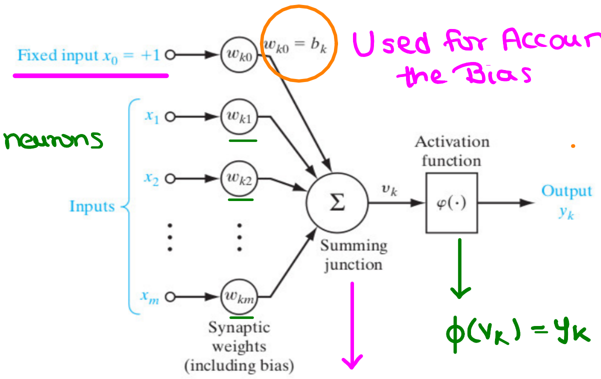
TEJAS KHAIRNAR

★ Neuron Model

m neurons



Used for Accounting the Bias

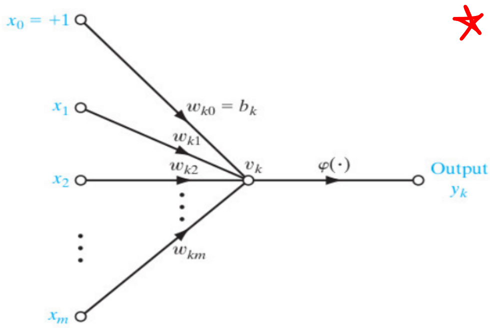
$\phi(v_k) = y_k$

★ Adder Function

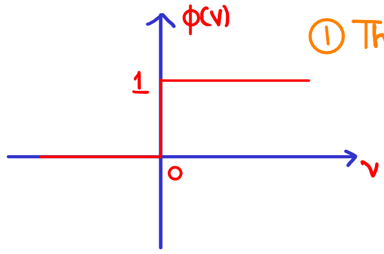
$$v_k = \sum_{j=0}^{m} w_{kj} x_j$$

$w_{k0} = b_k$, $x_0 = +1$ } Bias accounting

★ v_k we have w_{kj}



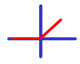
★ Directed Graph Rep of the Network

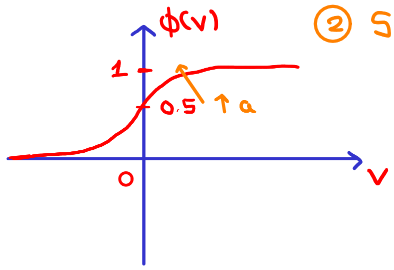


① Threshold Activation Function

③ Signum Activation Function

④ ReLU



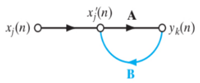


② Sigmoid Activation Function

$$\frac{1}{1 + e^{-va}}$$

→ where a is a constant

★ Feedback Loop



- Three Nodes are there $x_j(n), x_j'(n)$ and $y_k(n)$
- Two black colored directed links
- One blue colored directed link
- Node $x_j'(n)$ has two input links
  - One from node $x_j(n)$
  - One from node $y_k(n)$

$y_k(n) = \mathbf{A}[x_j'(n)]$

$x_j'(n) = x_j(n) + \underbrace{\mathbf{B}[y_k(n)]}_{feedbackoutput}$ (Addition in Directed Graph)

$y_k(n) = \mathbf{A}[x_j(n) + \underbrace{\mathbf{B}[y_k(n)]}_{feedbackoutput}\ ]$

$= \mathbf{A}[x_j(n)] + \mathbf{A}\ \underbrace{\mathbf{B}[y_k(n)]}_{feedbackoutput}$

$= \mathbf{A}[x_j(n)] + \mathbf{AB}[y_k(n)]$

$y_k(n) = \frac{\mathbf{A}}{(1-\mathbf{AB})}[x_j(n)]$

$$\boxed{y_k(n) = \frac{A}{1-AB}[x_j(n)]}$$

★ A search direction d^t is a descent direction at point x^t if the condition $\nabla f(x^t).d^t \leq 0$ is satisfied

→ $-\nabla f(x^t)$ → Direction of max descent

★ Gradient Descent

★ Here t is the current time / iteration

★ Descent direction Condition:

→ $f(x^{t+1}) = f(x^t + \alpha \nabla f(x^t).d^t) < f(x^t)$

Algorithm

Step 1 Choose a maximum number of iterations M to be performed, an initial point $x^{(0)}$, two termination parameters ϵ_1, ϵ_2, and set $k = 0$.

