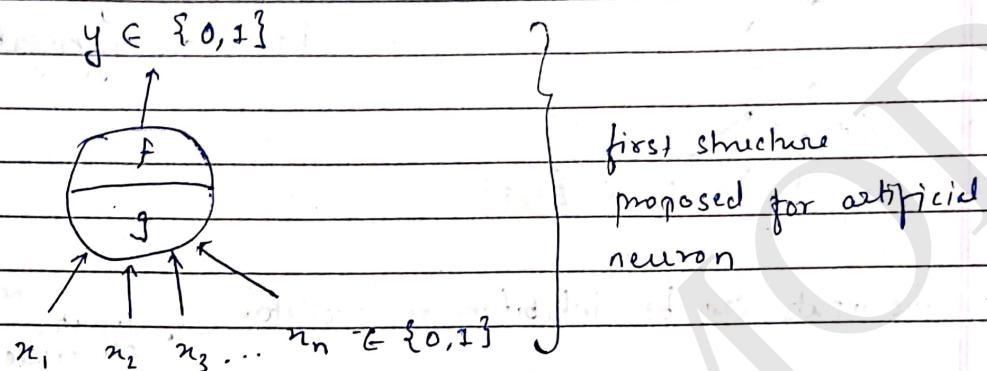


# DEEP LEARNING

~~WEEK 1~~

(Brief discussion on history of deep learning)

## \* Artificial Neuron



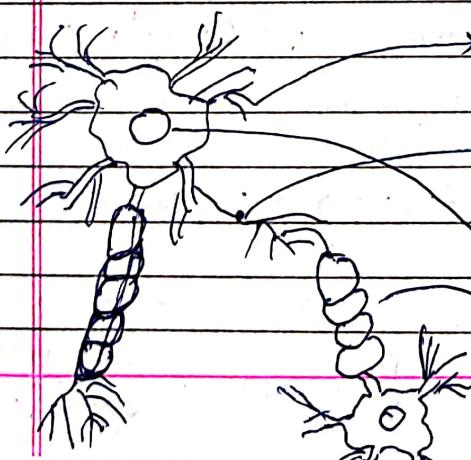
## \* Image Net challenge architectures

Network      Error      Layers

(1) Alex Net	16.1%	8
(2) ZF Net	11.2%	8
(3) VGG Net	7.3%	19
(4) Google Net	6.7%	22
(5) MS ResNet	3.6%	152

~~WEEK 2~~

## \* Biological Neuron



\* dendrite - receive signals from other neurons

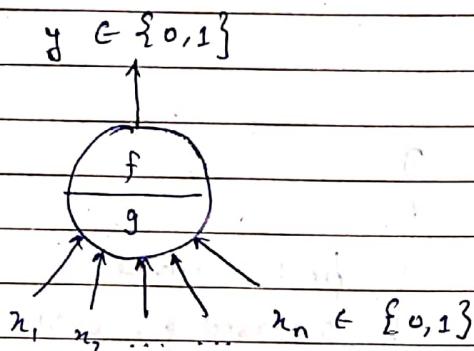
\* Synapse - point of connection between two neurons

\* Soma - processes the information

\* Axon - transmits the output of this neuron

## \* Artificial Neuron

→ McCulloch & Pitts (MP neuron)

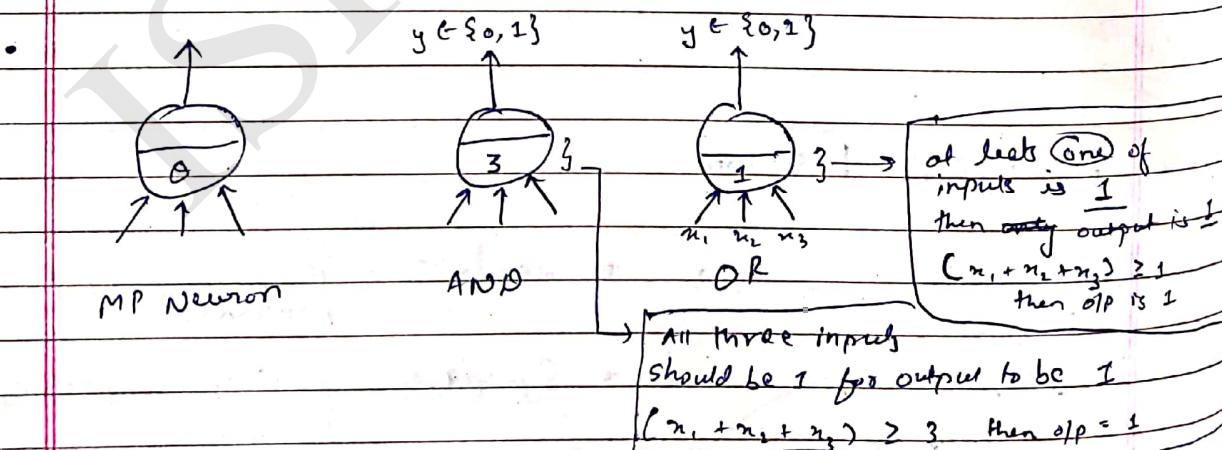


- $g$  aggregates the inputs
- $f$  function  $f$  takes decision based on aggregation

- The inputs can be inhibitory or excitatory ✓ when inputs combined can create an output
- does not matter what the other inputs are the output depends directly on this characteristic of inhibitory inputs → (if ~~one~~ input is 1 output of neuron is 0)
- eg  $g(x_1, x_2, \dots, x_n) = g(n) = \sum_{i=1}^n x_i$

$$\begin{aligned} y = f(g(n)) &= 1 \quad \text{if } g(n) \geq 0 \\ &= 0 \quad \text{if } g(n) < 0 \end{aligned}$$

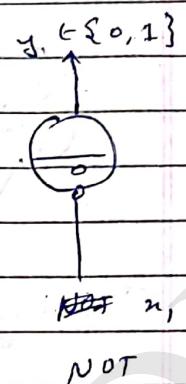
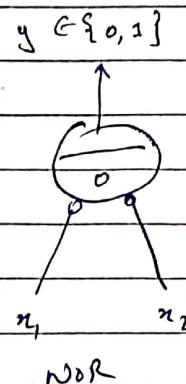
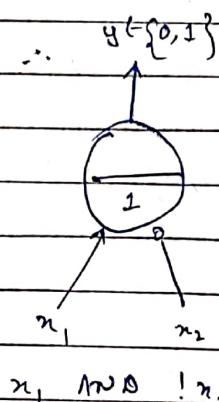
$\theta$  is thresholding parameter



• indicates inhibitory input

Here in below example

$$\left\{ \begin{array}{l} (x_1 + x_2) \geq 1 \\ (x_1 + x_2) \geq 1 \\ x_1 \geq 0 \end{array} \right\} \quad \left\{ \begin{array}{l} \text{threshold obtained} \\ \text{if inhibitory} \\ \text{input is considered} \end{array} \right\} [1]$$



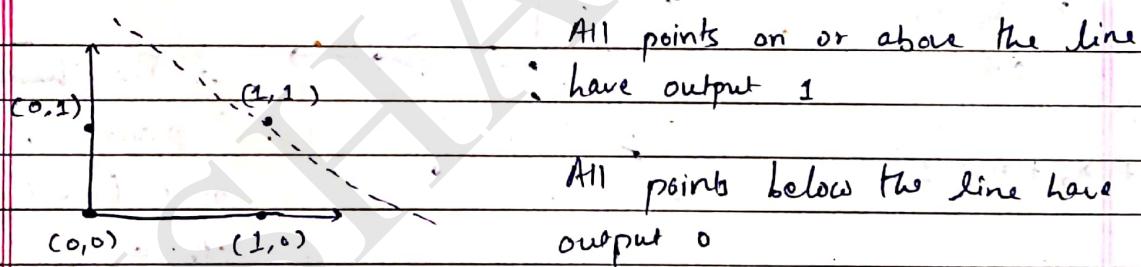
$(n_1 + n_2) \geq 1$   
then output = 1  
(because inhibitory input)

$(n_1 + n_2) \geq 0$   
 $o/p = 1$   
All inputs are inhibit they

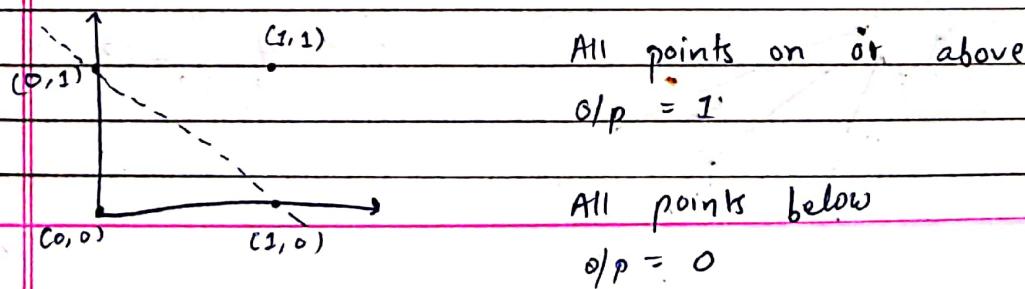
$n_1 \geq 0$   
 $o/p = 1$

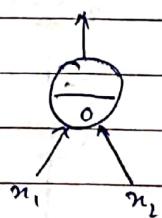
### → Geometric Interpretation (2D)

- AND  $\rightarrow$ 
 $\therefore n_1 + n_2 \geq 2$



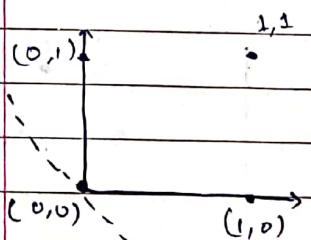
- OR  $\rightarrow$ 
 $\therefore n_1 + n_2 \geq 1$





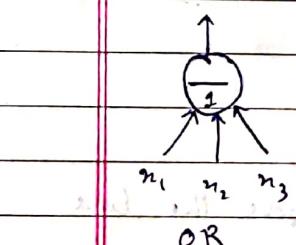
$$n_1 + n_2 \geq 0$$

Tautology (Means ON)



All points on or above  
 $\text{o/p} = 1$

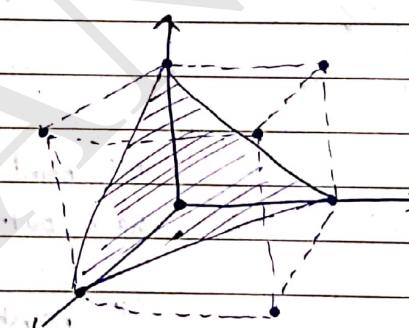
→ Geometric Implementation (3D)



OR

$$n_1 + n_2 + n_3 \geq 1$$

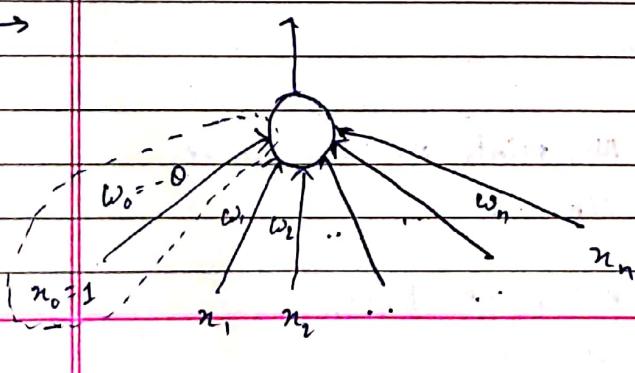
$(0,0,0) \rightarrow \text{o/p} = 0$   
else  $\rightarrow \text{o/p} = 1$



∴ The shaded  
plane separates  
point  $(0,0,0)$  from  
others

\* Perception

→



$$y = 1 \text{ if } \sum_{i=1}^n w_i x_i \geq 0$$

$$= 0 \text{ if } \sum_{i=1}^n w_i x_i < 0$$

Rewriting equation

$$y = 1 \text{ if } \sum_{i=1}^n w_i x_i - \theta \geq 0$$

$$0 \text{ if } \sum_{i=1}^n w_i x_i - \theta < 0$$

Now

$$\begin{aligned} y &= 1 & \text{if } \sum_{i=0}^n w_i x_i \geq 0 & \left. \begin{array}{l} \text{here} \\ w_0 = -\theta \end{array} \right\} \\ &0 & \text{if } \sum_{i=0}^n w_i x_i < 0 & \left. \begin{array}{l} \\ w_0 = 1 \end{array} \right\} \end{aligned}$$



$x_1 \quad x_2 \quad \text{OR}$

$$\begin{array}{ccc|c} 0 & 0 & 0 & w_0 + \sum_{i=1}^2 w_i x_i < 0 \\ 0 & 1 & 1 & w_0 + \sum_{i=1}^2 w_i x_i \geq 0 \\ 1 & 0 & 1 & w_0 + \sum_{i=1}^2 w_i x_i \geq 0 \\ 1 & 1 & 1 & w_0 + \sum_{i=1}^2 w_i x_i \geq 0 \end{array}$$

$$\therefore w_0 + 0 \cdot w_1 + 0 \cdot w_2 < 0$$

$$\Rightarrow w_0 < 0 \rightarrow ①$$

$$w_0 + 0 \cdot w_1 + 1 \cdot w_2 \stackrel{?}{=} 0$$

$$\Rightarrow w_2 > -w_0 \rightarrow ②$$

$$w_0 + 1 \cdot w_1 + 0 \cdot w_2 \geq 0$$

$$\Rightarrow w_1 > -w_0 \rightarrow ③$$

$$w_0 + 1 \cdot w_1 + 1 \cdot w_2 \geq 0$$

$$\Rightarrow w_1 + w_2 > -w_0 \rightarrow ④$$

4 equations form system of linear inequalities  
which can be solved to get values of  $w_0, w_1, w_2$

lets say we get solution

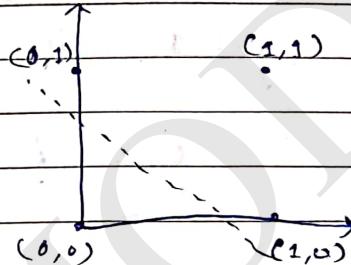
$$\omega_0 = -1, \omega_1 = 1.1, \omega_2 = 1.1$$

