear machine, for every class ω_i, we have $g_i(x)$.

If $g_i(x) > g_j(x), \forall i \neq j$

$\Rightarrow x \in \omega_i$

Refer Fig. 2 actual value of the j^{th} neuron $D_j = \sum_{i=0}^{n} w_{ij} \cdot x_i$

In case of SLSO, the output is a scalar.

In case of SLMO, the output is a vector.

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ \vdots \\ C_M \end{bmatrix}$$

If the class is from ω_3, only the third value in the o/p should gives us neuron value 1, while the others gives us neuron value '0'.

we can define the sum of squared error as

$$E = \frac{1}{2} \sum_{i=1}^{M} (D_j - d_j)^2$$

D_j = actual o/p j^{th} neuron

d_j = Target o/p of j^{th} neuron

no. of nodes in the output layer.

(10)

$$E = \frac{1}{2} \sum_{j=1}^{M} (D_j - d_j)^2$$

$$\frac{\partial E}{\partial w_{ij}} = (D_j - d_j) \cdot x_i$$ following the same procedure (*)

weight updation rule:

Training algorithm

$w_{ij}(0) \leftarrow$ random

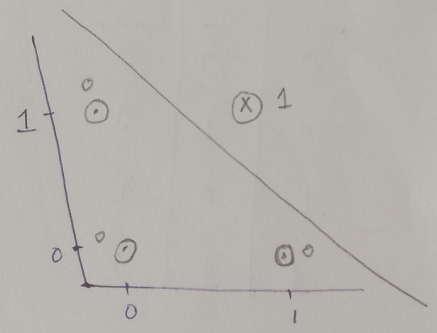
$w_{ij}(k+1) \leftarrow w_{ij}(k) - \eta (D_j - d_j) \cdot x_i$

(D_j: actual, d_j: Target)

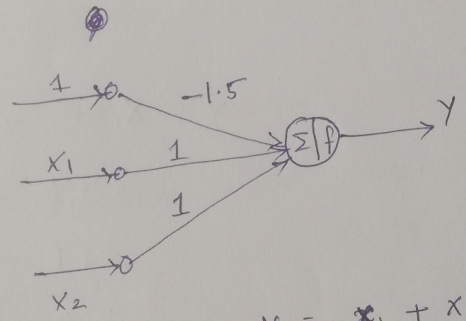
Examples for linearly separable cases:

AND Gate:

X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1



we can have, single layer single output NN which immitates AND Gate.



$$y = x_1 + x_2 - 1.5$$

$$= 0 + 0 - 1.5 < 0 \quad \left.\begin{matrix} \\ \\ \end{matrix}\right\} -1.5$$

$$0 + 1 - 1.5 < 0 \quad \left.\begin{matrix} \\ \\ \end{matrix}\right\} -0.5$$

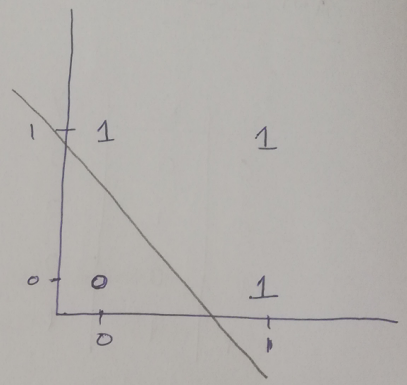
$$1 + 0 - 1.5 < 0 \quad \left.\begin{matrix} \\ \\ \end{matrix}\right\} -0.5$$

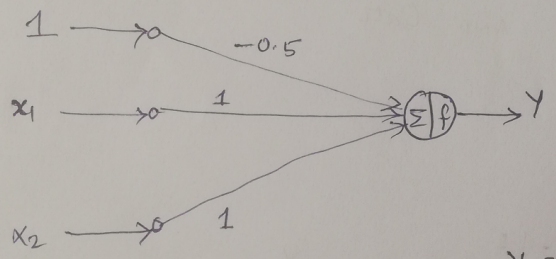
$$1 + 1 - 1.5 > 0 \quad \left.\begin{matrix} \\ \\ \end{matrix}\right\} +0.5$$

(12) Similarly,

OR Gate:

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





$Y = x_1 + x_2 - 0.5$

$0 + 0 - 0.5 = -0.5 < 0$

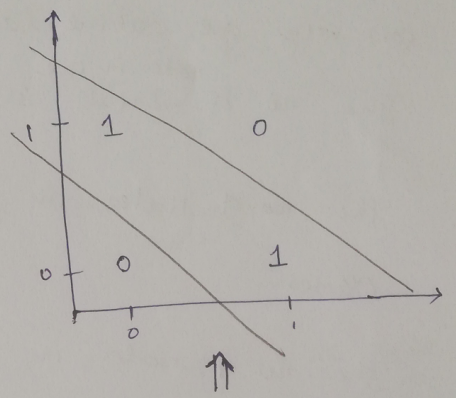
$0 + 1 - 0.5 = 0.5 > 0$

$1 + 0 - 0.5 = 0.5 > 0$

$1 + 1 - 0.5 = 1.5 > 0$

If we consider X-OR Gate:

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



we can't have the equation of a single st. line which can divide the space into such sort of the region, so naturally a single layer NN can't gives us a solution for this problem. Exactly this kind of situation, where the region is not linearly separable, we have to go for **multilayer Neural network.**