Step 2 Calculate $\nabla f(x^{(k)})$, the first derivative at the point $x^{(k)}$.

Step 3 If $\|\nabla f(x^{(k)})\| \leq \epsilon_1$, **Terminate**; (Slope almost zero)
Else if $k \geq M$; **Terminate**; (No of Iterations Exceeded)
Else go to Step 4.

Step 4 Perform a unidirectional search to find $\alpha^{(k)}$ using ϵ_2 such that $f(x^{(k+1)}) = f(x^{(k)} - \alpha^{(k)} \nabla f(x^{(k)}))$ is minimum. One criterion for termination is when $|\nabla f(x^{(k+1)}) \cdot \nabla f(x^{(k)})| \leq \epsilon_2$.

Step 5 Is $\frac{\|x^{(k+1)} - x^{(k)}\|}{\|x^{(k)}\|} \leq \epsilon_1$? If yes, **Terminate**;
Else set $k = k + 1$ and go to Step 2.

★ Binary Search Type Algorithm

Interval halving method - algorithm

Step 1 Given interval (a, b), choose ϵ. Let $x_m = \frac{(a+b)}{2}$; $L = (b - a)$

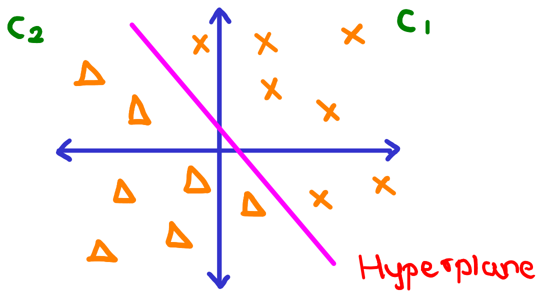
Step 2 Initialize $x_1 = a + \frac{L}{4}$; $x_2 = b - \frac{L}{4}$; Compute $f(x_1), f(x_2)$

Step 3 If $f(x_1) < f(x_m)$ then $b = x_m; x_m = x_1$; Go to step 5; else go to step 4

Step 4 If $f(x_2) < f(x_m)$ then $a = x_m; x_m = x_2$; Go to step 5; else $a = x_1, b = x_2$; go to step 5; (Very Important)

Step 5 Calculate $L = (b - a)$. If ($|L| < \epsilon$) terminate else go to step 2

Linearly Seperable data



$W = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} \rightarrow$ Coefficients

$x = \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} \rightarrow$ Inputs

★ Can be generalized to higher dimmensions

$\eta \rightarrow$ Learning Rate

★ Higher the value of the learning rate faster the opti sol arrival but less accuracy.

★ $w^T x > 0 \quad \forall\ x \in C_1$

★ $w^T x \leq 0 \quad \forall\ x \in C_2$

Update Rule:

1) $w(n+1) = w(n)$ if $w^T(n)x(n) > 0$ & $x(n) \in C_1$
   $w(n+1) = w(n)$ if $w^T(n)x(n) \leq 0$ & $x(n) \in C_2$

2) $w(n+1) = w(n) - \eta(n)x(n)$ if $w^T(n)x(n) > 0$ & $x(n) \in C_2$
   $w(n+1) = w(n) + \eta(n)x(n)$ if $w^T(n)x(n) \leq 0$ & $x(n) \in C_1$

Let $w = w(0) = 0$
& $\eta(n) = 1$

$\Rightarrow x(n) \in C_1$ & $w^T(n)x(n) \leq 0$

★ $w(n+1) = w(n) + x(n)$ ($\because \eta(n) = 1$)

$\Rightarrow w(n+1) = x(1) + x(2) + \cdots + x(n)$ .... ①

★ If C_1 & C_2 are linearly seperable
$\exists\ w_0 \ni w_0^T(n)x(n) > 0$
for $x(1), x(2), \ldots, x(n) \in C_1$

Let $\alpha = \min\limits_{x(n) \in C_1} w_0^T(n)x(n)$

★ Hence $\rightarrow w_0^T w(n+1) \geq n\alpha$ (# Multipy both sides)

★ $\|w_0^T\| \, \|w(n+1)\| \geq w_0^T w(n+1)$ (Cauchy Schwarz)