∴ line will be

$$\omega_0 + \omega_1 n_1 + \omega_2 n_2 = 0$$

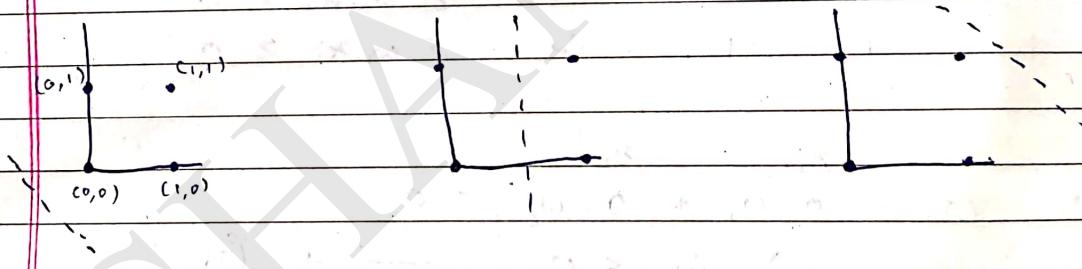
$$-1 + 1.1 n_1 + 1.1 n_2 = 0$$

$$\Rightarrow 1.1 n_1 + 1.1 n_2 = 1$$



## \* Errors and Error Surfaces

Considering at above mentioned OR Gate if the decision boundaries are as follows



$$\text{Error} = 1$$

Since (0,0) is  
misclassified

$$\text{Error} = 1$$

since (0,1) is  
misclassified

$$\text{Error} = 3$$

since (0,1), (1,1), (1,0)  
are misclassified

Equation of line  $\Rightarrow \omega_0 + \omega_1 n_1 + \omega_2 n_2 = 0$

thus we want values of  $\omega_0, \omega_1, \omega_2$  such that  
error is minimum and in best case it is 0.

## \* Perception Learning Algorithm

Let's say

$\begin{bmatrix} \vdots \\ x_0, x_1, x_2, \dots, x_n \end{bmatrix} \rightarrow \text{Attributes}$

$[w_0, w_1, w_2, \dots, w_n] \rightarrow \text{Weights}$

→ Algorithm

$P \leftarrow$  when all inputs result in target class 1

$N \leftarrow$  when all inputs result in target class 0

initialize random weight  $w$

while ! convergence do  $\rightarrow$  all data entries

pick random  $x \in (P \cup N)$

if  $x \in P$  and  $w \cdot x < 0$  then

$$w = w + x \quad \text{--- } \textcircled{1}$$

end

if  $x \in N$  and  $w \cdot x \geq 0$  then

$$w = w - x \quad \text{--- } \textcircled{2}$$

end

end

// the algorithm converges when all the inputs / rows of input are classified correctly

→ Intuition behind (1) & (2) statements

lets say,  $w \cdot x = \underbrace{w^T x}_{\text{dot product}} = \sum w_i x_i = 0$  is the line

then any point on this line say,  $\begin{bmatrix} x_0, x_1, x_2, x_3, \dots, x_n \end{bmatrix}$

random numbers satisfying  $w \cdot x = 0$

This point will be perpendicular to  $\omega$   
thus similar for all other points

$\therefore \omega$  is vector perpendicular to the line

because

$$\cos \alpha = \frac{\omega \cdot n}{\|\omega\| \|n\|}$$

$$\|\omega\| \|n\|$$

$$\Rightarrow \cos \alpha \times \|\omega\| \|n\| = \omega \cdot n$$

$$\underbrace{\cos \alpha}_{\substack{\downarrow \\ \text{can't be} \\ \text{zero}}} \times \underbrace{\|\omega\| \|n\|}_{\substack{\downarrow \\ \text{can't be} \\ \text{zero}}} = 0$$

$$\text{Thus } \underline{\alpha = 90^\circ}$$

$$\cos \alpha = 0$$

Now, from perceptron model

$$y = 1 \text{ if } \omega^T n \geq 0$$

$$y = 0 \text{ if } \omega^T n < 0$$

$$\therefore \cos \alpha = \frac{\omega \cdot n}{\|\omega\| \|n\|} = \frac{\omega^T n}{\|\omega\| \|n\|}$$

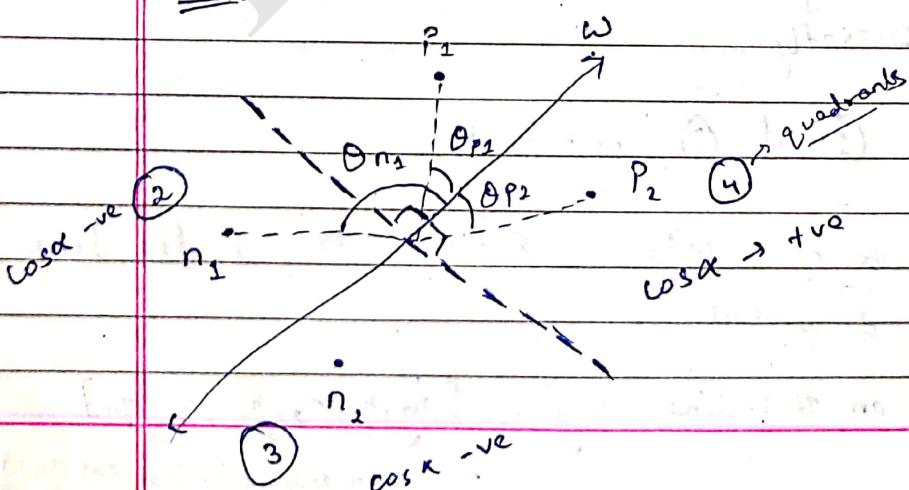
here if  $\omega^T n \geq 0$ ,  $\cos \alpha \geq 0$

$\omega^T n < 0$ ,  $\cos \alpha < 0$

( $\because \|\omega\| \text{ & } \|n\|$   
are  
always +ve)

Graph

(1)  $\cos \alpha > 0$



thus  
when  $\theta$   
is less than  
 $90^\circ$   $\cos \theta$  is  
+ve & when  
 $\theta > 90^\circ$   
 $\cos \theta$  is -ve

## Statement #

- Here, in (1)  $\rightarrow \omega = \omega + n$  | Here,  $\omega \cdot n < 0$

initially  $\cos \alpha = \omega^T n$

Now

$$\begin{aligned}\cos \alpha_{\text{new}} &= (\omega + n)^T n \\ &= \omega^T n + n^T n\end{aligned}$$

$$\cos \alpha_{\text{new}} = \cos \alpha + n^T n$$

$$\cos \alpha_{\text{new}} > \cos \alpha$$

$$\text{thus } \alpha_{\text{new}} < \alpha$$

- in statement (2)  $\rightarrow \omega = \omega - n$  | Here,  $\omega \cdot n \geq 0$

$$\cos \alpha = \omega^T n$$

Now,

$$\cos \alpha_{\text{new}} = \omega^T n - n^T n$$

$$\cos \alpha_{\text{new}} = \cos \alpha - n^T n$$

$$\cos \alpha_{\text{new}} < \cos \alpha$$

$$\text{thus } \alpha_{\text{new}} > \alpha$$

## → Proof of Convergence

Setup - If  $n \in N$  then  $-n \in P$  ( $\because \omega^T n < 0 \Rightarrow \omega^T -n \geq 0$ )

∴ We consider single set

$\nearrow N$  in which  $n$  is made  $-n$

$$P' = P \cup N^-$$

∴ every element  $p \in P'$ ,  $\omega^T p \geq 0$

## Improved Perception learning Algorithm

$P \leftarrow$  inputs which result into target class 1

$N \leftarrow$  inputs which result into target class 0

$N^-$  contains negation of all points (inputs) in  $N$

$p' \leftarrow PUN^-$

initialize  $w$  randomly

while !convergence do

pick random  $p \in P'$

$p \leftarrow p$  normalize to make unit norm  
 $\|p\|$

if  $w \cdot p < 0$  then

$w = w + p$

end

end

// Algorithm converges if all points are correctly classified

similar to earlier for  $n \in P$

but ~~for~~ for  $n \in N$  }  $\Rightarrow w^T w \geq 0$   
 since now,  $-n \in N^-$

( $\because$  if  $w \cdot p < 0$  then)  
 condition becomes

same for  $p \in N^-$   
 and thus for  $n$

## Proof

→ Suppose at step t we found that  $\omega^* \cdot p_i < 0$

thus we made correction  $\omega_{t+1} = \omega_t + p_i$

let  $\beta$  be angle between  $\omega^*$  and  $\omega_{t+1}$

assume that  $\omega^*$  is normalized unit vector and is the solution

$$\therefore \cos\beta = \frac{\omega^* \cdot \omega_{t+1}}{\|\omega_{t+1}\|} \quad (\because \|\omega^*\| = 1)$$

Now,

- Numerator =  $\omega^* \cdot \omega_{t+1} = \omega^* \cdot (\omega_t + p_i)$
- =  $\omega^* \cdot \omega_t + \omega^* \cdot p_i$
- $\geq \omega^* \cdot \omega_t + \delta \quad (\delta = \min\{\omega^* \cdot p_i \mid i\})$
- $\geq \omega^* \cdot (\omega_{t-1} + p_j) + \delta$
- $\geq \omega^* \cdot \omega_{t-1} + \omega^* \cdot p_j + \delta$
- $\geq \omega^* \cdot \omega_{t-1} + \cancel{\omega^*} \delta$
- $\geq \omega^* \cdot \omega_0 + k\delta \quad (\text{By induction})$

Here it is assumed that at time stamp t

(k) corrections are made such that  $k \leq t$

- Denominator<sup>2</sup> =  $\|\omega_t + \rho_i\|^2$

$$= (\omega_t + \rho_i) \cdot (\omega_t + \rho_i)$$

$$= \|\omega_t\|^2 + 2\omega_t \cdot \rho_i + \|\rho_i\|^2$$

$$\leq \|\omega_t\|^2 + \|\rho_i\|^2 \quad (\because \omega_t \cdot \rho_i < 0)$$

$$< \|\omega_t\|^2 + 1 \quad (\because \|\rho_i\|^2 = 1)$$

$$< (\|\omega_{t-1}\|^2 + 1) + 1$$

$$< \|\omega_0\|^2 + (k) \quad (\text{By induction})$$

Now

$$\cos \beta \geq \frac{\omega^* \cdot \omega_0 + k\delta}{\sqrt{\|\omega_0\|^2 + (k)}}$$

Now,

$\cos \beta$  is roughly proportional to  $\sqrt{k}$   
so as  $k$  grows  $\cos \beta$  grows

but  $\cos \beta$  can grow upto 1

since  $\cos \beta \leq 1$  thus after some finite  $k$  the algorithm converges

## \* Linearly inseparable boolean functions

$x_1 \quad x_2 \quad \text{XOR}$

$$0 \quad 0 \quad 0 \quad w_0 + \sum_{i=1}^2 w_i x_i < 0$$

$$0 \quad 1 \quad 1 \quad w_0 + \sum_{i=1}^2 w_i x_i \geq 0$$

$$1 \quad 0 \quad 1 \quad w_0 + \sum_{i=1}^2 w_i x_i \geq 0$$

$$1 \quad 1 \quad 0 \quad w_0 + \sum_{i=1}^2 w_i x_i < 0$$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \Rightarrow w_0 < 0$$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \Rightarrow w_2 \geq -w_0$$

$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \Rightarrow w_1 \geq -w_0$$

$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 < 0 \Rightarrow w_1 + w_2 < -w_0$$

not possible

Same can be observed graphically as well.

→ Boolean functions for two inputs

$x_1$	$x_2$	$f_1$	$f_2$	$f_3$	$\dots$	$f_{16}$
0	0	0	0	0		1
0	1	0	0	0		1
1	0	0	0	1		1
1	1	0	1	0		1

AND

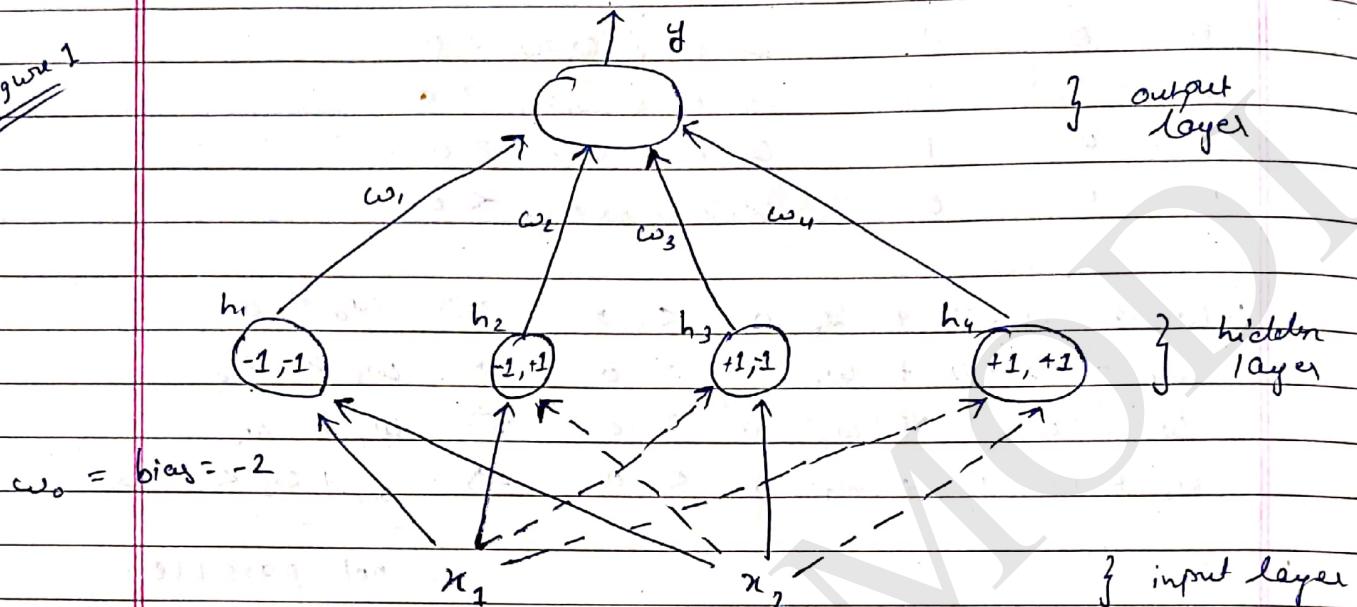
only XOR & !XOR from these 16 function are linearly inseparable

Formula for no. of functions  $\Rightarrow 2^n$  ( $n = \text{no. of inputs}$ )

# \* Network of Perceptrons

(Implementing linearly inseparable boolean functions)

Figure 1



$$w_0 = \text{bias} = -2$$

Here assume that  $x_1$  &  $x_2$  can assume two values

1  $\Rightarrow$  true

-1  $\Rightarrow$  False instead of zero.

Since bias = -2  $\therefore$  neurons will hit only if weighted sum  $\geq 2$

these lines have weights as mentioned  
 dotted line =  $w_1 = +1$       } connected with input layer  
 normal line =  $w_2 = -1$       }

Here,

$$h_1 = x_1 w_1 + x_2 w_2 = (-1 \cdot 1) + (1 \cdot -1) = -2 \geq 2$$

thus it fires

$$h_2 = x_1 w_2 + x_2 w_1 = (-1 \cdot -1) + (1 \cdot +1) = 2 \geq 2$$

thus it fires

$$h_3 = x_1 w_1 + x_2 w_2 = (+1 \cdot +1) + (-1 \cdot -1) = 2 \geq 2$$

thus it fires

$$h_4 = x_1 w_2 + x_2 w_1 = (1 \cdot 1) + (1 \cdot 1) = 2 \geq 2$$

thus it fires

Concept is called MLP (Multilayered Perceptrons)

(Theorem → we can implement any boolean function using MLP)

DELUXE

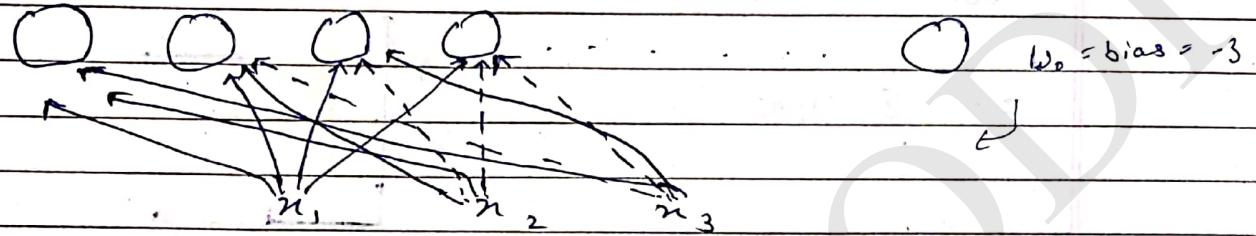
PAGE NO.:

DATE:

CloudTech

This was just for two inputs

If there are 3 inputs then the hidden layer contains  
 $2^3 = 8$  neurons



1	1	1	1
2	1	1	-1
3	1	-1	1
4	1	-1	-1
5	:	:	:
8	-1	-1	-1

Let's implement XOR gate with 2 inputs  $x_1, x_2$  using figure 1

$x_1$	$x_2$	XOR	$h_1$	$h_2$	$h_3$	$h_4$	$\sum_{i=1}^4 w_i h_i = y$
0	0	0	1	0	0	0	$w_1$
0	1	1	0	1	0	0	$w_2$
1	0	1	0	0	1	0	$w_3$
1	1	0	0	0	0	1	$w_4$

Here  $\sum w_i h_i \geq w_0$  for  $\text{XOR} = 1$

$\sum w_i h_i < w_0$  for  $\text{XOR} = 0$

$\therefore w_1 < w_0, w_2 \geq w_0, w_3 \geq w_0, w_4 < w_0$

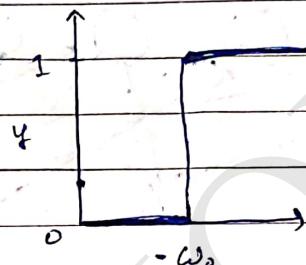
now we can randomly fix  $w_0$  and choose  $w_1, w_2, w_3, w_4$  and classification is done

WEEK 3

## \* Sigmoid Neurons

→ Need ?

$$z = \sum_{i=1}^n w_i x_i$$



based on  $w_0$  it is classified

if	$z = 0.51$	$\text{ans} = 1$	$\left. \begin{array}{l} \text{values} \\ \text{are close} \\ \text{but different ans} \end{array} \right\}$	$\rightarrow$ harsh classification
	$z = 0.49$	$\text{ans} = 0$		

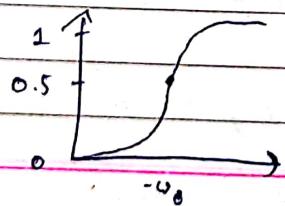
Thus we need sigmoid

$$\rightarrow \text{sigmoid}(z) = \frac{1}{1 + e^{-z}} \quad (e = 2.718)$$

Here  $z = \sum_{i=0}^n w_i x_i$  (includes bias)

- at  $z = \infty \Rightarrow \text{sigmoid}(z) = 1$
- at  $z = -\infty \Rightarrow \text{sigmoid}(z) = 0$
- at  $z = 0 \Rightarrow \text{sigmoid}(z) \approx 0.5$

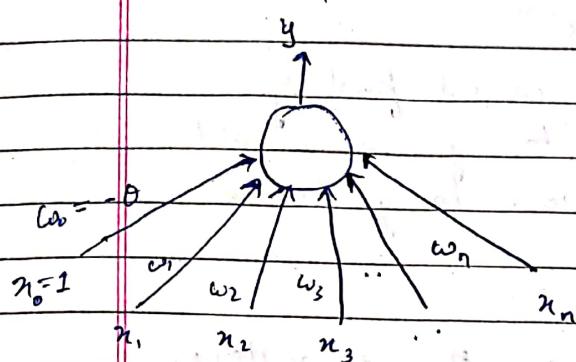
Sigmoid ranges from  $[0, 1]$  and thus it can be interpreted as probability



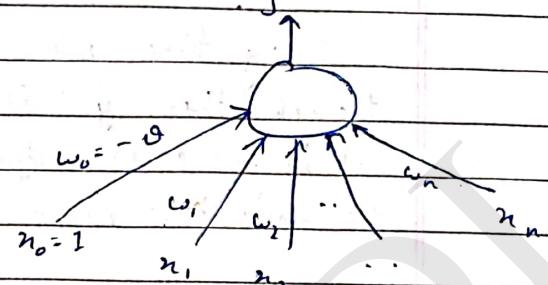
smooth function

(no sharp transition around threshold)

## Perception



## Sigmoid



$$y = 1 \text{ if } \sum_{i=0}^n w_i * x_i \geq 0$$

$$0 \text{ if } \sum_{i=0}^n w_i * x_i < 0$$

$$y = \frac{1}{1 + e^{-(\sum_{i=0}^n w_i * x_i)}}$$

(Smooth, continuous & differentiable)

## \* Supervised Machine Learning Setup



- Data :  $\{x_i, y_i\}_{i=1}^n$
- $\downarrow$        $\downarrow$   
set of all inputs      target output

- Model : Our approximation of relation between  $x$  &  $y$

$$\text{eg} \rightarrow \hat{y} = \frac{1}{1 + e^{-(w^T x)}} \quad (\text{logistic regression})$$

$$\hat{y} = w^T x \quad (\text{Linear regression})$$

$$\hat{y} = x^T w \quad (\text{quadratic form})$$

refer machine learning notes

w here are parameters which help us get close to the actual value.

lets say, if  $\hat{y} = \frac{1}{1+e^{-x}}$

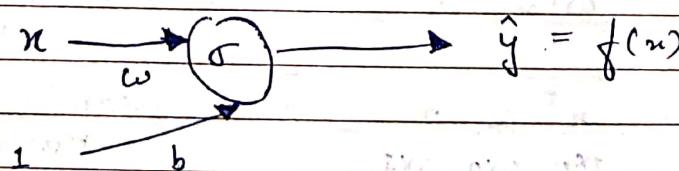
every time we get  $x$  we would find  $\hat{y}$ .  
though this  $\hat{y}$  is not close to  $y$  we could  
not be able to adjust to get close to  $y'$  because  
we don't have parameter like  $w$ .

- Parameters  $\rightarrow$  all the  $w$  are the parameters which is to be learned from data
- Learning Algorithm  $\rightarrow$  An algorithm for learning parameters of a model (eg - perceptron learning algo, gradient descent etc)
- Objective / Loss / Cost / Error function : guides learning algorithm

eg -  $L(w) = \sum_{i=1}^n (\hat{y}_i - y_i)^2$  } sum of squared errors

- why we use sum of squared errors
- It is differentiable
  - square is done because the positive & negative errors may cancel one another if square is not done

### \* Learning Parameters (guess Work) it is infeasible



$$f(x) = \frac{1}{1 + e^{-(wx + b)}}$$

lets say we have,

$$(x, y) = (0.5, 0.2) \quad \text{&} \quad (2.5, 0.9)$$

two data points as training data

and at end of training we expect  $w^*$ ,  $b^*$  such that

$$f(0.5) \rightarrow 0.2 \quad \text{&} \quad f(2.5) \rightarrow 0.9$$

→ lets take an initial random guess

$$w = 0.5, b = 0$$

Now, we calculate loss function } difference of predicted value from actual values

$$\begin{aligned} L(w, b) &= \frac{1}{2} \sum_{i=1}^n (y_i - f(x_i))^2 = \frac{1}{2} * (0.9 - f(2.5))^2 \\ &\quad + \frac{1}{2} * (0.2 - f(0.5))^2 \\ &= 0.073 \end{aligned}$$

w	b	L(w, b)
0.5	0	0.0730
-0.10	0	0.1481
0.94	-0.94	0.0214
1.42	-1.73	0.0028
1.65	-2.08	0.0003
1.78	-2.27	0.0000

} next random guess loss increases  
so now we increase w beyond 0.5

} do we keep on making random guesses in direction where loss is decreasing to stop at desired value of loss

This method is not possible if we have large amount of data with lots of features and thus lots of parameters.

## \* Gradient Descent

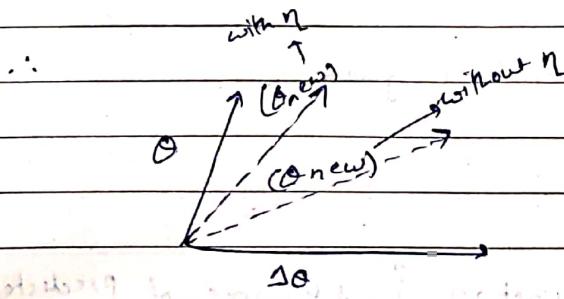
→ vector of parameters randomly initialized

$$\theta = [\omega, b] \in \mathbb{R}^2$$

↳ since two parameters  
it is 2 dimensional vector

Now,  $\Delta\theta = [\Delta\omega, \Delta b]$

↳ change in values of  $\omega, b$



we move  $\theta$  in direction  
of  $\Delta\theta$  by small  
amount  $\eta$  (scalar)

$$\therefore \theta_{\text{new}} = \theta + \eta \cdot \Delta\theta$$

this vector here  
denotes the direction

→ what is direction of  $\Delta\theta$  ?