$\Rightarrow \|w(n+1)\|^2 \geq \dfrac{n^2\alpha^2}{\|w_0\|^2}$

★ $w(k+1) = w(k) + x(k)$

$\Rightarrow \|w(k+1)\|^2 - \|w(k)\|^2 \leq \|x(k)\|^2$

Let $\beta = \max\limits_{x(k) \in C_1} \|x(k)\|^2$

Then $\|w(n+1)\|^2 \leq n\beta$

$\Rightarrow n\beta \geq \dfrac{n^2\alpha^2}{\|w_0\|^2}$

$\Rightarrow n_{max} = \dfrac{\beta \|w_0\|^2}{\alpha^2}$ (Max Number of iterations)

★ Algorithm

Initialization	Set $\mathbf{w}(0) = \mathbf{0}$; Perform following computations for n = 1, 2, ...
Activation	At time step n, provide the input vector $\mathbf{x}(n)$ and desired response $d(n) \rightarrow$ desired output
Response	$sgn(\mathbf{w}^T(n)\mathbf{x}(n))$ Output is {-1, +1} $\rightarrow y(n)$
Adaptation	$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta\,[d(n) - y(n)]\,\mathbf{x}(n)$ Where $d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \in \mathcal{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \in \mathcal{C}_2 \end{cases}$
Iterate	Increment n and go to activation step

★ Example:

If $w^T(n)x(n) > 0$

$\Rightarrow y(n) = +1$

if $x(n) \in C_1 \Rightarrow d(n) = 1 \rightarrow w(n+1) = w(n)$

if $x(n) \in C_2 \Rightarrow d(n) = -1 \rightarrow w(n+1) = w(n) - 2\eta x(n)$

★ Batch Algorithm

- Compute: $\mathbf{w}^T(n)\mathbf{x}(n)$
- Treat the above quantity as the objective function
- With the modification $\mathbf{w}^T(n)\mathbf{x}(n)d(n)$
- For one $\mathbf{x}(n)$ the above objective function is used:
- For many $\mathbf{x}(n)$'s we have:

$$J(\mathbf{w}) = \sum_{\mathbf{x}(n)\in\mathcal{H}} \left(-(\mathbf{w}^T(n)\mathbf{x}(n)d(n))\right)$$

- The above objective function should be minimized

→ Objective Cost Function

- We have to minimize or maximize a given objective function
- Percentron rule: $\mathbf{w}^T(n)\mathbf{x}(n) > 0$ $\mathbf{x}(n) \in \mathcal{C}_1$
- $\mathbf{w}^T(n)\mathbf{x}(n)$ quantity for $\mathcal{C}_1$ is positive, d(n) = +1. Decrease it by multiplying it -1
- Percentron rule: $\mathbf{w}^T(n)\mathbf{x}(n) \leq 0$ $\mathbf{x}(n) \in \mathcal{C}_1$
- $\mathbf{w}^T(n)\mathbf{x}(n)$ quantity for $\mathcal{C}_1$ is negative, d(n) = -1. Decrease it by multiplying it -1
- That is for any $\mathbf{x}(n)$, the quantity $-(\mathbf{w}^T(n)\mathbf{x}(n)d(n))$ to be minimized

→ Intuition

- Compute direction: $\bigtriangledown J(\mathbf{w}) = \sum_{\mathbf{x}(n)\in\mathcal{H}} (-\mathbf{x}(n)d(n))$
- Update $\mathbf{w}(n+1) = \mathbf{w}(n) - \eta(n) \bigtriangledown J(\mathbf{w})$
- That is $\mathbf{w}(n+1) = \mathbf{w}(n) - \eta(n) \sum_{\mathbf{x}(n)\in\mathcal{H}} (-\mathbf{x}(n)d(n)))$

→ Gradient Descent Rule

★ MultiLayer Perceptron

- error at each neuron : $e_j(n) = d_j(n) - y_j(n)$
- Instantaneous error energy $\mathcal{E}_j(n) = \frac{1}{2} e_j^2(n)$
- Avg error $\mathcal{E}_{av}(N) = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j\in C} e_j^2(n)$ → Here N = no. of training examples

- $\mathcal{E}_{av} \to \mathcal{E}(n)$
- $\mathcal{E}(n) \to e_j(n)$
- $e_j(n) \to y_j(n)$
- $y_j(n) \to v_j(n)$
- $v_j(n) \to w_{ji}(n)$