Ans - direction opposite to that of gradient

lets say  $\Delta\theta = u$   $\therefore$  from Taylor series

$$L(\theta + \eta u) = L(\theta) + \eta * u^T \nabla L(\theta) + \frac{\eta^2}{2!} u^T \nabla^2 L(\theta) u$$

$$+ \frac{\eta^3}{3!} \dots + \frac{\eta^4}{4!} \dots$$

$$= L(\theta) + \eta * u^T \nabla L(\theta)$$

ignoring  
 $\eta^2, \eta^3, \dots$   
because  $\eta$  is  
a small value  
 $\eta^2, \eta^3, \dots$  are close  
to 0

## Formulas

Refer

- Gradient  $\rightarrow \nabla L[\omega, b]$

$$= \begin{bmatrix} \frac{\partial L(\omega, b)}{\partial \omega} \\ \frac{\partial L(\omega, b)}{\partial b} \end{bmatrix}$$

Machine  
Learning  
notes  
for better  
understanding

- Hessian  $\rightarrow \nabla^2 L[\omega, b] = \nabla \begin{bmatrix} \frac{\partial L(\omega, b)}{\partial \omega} \\ \frac{\partial L(\omega, b)}{\partial b} \end{bmatrix}$

$$= \begin{bmatrix} \frac{\partial^2 L(\omega, b)}{\partial \omega^2} & \frac{\partial^2 L(\omega, b)}{\partial \omega \partial b} \\ \frac{\partial^2 L(\omega, b)}{\partial b \partial \omega} & \frac{\partial^2 L(\omega, b)}{\partial b^2} \end{bmatrix}$$

- Taylor Series  $\rightarrow$

$$f(n + \Delta n) = f(n) + \Delta n f'(n) + \frac{(\Delta n)^2}{2!} f''(n) + \frac{(\Delta n)^3}{3!} f'''(n)$$

Now, from obtained equation new loss is obtained which should be  $<$  old loss

$$\therefore L(\theta + \eta u) - L(\theta) < 0 \quad \text{then we say value is correct}$$

$\therefore$  It can be said

$$\eta * u^T \nabla L(\theta) < 0$$

↑ because we wanted to move in direction of  $\Delta \theta$  thus we add the constant

$$\therefore u^T \nabla L(\theta) < 0$$

lets say  $\beta$  is angle between  $u^T$  &  $\nabla L(\theta)$ .

$$-1 \leq \cos\beta = \frac{u^T \nabla L(\theta)}{\|u\| \cdot \|\nabla L(\theta)\|} \leq 1$$

multiply by  $k = \|u\| + \|\nabla L(\theta)\|$  throughout

$$-k \leq k + \cos(\beta) = u^T \nabla L(\theta) \leq k$$

Now we want

$$u^T \nabla L(\theta) < 0$$

for loss to be minimum it should be as -ve as possible

thus  $\beta = 180^\circ$  &  $\cos\beta = -1$  for it to be as -ve as possible

$\therefore u^T$  is in opposite direction of gradient

### → Gradient Descent Rule

- The direction in that we intend to move in should be at  $180^\circ$  w.r.t the gradient
- We move in direction opposite to the gradient

$$\therefore \begin{aligned} \theta_{t+1} &= \theta_t - \eta \nabla L(\theta) \\ b_{t+1} &= b_t - \eta \nabla b_t \end{aligned}$$

Here,

$$\nabla \omega_t = \frac{\partial L(\omega, b)}{\partial \omega} \quad \nabla b = \frac{\partial L(\omega, b)}{\partial b}$$

$$\omega = \omega_L, \quad b = b_L$$

$$\Rightarrow \theta_{t+1} = \theta_t + \eta (-\nabla L(\theta))$$

## → Algorithm

$$t \leftarrow 0$$

$$\text{max\_iterations} \leftarrow 1000;$$

while  $t < \text{max\_iterations}$  do

$$\omega_{t+1} = \omega_t - \eta \nabla \omega_t;$$

$$b_{t+1} = b_t - \eta \nabla b_t;$$

end

## → Calculating $\nabla \omega_t$ & $\nabla b_t$

Initially we assumed  $f(x)$  is sigmoid  
↑ of linear combination

$$\nabla \omega_t = \frac{\partial \frac{1}{2} \sum (y_i - f(x_i))^2}{\partial \omega} = \left[ \sum y_i - f(x_i) \right] \frac{\partial f(x_i)}{\partial \omega} \times 1$$

$$= \sum (y_i - f(x_i)) \times \left( \frac{\partial}{\partial \omega} (\omega x + b) \right)$$

$$= \frac{1}{1 + e^{-(\omega x + b)}} \times (y_i - f(x_i))$$

$$= (y_i - f(x_i)) \times$$

$$\text{calculating } \frac{\partial}{\partial w} \left( \frac{1}{1+e^{-(wx+b)}} \right)$$

$$= -(-e^{-(wx+b)}) \times n \quad \text{by quotient rule}$$

$$= \frac{e^{-(wx+b)}}{(1+e^{-(wx+b)})^2} \times n$$

$$= \frac{1}{1+e^{-(wx+b)}} \times \frac{e^{-(wx+b)}}{1+e^{-(wx+b)}} \times n$$

$$= \frac{1}{1+e^{-(wx+b)}} \times \left( \frac{e^{-(wx+b)}}{1+e^{-(wx+b)}} - 1 + 1 \right) \times n$$

$$= \frac{1}{1+e^{-(wx+b)}} \times \left[ \frac{e^{-(wx+b)} - 1 - e^{-(wx+b)} + 1}{1+e^{-(wx+b)}} \right] \times n$$

$$= \frac{1}{1+e^{-(wx+b)}} \times \left[ 1 - \frac{1}{1+e^{-(wx+b)}} \right] \times n$$

$$\therefore -\sigma'(wx+b) \times (1 - \sigma'(wx+b)) \times n$$

$$\cancel{\therefore f'(n_i) \times (1 - f(n_i)) \times n}$$

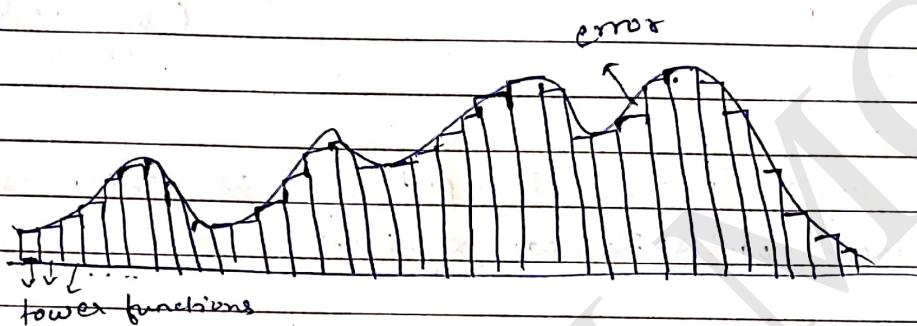
$$\Rightarrow \sum [y_i - f(n_i)] \times f(n_i) \times (1 - f(n_i)) \times n_i$$

$$\nabla b_t = \sum [(y_i - f(n_i)) \times f(n_i) \times (1 - f(n_i))]$$

## \* Representation Power of Multilayer Network of Sigmoid Neurons

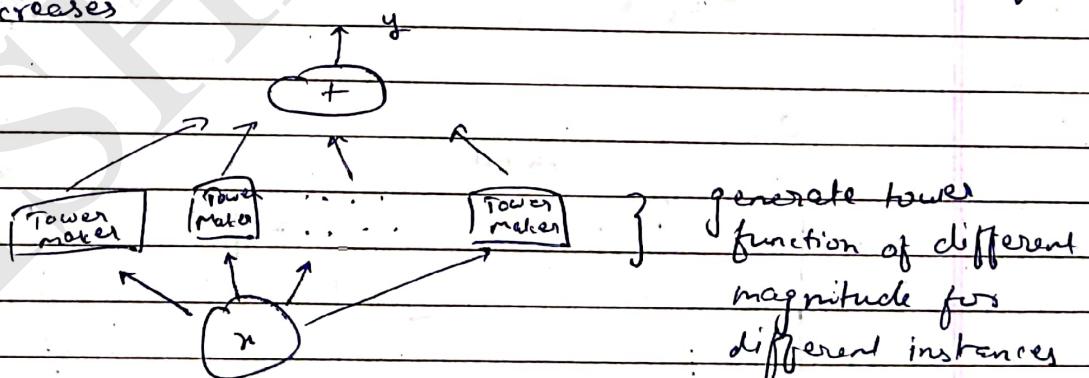
### → Universal Approximation Theorem

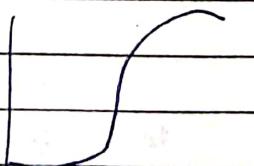
There is a guarantee that for any function  $f(n) : \mathbb{R}^n \rightarrow \mathbb{R}^m$  we can always find a neural network (with 1 hidden layer containing enough neurons) whose output  $g(n)$  satisfies  $|g(n) - f(n)| < \epsilon$  !!



for any function denoted by the curved line above there are tower functions which help us approximate the curve

As the number of tower function increases the amount of error decreases

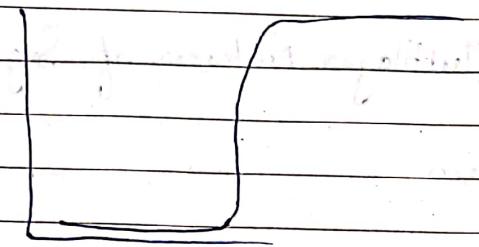


→  in a sigmoid curve when the value of  $w$ ,  $b$  are normal

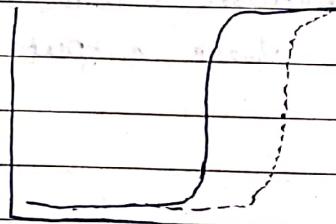
 Sigmoid curve



$$\frac{1}{1 + e^{-(wx + b)}}$$

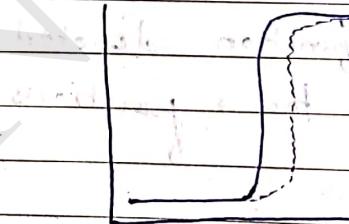
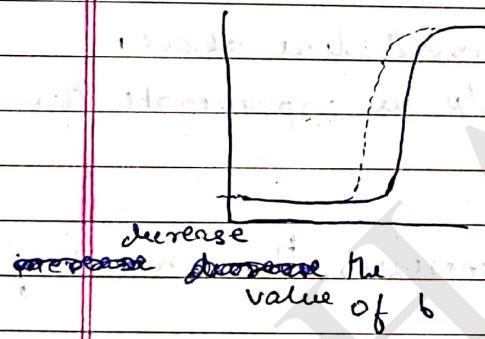


Sigmoid function becomes similar to step function when the value of  $w$  become very high (because  $w$  is the slope)

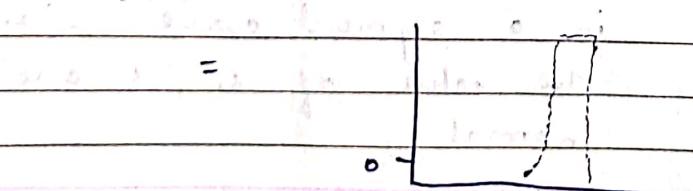
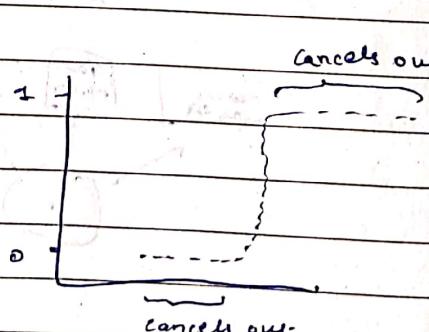
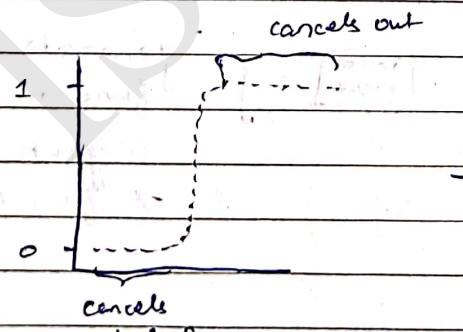


Sigmoid function shifts its position when  $b$  becomes very high or very low (because  $b$  is the intercept)

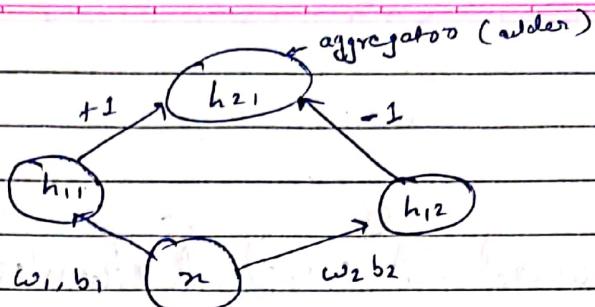
- This is a single dimensional input i.e.,  $w$ , there is only one input parameter  $w$  against  $b$ .



Now



tower function is obtained



Refer → notes / week 3 / network 1D tower maker.png

→ For a case which is 2D. Has parameters  $w_1, w_2$  & bias

Refer → notes / week 3 / tower maker.py

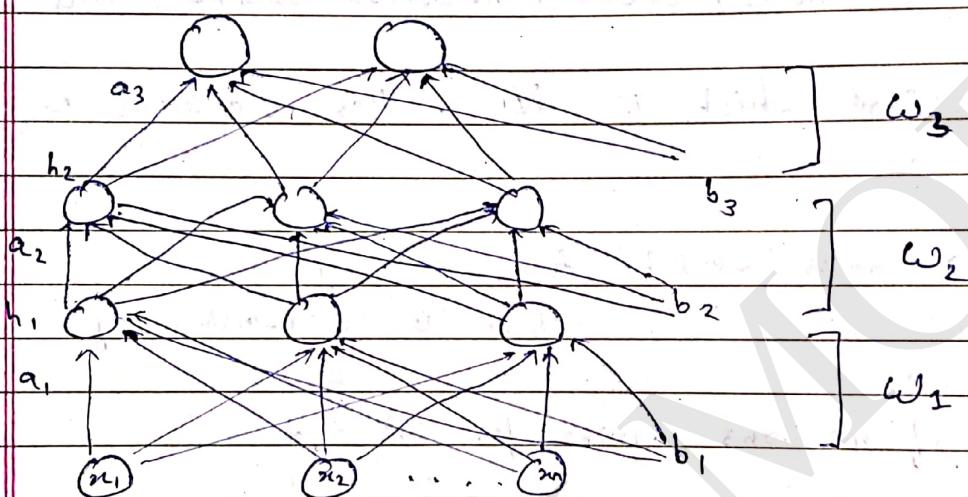
1 " " " " network 2D tower maker.png

→ ∴ No of neurons required for 1D = 2  
2D = 4

∴ No. of neurons for nD = O(2^n)

~~WEEK 4~~

## \* Feed Forward Neural Networks



- Pre-activation at layer  $i$  (aggregation  $\Rightarrow$  pre-activation) in this case

$$a_i(x) = b_i + w_i h_i(x)$$

$$\text{eg } a_1 = b_1 + w_1 h_0$$

$$\begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \end{bmatrix} = \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix} + \begin{bmatrix} w_{111} & w_{112} & w_{113} \\ w_{121} & w_{122} & w_{123} \\ w_{131} & w_{132} & w_{133} \end{bmatrix} \begin{cases} h_{01} = x_1 \\ h_{02} = x_2 \\ h_{03} = x_3 \end{cases}$$

$$= \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix} + \begin{bmatrix} w_{111}x_1 + w_{112}x_2 + w_{113}x_3 \\ w_{121}x_1 + w_{122}x_2 + w_{123}x_3 \\ w_{131}x_1 + w_{132}x_2 + w_{133}x_3 \end{bmatrix}$$

$$= \begin{bmatrix} \sum w_{11i}x_i + b_{11} \\ \sum w_{12i}x_i + b_{12} \\ \sum w_{13i}x_i + b_{13} \end{bmatrix}$$

- Activation Function at i layer

$$h_i(n) = g(a_i(n))$$

$$\begin{bmatrix} h_{i,1} \\ h_{i,2} \\ h_{i,3} \end{bmatrix} = g\left(\begin{bmatrix} a_{i,1} \\ a_{i,2} \\ a_{i,3} \end{bmatrix}\right) = \begin{bmatrix} g(a_{i,1}) \\ g(a_{i,2}) \\ g(a_{i,3}) \end{bmatrix}$$

$g$  can be anything  
e.g.  $\sigma$

- Activation at final layer

$$\therefore f(n) = (h_L(n)) = \sigma(a_L(n))$$

- Now, if

- Data:  $\{x_i, y_i\}_{i=1}^N$

- Model:

$$\hat{y}_i = f(x_i) = \sigma(w_3^T (\omega_2^T g(\omega_1^T x_i + b_1) + b_2) + b_3)$$

- Parameters

$$\Theta = \{\omega_1, \omega_2, \dots, \omega_L, b_1, b_2, \dots, b_L\} \quad \left\{ L = 3 \right\}$$

- Algorithm - Gradient Descent with back prop

- Objective Function

$$\min \frac{1}{N} \sum_{i=1}^N \sum_{j=1}^k (\hat{y}_{ij} - y_{ij})^2$$

no. of data points      predicted output vector

PAGE NO. :  
 DATE :

## \* Learning Parameters Feed Forward N.N

→ In gradient descent

$$\theta \leftarrow [w_1, \dots, w_n, b_1, \dots, b_n]$$

while

$$\theta = \theta - \nabla \theta$$

end

Now,

Here value of  $\nabla \theta$  will be as follows

since  $w_1, \dots, w_n$  are matrix. and  $b_1, \dots, b_n$  are vector

$$\nabla \theta = \begin{bmatrix} \frac{\partial L(\theta)}{\partial w_{11}} & \dots & \frac{\partial L(\theta)}{\partial w_{1n}} & \frac{\partial L(\theta)}{\partial w_{21}} & \dots & \frac{\partial L(\theta)}{\partial w_{2n}} & \dots & \frac{\partial L(\theta)}{\partial w_{L1}} & \dots & \frac{\partial L(\theta)}{\partial w_{Ln}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ \frac{\partial L(\theta)}{\partial w_{1m}} & \dots & \frac{\partial L(\theta)}{\partial w_{nm}} & \frac{\partial L(\theta)}{\partial w_{2m}} & \dots & \frac{\partial L(\theta)}{\partial w_{nm}} & \dots & \frac{\partial L(\theta)}{\partial w_{Lm}} & \dots & \frac{\partial L(\theta)}{\partial w_{nm}} \end{bmatrix}$$

$w_1 \quad w_2 \quad w_L$

$$\begin{bmatrix} \frac{\partial L(\theta)}{\partial b_{11}} & \dots & \frac{\partial L(\theta)}{\partial b_{1L}} \\ \vdots & \ddots & \vdots \\ \frac{\partial L(\theta)}{\partial b_{1n}} & \dots & \frac{\partial L(\theta)}{\partial b_{Ln}} \end{bmatrix}$$

$b_1 \quad b_2 \quad \dots \quad b_L$

$$= - \sum p(x_i) \ln$$

∴ total parameters

$(L-1) (n \times n)$	$w$	$A$	$1 \times (h \times k)$	$w$
$(L-1) (n)$	$b$		$1 \times (k \times 1)$	$b$

Assuming that we multiply  $w^T$  with the input

→ These probability distribution function called softmax

$$\text{softmax}(z) =$$

## \* Output Function & Loss Function

→ Expectation

$$\sum p(x_i) \times v(x_i) = \text{Expectation}$$

Refer notes / week 4 / mean and expectation. png

now for certain outcomes probability is 1 thus information gain is 0. And for values probability with small values information gain is high

$$\text{gain} := I(A) = -\log_2 P(A)$$

Thus expected information

$$\frac{\partial L(\theta)}{\partial b_{11}} \dots \frac{\partial L(\theta)}{\partial b_{L1}} \quad \left. \begin{array}{c} \\ \vdots \\ \end{array} \right\} b_1 \quad = -\sum p(x_i) \log_2 (P(x_i))$$
$$\frac{\partial L(\theta)}{\partial b_{12}} \dots \frac{\partial L(\theta)}{\partial b_{L2}} \quad \left. \begin{array}{c} \\ \vdots \\ \end{array} \right\} b_2 \quad \text{Now lets say,}$$
$$\frac{\partial L(\theta)}{\partial b_{1n}} \dots \frac{\partial L(\theta)}{\partial b_{Ln}} \quad \left. \begin{array}{c} \\ \vdots \\ \end{array} \right\} b_k \quad \text{predicted } \hat{y} = q \quad (\text{probability distribution})$$
$$y = p \quad (\text{actual probability distribution})$$

So, expected information

$$= -\sum p(x_i) \log_2 (q(x_i)) \quad ] \quad \text{This is called cross entropy}$$

→ These probability distributions are obtained by activation function called softmax

$$\text{softmax}(z) = \frac{e^{z_j}}{\sum_{i=1}^k e^{z_i}}$$

∴ We generally use softmax at the last layer

$$a_L = w_L h_{L-1} + b_L$$

$$\hat{y}_j = \sigma(a_L)_j = \frac{e^{a_{L,j}}}{\sum_{i=1}^k e^{a_{L,i}}}$$

e.g.

$$a_L = [a_{11}, a_{12}, a_{13}] \rightarrow \hat{y}_1 = \frac{e^{10}}{e^{10} + e^{-20} + e^{30}}$$

$$\hat{y} = [\hat{y}_1, \hat{y}_2, \hat{y}_3]$$

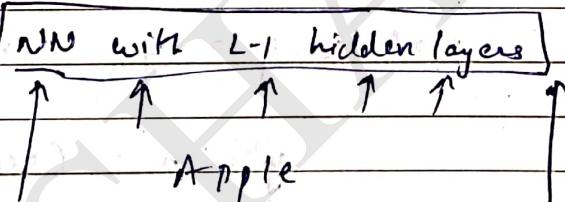
Note →

$$\sum \hat{y}_i = 1 \quad \text{since } \hat{y} \text{ is probability distribution}$$

→ Now in Classification task

$$y = [1 \ 0 \ 0 \ 0]$$

apple mango orange Banana



Here if we use cross entropy

$$-\sum_{c=1}^k y_c \log \hat{y}_c$$

Since one hot vector = original value of  $y$

Thus,

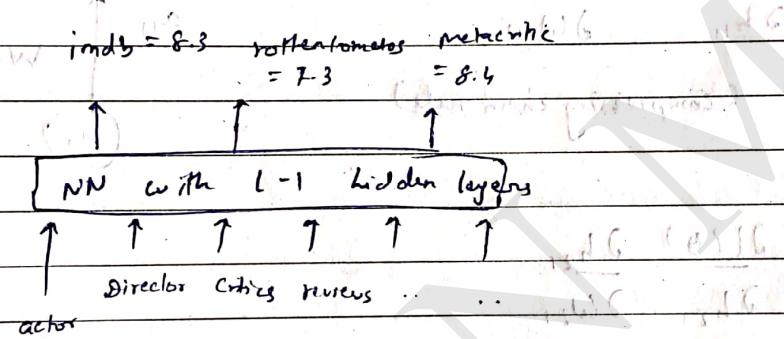
$$-\sum_{c=1}^k y_c \log \hat{y}_c \Rightarrow -\log \hat{y}_{y_c} \quad (\text{if } y_c \text{ is correct classification})$$

Now, our target to make  $\hat{y}_c = 1$  and

so we would minimize the value  $-\log \hat{y}_c$   
because  $-\log \hat{y}_c$  is minimum at  $\hat{y}_c = 1$

so if we obtain minimum of  $-\log \hat{y}_c$  we would obtain  
correct value of  $\hat{y}_c$

→ Now in Regression Task



In this case output function can't be sigmoid because  
it gives value between 0 and 1

so we don't apply sigmoid & straight away let the linear  
function go in output. i.e -  $a$

### Outputs

Real Values      Probabilities

Output Activation

Linear

Softmax

Loss Function

Squared error

Cross Entropy

→

impr

# \* Back Propagation (Intuition)

Normal chain rule

$$\frac{\partial L(\theta)}{\partial w_{111}} = \left( \frac{\partial L(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{111}} \frac{\partial a_{111}}{\partial h_{21}} \frac{\partial h_{21}}{\partial a_{21}} \frac{\partial a_{21}}{\partial h_{111}} \right)$$

$$\times \left( \frac{\partial h_{111}}{\partial a_{111}} \frac{\partial a_{111}}{\partial w_{111}} \right)$$

$$\frac{\partial L(\theta)}{\partial w_{111}} = \frac{\partial L(\theta)}{\partial h_{111}} \times \frac{\partial h_{111}}{\partial w_{111}}$$

(compressing chain rule)

similarly

$$\frac{\partial L(\theta)}{\partial w_{211}} = \frac{\partial L(\theta)}{\partial h_{211}} \frac{\partial h_{211}}{\partial w_{211}}$$

$$\frac{\partial L(\theta)}{\partial w_{L11}} = \frac{\partial L(\theta)}{\partial a_{L11}} \frac{\partial a_{L11}}{\partial w_{L11}}$$

→ Quantities of interest

- Gradient wrt output units
- " " hidden units
- " " weights/biases

$$\frac{\partial L(\theta)}{\partial w_{111}} = \frac{\partial L(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1} \underbrace{\frac{\partial a_1}{\partial h_2}}_{\text{Talk to o/p layer}} \underbrace{\frac{\partial h_2}{\partial a_2}}_{\text{Talk to previous hidden layer}} \underbrace{\frac{\partial a_2}{\partial h_{111}}}_{\text{Talk to previous hidden layer}} \frac{\partial h_{111}}{\partial a_{111}} \underbrace{\frac{\partial a_{111}}{\partial w_{111}}}_{\text{Now talk to activation}}$$

$\frac{\partial L(\theta)}{\partial \hat{y}_l}$  proves that  $L(\theta)$  only depends on  $\hat{y}_l$  since all other terms become zero

DELUXE  
PAGE NO. :  
DATE :  
Cloud Tech

## \* Computing Gradients w.r.t Output Units

→ Let's say  $\hat{y}_k$  is the output vector (classification problem)

$$\frac{\partial L(\theta)}{\partial \hat{y}_{k,i}}$$

$$\text{Loss function} = L(\theta) = -\log(\hat{y}_{k,l}) \quad (l = \text{true class label})$$

$$\text{Now } \frac{\partial L(\theta)}{\partial \hat{y}_{k,i}} = \frac{\partial -\log(\hat{y}_{k,i})}{\partial \hat{y}_{k,i}}$$

$$[\hat{y}_1, \hat{y}_2, \hat{y}_3, \dots, \hat{y}_k]$$

$$= \begin{cases} -\frac{1}{\hat{y}_l} & \text{if } i = l \\ 0 & \text{otherwise} \end{cases}$$