$\mathcal{E}(n) = \frac{1}{2} \sum_{j\in C} e_j^2(n)$

$e_j(n) = d_j(n) - y_j(n)$

$y_j(n) = \phi(v_j(n))$

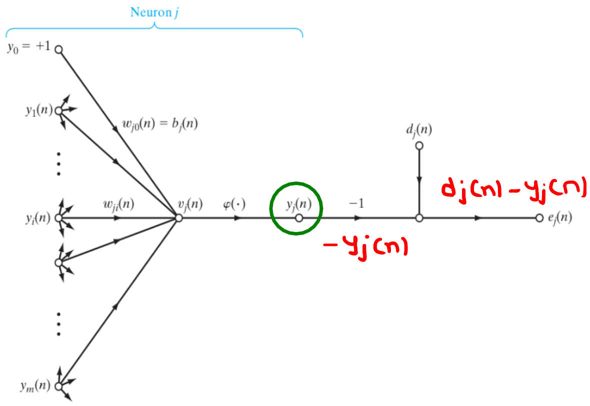
$\sum_{i=0}^{m} w_{ji}(n) y_i(n) = v_j(n)$

Dependencies used for gradient cal

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = \underbrace{\frac{\partial \mathcal{E}(n)}{\partial e_j(n)}}_{e_j(n)} \underbrace{\frac{\partial e_j(n)}{\partial y_j(n)}}_{-1} \underbrace{\frac{\partial y_j(n)}{\partial v_j(n)}}_{\phi_j'(v_j(n))} \underbrace{\frac{\partial v_j(n)}{\partial w_{ji}(n)}}_{y_i(n)}$$

Chain Rule

★ Back Propogation Algo Online



$d_j(n) - y_j(n)$

$-y_j(n)$

★ IF j is a hidden layer we dont know the desired output hence cant find $e_j(n)$

- $w_{ji}(n+1) = w_{ji} - \eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$
- $\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$
- $\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$
- Where $\delta_j(n)$ is the local gradient at v_j (before the activation function)
- Local gradient is computed using chain rule as:

★ Local gradient output neuron.

$$★\ \delta_j(n) = \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n) \times -1 \times \phi'_j(v_j(n))$$

$$(output)\delta_j(n) = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$

$$(hidden)\delta_j(n) = \frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = \frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \phi'(v_j(n))$$

★ When k is a hidden neuron, the error is computed by summing all the hidden neuron errors. $\Rightarrow \mathcal{E}(n) = \frac{1}{2} \sum_k e_k^2(n)$

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} = \sum_k e_k(n) \underbrace{\frac{\partial e_k(n)}{\partial v_k(n)}}_{\phi'_k(v_k(n))} \underbrace{\frac{\partial v_k(n)}{\partial y_j(n)}}_{w_{kj}(n)}$$

★ $e_k(n) = d_k(n) - \phi_k(v_k(n))$

★ $v_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_j(n)$

- The complete derivative is: $\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n)$

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} = \sum_k e_k(n) \phi'_k(v_k(n)) w_{kj}(n) = \sum_k e_k(n) \phi'_k(v_k(n)) w_{kj}(n) = \sum_k \delta_k(n) w_{kj}(n)$$

- For all the k^{th} neurons in the forward layer that connect to j^{th} neuron

★ Gradient Descent rule in Online MLP

$$\underbrace{\Delta w_{ji}(n)}_{\text{Weight correction}} = \underbrace{\eta}_{\text{Learning rate}} \times \underbrace{\delta_j(n)}_{\text{Local gradient}} \times \underbrace{y_i(n)}_{\text{Input signal at neuron } j}$$

- $w_{ji}(n+1) = w_{ji}(n) - \eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$
- $\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$
- $\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$

$$\delta_j(n) = \begin{cases} a[d_j(n) - o_j(n)] o_j(n)[1 - o_j(n)] & \text{if } j \text{ is output neuron} \\ a\, y_j(n)[1 - y_j(n)] \sum_k \delta_k(n) w_{kj}(n) & \text{if } j \text{ is a hidden neuron} \end{cases}$$

$\rightarrow$ for sigmoid function:

$\phi_j(v_j(n)) = \frac{1}{1 + e^{-a v_j(n)}}$

$\Rightarrow \phi'_j(v_j(n)) = a\, y_i(n)[1 - y_i(n)]$

\# $y_i(n) = \phi_j(v_j(n))$