Differentiating loss with each element (i) in  $\hat{y}_k$  vector

↙ can be written as

$\underbrace{\Pi(i=l)}$  } is indicator function gives 1 if  $i=l$  0 otherwise

$$\hat{y}_k$$

$$\nabla_{\hat{y}_k} L(\theta) = \frac{\partial L(\theta)}{\partial \hat{y}_{k,i}} = \frac{\Pi(i=l)}{\hat{y}_l} \quad \left\{ \begin{array}{l} \frac{\partial L(\theta)}{\partial \hat{y}_1} = -\frac{1}{\hat{y}_l} \\ \frac{\partial L(\theta)}{\partial \hat{y}_2} = 0 \\ \vdots \\ \frac{\partial L(\theta)}{\partial \hat{y}_k} = 0 \end{array} \right. \quad \begin{array}{c} \Pi(l=1) \\ \vdots \\ \Pi(l=k) \end{array}$$

$$\rightarrow \frac{\partial L(\theta)}{\partial a_{L_i}} = \frac{\partial L(\theta)}{\partial \hat{y}_l} \times \frac{\partial \hat{y}_l}{\partial a_{L_i}}$$

else  
(one hot vector)

$$= \frac{\partial L(\theta)}{\partial \hat{y}_l} \times \frac{\partial \hat{y}_l}{\partial a_{L_i}}$$

$$= -\frac{1}{\hat{y}_l} \times \frac{\partial \left( \frac{\exp(a_{L_i})}{\sum_{j=1}^k \exp(a_{L_j})} \right)}{\partial a_{L_i}}$$

derivative of softmax

$$\frac{\partial \exp(a_{l,i})}{\sum \exp(a_{l,i})} = \frac{\exp(a_{l,i}) \prod_{(j \neq i)} \exp(a_{l,j})}{\sum \exp(a_{l,i})^2}$$

$$\frac{\partial a_{l,i}}{\sum a_{l,i}}$$

$[a_{l,1}, a_{l,2}, a_{l,3}, \dots, a_{l,n}]$

$$\frac{\exp(a_{l,i}) \exp(a_{l,j})}{\sum \exp(a_{l,i})^2}$$

$$= \prod_{(i \neq l)} \text{softmax}(a_{l,i}) - \text{softmax}(a_{l,l}) \times \text{softmax}(a_{l,i})$$

~~$\prod_{(i \neq l)} \text{softmax}(a_{l,i}) \text{softmax}(y_{l,i}) \text{softmax}(y_{l,j}) \text{softmax}(y_{l,k})$~~

$$= - \frac{1}{y_{l,i}} \prod_{(i \neq l)} \left[ \prod_{(j \neq i)} \text{softmax}(y_{l,j}) - \text{softmax}(y_{l,i}) \times \text{softmax}(y_{l,j}) \right]$$

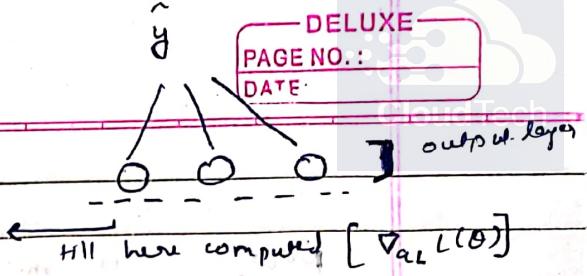
$$= - \prod_{(i \neq l)} \hat{y}_{l,i} - \hat{y}_{l,i} \hat{y}_i$$

$$= - \frac{1}{\hat{y}_{l,i}} \left( \prod_{(i \neq l)} \hat{y}_{l,i} - \hat{y}_{l,i} \hat{y}_i \right)$$

$$= - \left( \prod_{(i \neq l)} \hat{y}_{l,i} - \hat{y}_i \right)$$

$$\nabla_{a_L} = \begin{bmatrix} \frac{\partial L(\theta)}{\partial a_{L,1}} \\ \vdots \\ \frac{\partial L(\theta)}{\partial a_{L,k}} \end{bmatrix} = \begin{bmatrix} - \left( \prod_{(i \neq l)} \hat{y}_{l,i} - \hat{y}_l \right) \\ - \left( \prod_{(i \neq l)} \hat{y}_{l,i} - \hat{y}_2 \right) \\ \vdots \\ - \left( \prod_{(i \neq l)} \hat{y}_{l,i} - \hat{y}_k \right) \end{bmatrix}$$

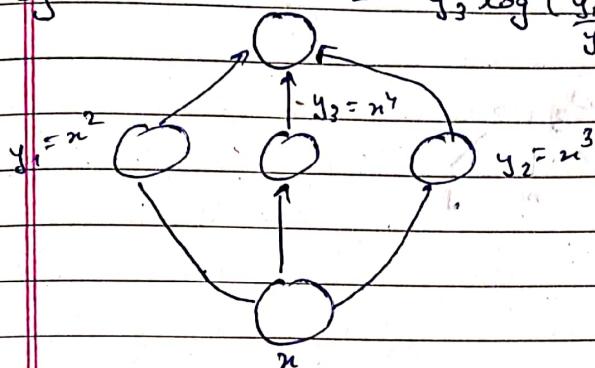
$$= -(e(l) - f(n))$$



## \* Computing Gradients wrt. Hidden Units

eg

$$z = y_3 \log\left(\frac{y_1}{y_2}\right)$$



$$\frac{\partial z}{\partial n} = \sum_{i=1}^3 \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial n}$$

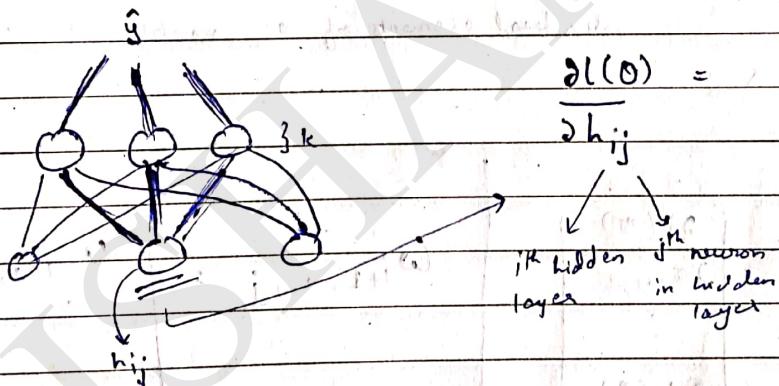
Same concept is used

for N.N (adding chained derivatives)

[Compute all derivatives from path  $z$  to  $n$  & add them]

→ refer backprop notes / weeks / backpropdiagram1.png

→



$$\frac{\partial L(\theta)}{\partial h_{ij}} = \sum_{m=1}^k \frac{\partial L(\theta)}{\partial a_{i+1, m}} \frac{\partial a_{i+1, m}}{\partial h_{ij}}$$

let's say we have only 2 units in output layer

$$a_{i+1, m} = a_{31}, a_{32}$$

$$h_{ij} = h_{21}$$

$$\begin{bmatrix} a_{31} \\ a_{32} \end{bmatrix} = \begin{bmatrix} w_{311} & w_{312} & w_{313} \\ w_{321} & w_{322} & w_{323} \end{bmatrix} \begin{bmatrix} h_{21} \\ h_{22} \\ h_{23} \end{bmatrix} + \begin{bmatrix} b_{31} \\ b_{32} \end{bmatrix}$$

Now,

$$\frac{\partial a_{31}^T}{\partial h_{22}} = \frac{\partial w_{311} h_{21} + w_{312} h_{22} + w_{313} h_{23} + b_3}{\partial h_{22}}$$

$$\frac{\partial h_{22}}{\partial j}$$

$$= \frac{w_{312}}{i+1 \quad m \quad j}$$

~~Now,  $\frac{\partial L(\theta)}{\partial a_{i+1,m}}$~~

$$= \sum_{m=1}^k \frac{\partial L(\theta)}{\partial a_{i+1,m}} \frac{\partial a_{i+1,m}}{\partial h_{ij}}$$

$$= \sum_{m=1}^k \frac{\partial L(\theta)}{\partial a_{i+1,m}} w_{i+1,m,j}$$

↓  
individual elements of a vector.

Now,

$$\nabla_{a_{i+1}} L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial a_{i+1,1}} \\ \vdots \\ \frac{\partial L(\theta)}{\partial a_{i+1,k}} \end{bmatrix} \quad \omega_{i+1,\cdot,j} = \begin{bmatrix} w_{i+1,1,j} \\ \vdots \\ w_{i+1,k,j} \end{bmatrix}$$

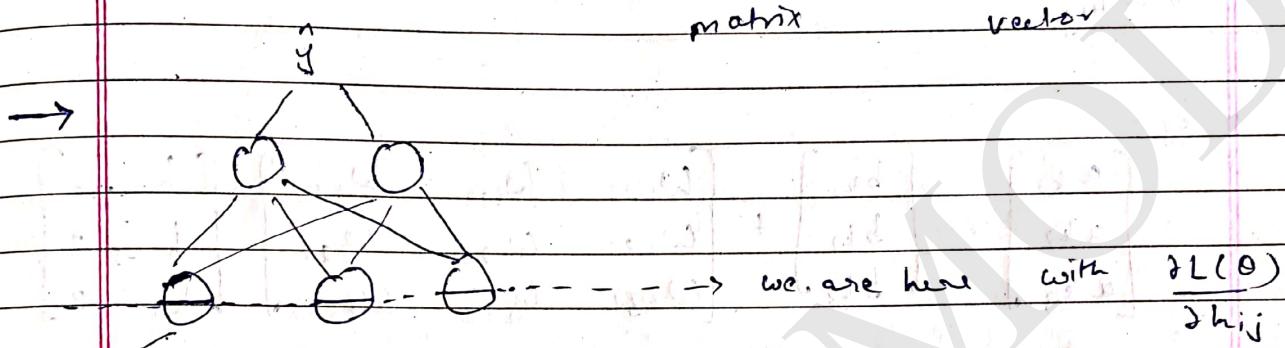
$$\therefore \nabla_{a_{i+1}} L(\theta) = (\omega_{i+1,\cdot,j})^T \nabla_{a_{i+1}} L(\theta)$$

single  
neuron  
of hidden  
layer

$$\nabla_{h_i} L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial h_{i1}} \\ \vdots \\ \frac{\partial L(\theta)}{\partial h_{in}} \end{bmatrix} = \begin{bmatrix} (\omega_{i1}, \dots, 1)^T & \nabla_{a_{i+1}} L(\theta) \\ \vdots & \vdots \\ (\omega_{i+n}, \dots, n)^T & \nabla_{a_{i+n}} L(\theta) \end{bmatrix}$$

matrix      vector

*all hidden neurons*



$$\nabla_{a_i} L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial h_{i1}} g'(a_{i1}) \\ \vdots \\ \frac{\partial L(\theta)}{\partial h_{in}} g'(a_{in}) \end{bmatrix} \quad \text{e.g.}$$

## \* Computing Gradient wrt Parameters

$$a_k = b_k + \omega_k h_{k-1}$$

Weights

$$\frac{\partial a_k}{\partial \omega_{kj}} = h_{k-1, j}$$

$$\frac{\partial a_k}{\partial b_k}$$

e.g -

$$\begin{bmatrix} a_{k_1} \\ a_{k_2} \\ a_{k_3} \end{bmatrix} = \begin{bmatrix} b_{k_1} \\ b_{k_2} \\ b_{k_3} \end{bmatrix} + \begin{bmatrix} \omega_{k_11} & \omega_{k_12} & \omega_{k_13} \\ \omega_{k_21} & \omega_{k_22} & \omega_{k_23} \\ \omega_{k_31} & \omega_{k_32} & \omega_{k_33} \end{bmatrix} \begin{bmatrix} h_{k-1, 1} \\ h_{k-1, 2} \\ h_{k-1, 3} \end{bmatrix}$$

$$\frac{\partial a_{k_1}}{\partial \omega_{k_1j}} = \frac{\partial b_{k_1}}{\partial \omega_{k_1j}} + \omega_{k_11} h_{k-1, 1} + \omega_{k_12} h_{k-1, 2} + \omega_{k_13} h_{k-1, 3}$$

$$\frac{\partial a_{k_1}}{\partial \omega_{k_1j}} = \frac{\partial b_{k_1}}{\partial \omega_{k_1j}}$$

$$= h_{k-1, j}$$

$$\therefore \frac{\partial a_{k_1}}{\partial \omega_{k_1j}} = h_{k-1, j}$$

Now,

$$\frac{\partial L(\theta)}{\partial \omega_{k_1j}} = \frac{\partial L(\theta)}{\partial a_{k_1}} \frac{\partial a_{k_1}}{\partial \omega_{k_1j}} = \frac{\partial L(\theta)}{\partial a_{k_1}} h_{k-1, j}$$

Let's take  $3 \times 3$  matrix as weight for example

$$\nabla_{w_k} L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial w_{k,00}} & \frac{\partial L(\theta)}{\partial w_{k,01}} & \frac{\partial L(\theta)}{\partial w_{k,02}} \\ \frac{\partial L(\theta)}{\partial w_{k,10}} & \frac{\partial L(\theta)}{\partial w_{k,11}} & \frac{\partial L(\theta)}{\partial w_{k,12}} \\ \frac{\partial L(\theta)}{\partial w_{k,20}} & \frac{\partial L(\theta)}{\partial w_{k,21}} & \frac{\partial L(\theta)}{\partial w_{k,22}} \end{bmatrix}$$

$$\frac{\partial L(\theta)}{\partial w_{kij}} = \frac{\partial L(\theta)}{\partial a_{ki}} \frac{\partial a_{ki}}{\partial w_{kij}} = \frac{\partial L(\theta)}{\partial a_{ki}} h_{k-1,jj}$$

$$\rightarrow \nabla_{w_k} L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial a_{k0}} h_{k-1,0} & \frac{\partial L(\theta)}{\partial a_{k0}} h_{k-1,1} & \frac{\partial L(\theta)}{\partial a_{k0}} h_{k-1,2} \\ \frac{\partial L(\theta)}{\partial a_{k1}} h_{k-1,0} & \frac{\partial L(\theta)}{\partial a_{k1}} h_{k-1,1} & \frac{\partial L(\theta)}{\partial a_{k1}} h_{k-1,2} \\ \frac{\partial L(\theta)}{\partial a_{k2}} h_{k-1,0} & \frac{\partial L(\theta)}{\partial a_{k2}} h_{k-1,1} & \frac{\partial L(\theta)}{\partial a_{k2}} h_{k-1,2} \end{bmatrix}$$

$$= \nabla_{a_k} L(\theta) \cdot h_{k-1}^T$$

$(3 \times 1)$        $(2 \times 3)$

↓                  ↓

$(3 \times 3)$

$$a_{ki} = b_{ki} + \sum_j w_{kij} h_{k-1,j}$$

$$\frac{\partial L(\theta)}{\partial b_{ki}} = \frac{\partial L(\theta)}{\partial a_{ki}} \frac{\partial a_{ki}}{\partial b_{ki}} = \frac{\partial L(\theta)}{\partial a_{ki}}$$

$$\nabla_{\theta_L} L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial a_{1,0}} & \frac{\partial L(\theta)}{\partial a_{1,1}} & \dots & \frac{\partial L(\theta)}{\partial a_{1,n}} \end{bmatrix}^T$$

$$= \nabla_{a_L} L(\theta)$$

## \* Back Propagation (Algorithm)

$\rightarrow t \leftarrow 0$

max\_iterations  $\leftarrow 1000$

Initialize  $\theta_0 = [w_0^0, \dots, w_L^0, b_1^0, \dots, b_L^0]$

while  $t++ < \text{max\_iterations}$  do

$h_1, h_2, \dots, h_L, a_1, a_2, \dots, a_L, \hat{y} = \text{forwardprop}(\theta_0);$

$\nabla \theta_t = \text{backward prop}(h_1, h_2, \dots, h_L, a_1, a_2, \dots, a_L, \hat{y});$

$\theta_{t+1} \leftarrow \theta_t - \eta \nabla \theta_t;$

end

forward prop

$\rightarrow$  for  $k = 1$  to  $L-1$  do

$q_k = b_{k,0} + w_k h_{k-1};$

$h_k = g(a_k);$

end

$a_L = b_L + w_L h_{L-1};$

$\hat{y} = \theta(a_L);$

backward prop

$\rightarrow \nabla_{a_L} L(\theta) = -(e(y) - f(x));$  // Compute output gradient

for  $k = 1$  to  $1$  do

$\nabla_{w_k} L(\theta) = \nabla_{a_{k+1}} L(\theta) h_{k-1}^T;$

$\nabla_{b_k} L(\theta) = \nabla_{a_k} L(\theta);$

} // Compute gradients w.r.t parameters

$$\nabla_{h_{k-1}} L(\theta) = w_k^T (\nabla_{a_k} L(\theta)) \quad // \text{Compute gradients with wrt layer below}$$

$$\nabla_{a_{k-1}} L(\theta) = \nabla_{h_{k-1}} L(\theta) \odot [\dots \cdot g'(a_{k-1,j}) \dots];$$

end

## \* Derivative of Activation Function

→ Sigmoid

$$g(z) = \sigma(z) = \frac{1}{1+e^{-z}}$$

$$\begin{aligned} g'(z) &= (-1) \times \frac{1}{(1+e^{-z})^2} \frac{d}{dz}(1+e^{-z}) = -1 \times \frac{1}{(1+e^{-z})^2} \times (-e^{-z}) \\ &= \frac{1}{1+e^{-z}} \left( \frac{1+e^{-z}-1}{1+e^{-z}} \right) \\ &= g(z)(1-g(z)) \end{aligned}$$

→ Tanh

$$g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$g'(z) = \frac{(e^z + e^{-z}) \frac{d}{dz}(e^z - e^{-z}) - (e^z - e^{-z}) \frac{d}{dz}(e^z + e^{-z})}{(e^z + e^{-z})^2}$$

$$= \frac{(e^z + e^{-z})^2 - (e^z - e^{-z})^2}{(e^z + e^{-z})^2}$$

$$= \frac{1 - (e^z - e^{-z})^2}{(e^z + e^{-z})^2} = 1 - (g(z))^2$$

## \* Information Content, Entropy & cross Entropy

$$\rightarrow \text{Information Gain} \propto \frac{1}{P(A)} \quad - \textcircled{1}$$

/content

let  $X$  &  $Y$  be random variables of independent events

$$IG(X \cap Y) = IG(X) + IG(Y)$$

$$f(P(X \cap Y)) = f(P(X)) + f(P(Y)) \quad - \text{from } \textcircled{1}$$

( $\because$  events are independent)

$$(f(P(X).P(Y))) = f(P(X)) + f(P(Y))$$

$$f(a.b) = f(a) + f(b)$$

$$IG(A) = \log\left(\frac{1}{P(A)}\right) = -\log P(A)$$

$\rightarrow$  Entropy = Expected information content

$$= -\sum_{i \in A, B, C, D} p(x=i) \log P(x=i)$$

$\rightarrow$  Cross Entropy

$\times$  bits required

$$A \rightarrow \frac{1}{4} \quad 00 \quad IG(A) = -\log_2 \frac{1}{4} = 2 \text{ bits required}$$

$$IG(B) = " = 2$$

$$B \rightarrow \frac{1}{4} \quad 01 \quad IG(C) = " = 2$$

$$IG(D) = " = 2$$

$$C \rightarrow \frac{1}{4} \quad 10$$

$$D \rightarrow \frac{1}{4} \quad 11$$

$$\text{Avg bits required} = 2 \quad \} \text{Entropy}$$

Similarly for 8 variables

$$IG(A, B, C, D, \dots, H) = -\log_2 \frac{1}{8} = 3 \text{ bits required}$$

Now if,

X

$$\begin{array}{ll} A \rightarrow \frac{1}{2} & IG(A) = 1 \\ B \rightarrow \frac{1}{4}, & IG(B) = 2 \\ C \rightarrow \frac{1}{8} & IG(C) = 3 \\ D \rightarrow \frac{1}{8} & IG(D) = 3 \end{array} \quad \left. \begin{array}{l} \text{less bits required} \\ \text{for more frequently} \\ \text{occurring event} \end{array} \right\}$$

$$\text{Avg bits required} = 1.75 \quad \left. \begin{array}{l} \text{Entropy} \end{array} \right\}$$

Thus

Cross Entropy

$$-\sum p_i \log_2 q_i \quad \xrightarrow{\text{estimated distribution}}$$

$\downarrow$  bits required  
it depends on the actual distribution  
for estimated distribution

Now minimize

$$-\sum p_i \log_2 q_i \quad \left. \begin{array}{l} \text{(take partial derivative w.r.t } q_i \\ \text{ & equate to 0) } \end{array} \right.$$

constraint  
to follow during  
optimization  
such that  $\sum q_i = 1$

$\rightarrow$  replace multiplier

$$\therefore -\sum p_i \log_2 q_i + (\lambda (\sum q_i - 1)) \quad \left. \begin{array}{l} \text{minimum at } q_i = 1 \end{array} \right.$$

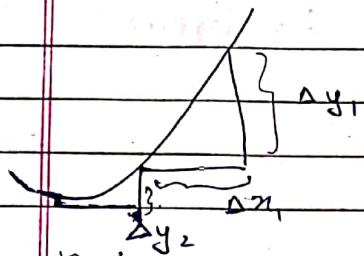
equating partial derivative to zero  $\left( \frac{-p_i}{q_i} + \lambda = 0 \right)$

$$\therefore p_i = \lambda q_i$$

$\downarrow$

$$p_i = q_i \quad (\text{if } \lambda = 1)$$

~~WEEK 5~~



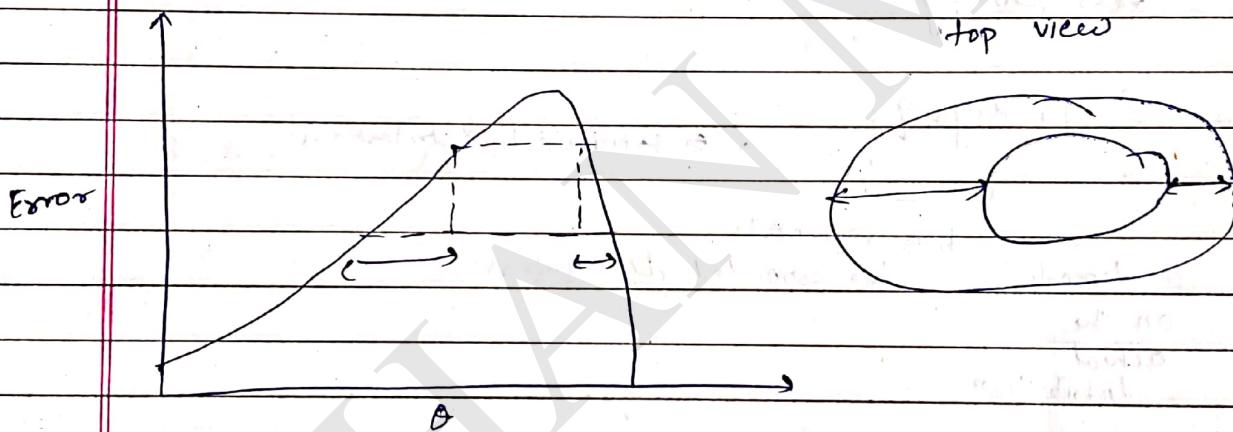
Now whenever there is  
steep slope derivative is high

gentle  
Whenever shallow slope derivative is low

$$\text{eg } \frac{\Delta y_1}{\Delta x_1} = \text{low} \quad \text{and} \quad \frac{\Delta y_2}{\Delta x_2} = \text{high}$$

gradient is similar to these derivatives

## \* Contour Maps



- A small distance between the contours indicates a steep slope along that direction
- A large distance between the contours indicates a gentle slope along that direction

Refer notes /week 5/ contour 1.png  
Contour 2.png  
Contour 3.png

## \* Momentum Based Gradient Descent

$$\text{update}_t = \gamma \text{update}_{t-1} + \eta \nabla w_t$$

$$w_{t+1} = w_t - \text{update}_t$$

working

$$\text{update}_0 = 0 \quad \} \text{ initially}$$

$$\text{update}_1 = \gamma \cdot \text{update}_0 + \eta \nabla w_1 = \eta \nabla w_1$$

$$\text{update}_2 = \gamma \cdot \text{update}_1 + \eta \nabla w_2 = \gamma \cdot \eta \nabla w_1 + \eta \nabla w_2$$

$$\begin{aligned} \text{update}_3 &= \gamma \cdot \text{update}_2 + \eta \nabla w_3 = \gamma(\gamma \cdot \eta \nabla w_1 + \eta \nabla w_2) + \eta \nabla w_3 \\ &= \gamma^2 \cdot \eta \nabla w_1 + \gamma \cdot \eta \nabla w_2 + \eta \nabla w_3 \end{aligned}$$

$$\text{update}_4 = \gamma \cdot \text{update}_3 + \eta \nabla w_4 = \gamma^3 \cdot \eta \nabla w_1 + \gamma^2 \eta \nabla w_2 + \gamma \eta \nabla w_3 + \eta \nabla w_4$$

exponentially weighted average  
 $\gamma < 1$

$$\begin{aligned} \therefore \text{update}_t &= \gamma \cdot \text{update}_{t-1} + \eta \nabla w_t \\ &= \gamma^{t-1} \cdot \eta \nabla w_1 + \gamma^{t-2} \cdot \eta \nabla w_2 + \dots + \eta \nabla w_t \end{aligned}$$

→ ~~Visualization of MBGD on 3D error surface~~  
(U-turns that it takes and changes in sigmoid curve)

The main problem is oscillations when it reaches the valley of error surface. But eventually they stop oscillations decrease. It is better than normal gradient descent

## \* Nesterov's Gradient Descent

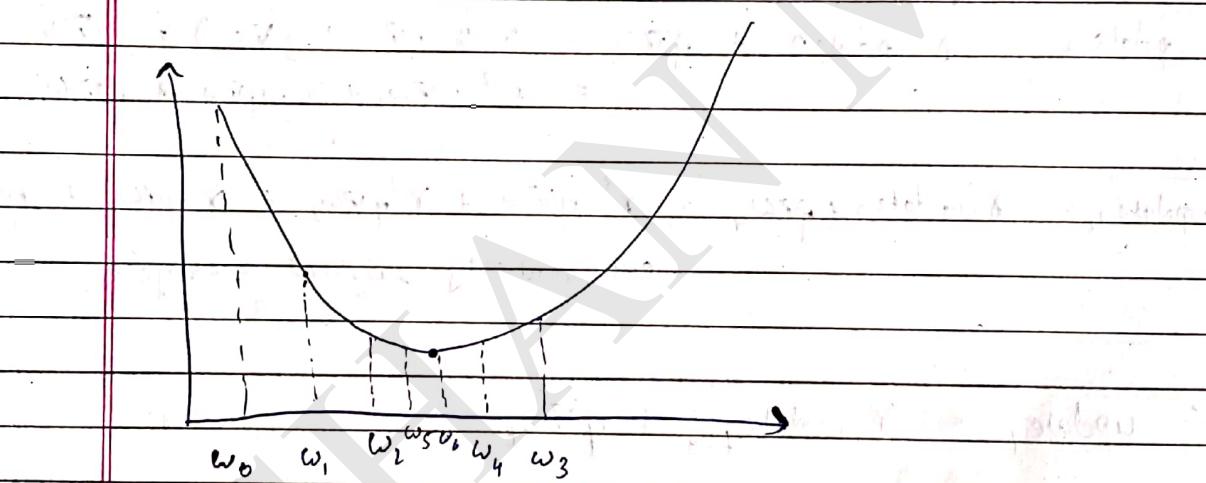
$$\text{Momentum based GD} \rightarrow w_{t+1} = \underbrace{\left( w_t - \frac{\text{update}_{t-1}}{\text{history}} \right)}_{\text{w\_look\_ahead}} - \frac{\eta \nabla w_t}{\text{gradient}}$$

$$w_{\text{look\_ahead}} = w_t - \gamma \cdot \text{update}_{t-1}$$

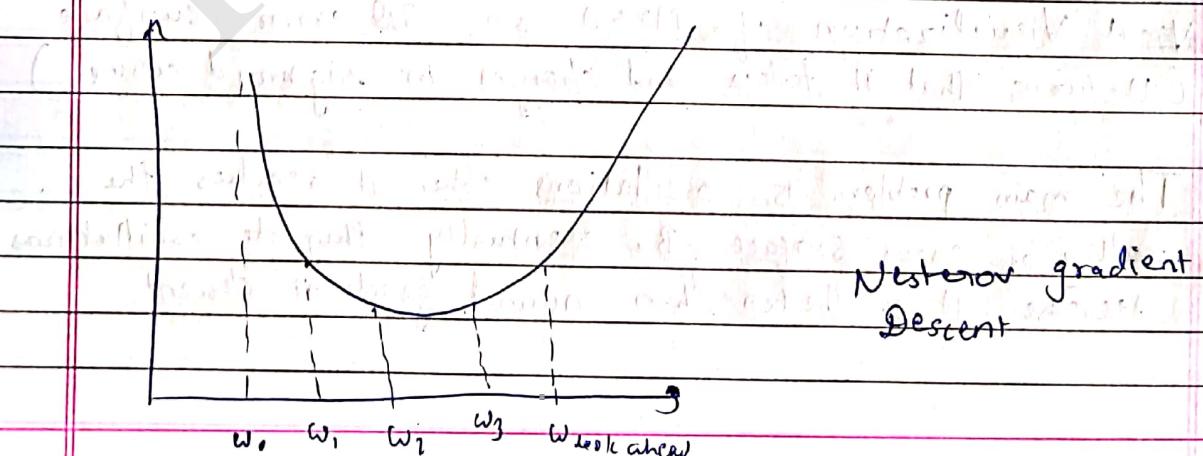
$$\text{update}_t = \gamma \cdot \text{update}_{t-1} + \eta \nabla w_{\text{look\_ahead}}$$

$$w_{t+1} = w_t - \text{update}_t$$

Explanation



This is momentum based gradient descent



Basic idea of Nesterov's GD is that it finds  $w_{\text{lookahead}}$  of the history of gradients before current  $w_t$ .

Then find the gradient w.r.t  $w_{\text{lookahead}}$  and calculate update and then update  $w_t$  with update.

## \* Stochastic & Minibatch Gradient Descent

→ Normal Gradient Descent (Batch) [NGD]

Passing over entire data  
adding gradients of individual inputs  
then updating weights after every pass

→ Stochastic GD [SGD]

Passing over entire data  
updating weights using gradient of individual inputs after every input

→ MiniBatch GD [MBGD]

Passing over entire data  
adding gradients of every  $n^{batch \ size}$  inputs then updating the weights.

JMP  
Notes

Algorithm

No. of steps in 1 epoch

- 1 epoch = one pass over entire data
- 1 step = one update of the parameters
- $N$  = number of data points
- $B$  = mini batch size

NGD

1

SGD

$N$

MBGD

$N/B$

Convex loss function has only one minima  
Non-convex loss function has several minima

DELUXE  
PAGE NO.:  
DATE  
CloudTech

## \* Adaptive Learning Rate

### → Tuning

Try out on log scale 0.1, 0.01, 0.001 etc  
calculate loss wrt above learning rate for limited epochs

pick learning rate that gives minimum loss

Try learning rate around the obtained eg 0.1, 0.2, 0.3, 0.4 etc  
↓ obtained

### → Step Decay

- Half the learning rate after limited epoch
- Half the learning rate after an epoch if the validation error is more than what it was at the end of previous epoch.

### → Exponential Decay

$$[\eta = \eta_0 e^{-kt}] \text{ after every epoch update } \eta \text{ is updated}$$

here I have assumed  $t=1$

if no. of epoch = 2 after which update  $\eta$  takes place then  $t=2$

### → $\frac{1}{t}$ Decay

$$\eta = \frac{\eta_0}{1+kt} \quad k \text{ is hyperparameter}$$

$t$  is step size



## \* Momentum

$$\rightarrow \gamma_t = \min \left( 1 - 2^{-1 - \log_2 (\lceil t/250 \rceil + 1)}, \mu_{\max} \right)$$

$$\mu_{\max} = \{0.999, 0.995, 0.99, 0.9, 0\}$$

$$\gamma_0 \approx 0.5$$

$$\gamma_{250} \approx 0.75$$

$$\gamma_{750} \approx 0.875$$

As no. of steps increases the dependency on the history increases

## \* Line Search

→ At every epoch try n no. of learning rates  
eg let's say n=5

$$\{0.01, 0.1, 0.5, 5, 10\}$$

Now we calculate loss wrt all learning rate & consider learning rate with minimum loss

: For every # epoch different learning rate are chosen based on error surface

if surface is gentle higher learning rate

if surface is steep lower learning rate

## \* Gradient Descent with Adaptive Learning Rate

$$\nabla w^1 = (f(n) - y) * f(n) * (1 - f(n)) + x_1^*$$

$$\nabla w^2 = (f(n) - y) * f(n) * (1 - f(n)) + x_2^*$$

here we are not talking about the input, but feature (column) of the entire set of input

Now,

if  $n_2$  is a sparse feature ie it contains lot of 0's than

the total gradient would be less than gradients of other attributes, since it is updated less no. of times

Thus to bring the gradient of this attribute at the level of other gradients. The learning rate of this attribute has to be increased

## → Adagrad

Decay the learning rate for parameters in proportion to their update history (more update history more decay)

$$v_t = v_{t-1} + (\nabla w_t)^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} * \nabla w_t$$

initially  $v_t = 0$

This  $v_t$  is calculated w.r.t all gradients / elements in  $\nabla w_t$

Thus size of  $v_t$  = size of  $\nabla w_t$  matrix

→ RMS prop parameters corresponding to inputs  
 $b$  here means the parameters updated more no. of times

In adagrad over time the effective learning rate for  $b$  will decay to an extent that there will be no further updates to  $b$ . So proper exact convergence won't be obtained thus we use RMS prop

$$V_t = \beta V_{t-1} + (1-\beta) * (\nabla w_t)^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{V_t + \epsilon}} + \nabla w_t$$

Denominator grows very slowly

→ Adam

$$m_t = \beta_1 * m_{t-1} + (1-\beta_1) * \nabla w_t \quad \left. \right\} \text{similar to momentum based GD}$$

$$v_t = \beta_2 * v_{t-1} + (1-\beta_2) * (\nabla w_t)^2 \quad \left. \right\} \text{similar to Adagrad/RMS prop}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} - \quad \textcircled{1}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} + \hat{m}_t$$

• Bias Correction (obtaining statement  $\textcircled{1}$ )

When we perform normal gradient descent the target is that we want to find new weight with respect to current gradient

But for other variations we take decaying running exponential average of all the gradients. Thus by computing this what we are trying to do is.

$$\boxed{m_t} \leftarrow \beta * m_{t-1} + (1-\beta) \nabla w_t$$

Now, what we want is to change the total weight by average of all gradients in history that is  $E[\nabla w_t]$

$\therefore$  Technically in above equation to obtain  $E[\nabla w_t]$  we need  $E[m_t]$

& if  $E[m_t] = E[\nabla w_t]$  condition satisfy we have correct solution

#### • Derivation

lets say  $\nabla w_t = g_t$

$$m_t = \beta * m_{t-1} + (1-\beta) + g_t$$

$$m_0 = 0$$

$$m_1 = \beta m_0 + (1-\beta) g_1 = (1-\beta) g_1$$

$$m_2 = \beta m_1 + (1-\beta) g_2 = \beta(1-\beta) g_1 + (1-\beta) g_2$$

$$m_3 = \beta m_2 + (1-\beta) g_3 = \beta(\beta(1-\beta)) g_1 + \beta(1-\beta) g_2 + (1-\beta) g_3$$

$\therefore$  General

$$m_t = (1-\beta) \sum_{i=1}^t \beta^{t-i} g_i$$

We need  $E[m_t]$

$$E[m_t] = E[(1-\beta) \sum_{i=1}^t \beta^{t-i} g_i]$$

$$= (1-\beta) \cdot E \left[ \sum_{i=1}^t \beta^{t-i} g_i \right]$$

$$= (1-\beta) \sum_{i=1}^t \beta^{t-i} E[g_i]$$

Assume all  $g_i$ 's come from same distribution  $\therefore E[g_i] = E[g]$

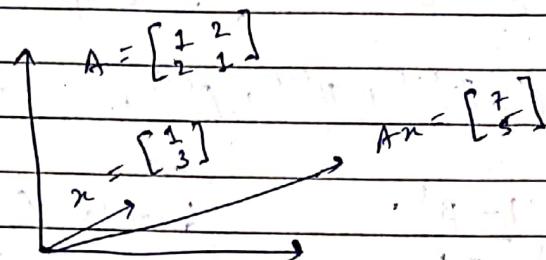
$$\begin{aligned}
 E[m_t] &= (1-\beta) \sum_{i=1}^t \beta^{t-i} E[g_i] \\
 &= E[g] (1-\beta) \sum_{i=1}^t \beta^{t-i} \quad \text{is a GP} \\
 &= E[g] (1-\beta) (1 + \beta^{t-1} + \beta^{t-2} + \beta^{t-3} + \dots + \beta^0) \\
 &= E[g] (1-\beta) \left( \frac{1-\beta^t}{1-\beta} \right) \quad \begin{array}{l} \text{formula } \left[ \frac{1-r^n}{1-r} \right] \\ \text{sum of GP} \end{array} \\
 \therefore E[m_t] &= E[g] (1-\beta^t)
 \end{aligned}$$

$$\begin{aligned}
 E\left[\frac{m_t}{1-\beta^t}\right] &= E[g] \\
 E[\hat{m}_t] &= E[g] \quad \left( \because \hat{m}_t = \frac{m_t}{1-\beta^t} \right)
 \end{aligned}$$

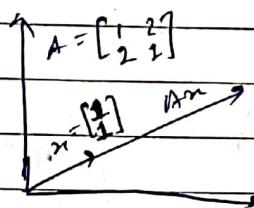
similar for  $\hat{v}_t$

~~WEEK 6~~

## \* Eigen Values & Eigen Vectors



When we multiply a matrix with normal vector its direction and magnitude changes



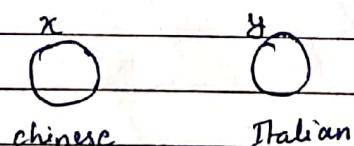
When we multiply a matrix with Eigen-vector its direction doesn't change, magnitude may change.

$$Ax = \lambda x$$

↓      ↓  
Eigen      Eigen  
value      vector

[direction remains same]

→ Let's consider an example



$$\text{initially } \begin{bmatrix} x \\ y \end{bmatrix} = v_0$$

for first timestamp

Next day p fraction continue to eat chinese  
q fraction " " " italan

$$px + (1-q)y \quad (1-p)x + qy$$



chinese

$$v_1 = \begin{bmatrix} px + (1-q)y \\ (1-p)x + qy \end{bmatrix} = \begin{bmatrix} p & 1-q \\ 1-p & q \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Similarly

$$v_2 = \begin{bmatrix} p & 1-q \\ 1-p & q \end{bmatrix} \left( \begin{bmatrix} p & 1-q \\ 1-p & q \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \right)$$

$$\therefore v_n = M^n v_0$$

$\downarrow \quad \rightarrow$

$$\begin{bmatrix} p & 1-q \\ 1-p & q \end{bmatrix}^n \begin{bmatrix} x \\ y \end{bmatrix}$$

- Dominant eigen vectors & values

If  $\lambda_1, \lambda_2, \dots, \lambda_n$  are eigen values of  $n \times n$  matrix A  
then  $\lambda_1$  is called dominant eigen value if

$$|\lambda_1| > |\lambda_i| \quad i=2, \dots, n$$

A corresponding vector of  $\lambda_1$  is called eigen vector

- Matrix M is stochastic matrix if all entries are positive & sum of all elements in each column is 1.

Theorem

- The dominant eigen value of stochastic matrix is 1

Theorem

- If sequence of matrix

$$A v_0, A^2 v_0, A^3 v_0, \dots, A^n v_0, \dots$$

at some point n

$$A^n v_0 = k \times \text{dominant eigen vector}$$

Now

$$v_n = M^n v_0 = k \times \text{dominant eigen vector (e_d)}$$

$$v_{n+1} = M v_n = M k e_d$$

Now,

$$M e_d = \lambda_d e_d$$

dominant eigen value      dominant eigen vector

$$\therefore = k M e_d = k \lambda_d e_d$$

(1) If M is stochastic matrix

$$k \lambda_d e_d = k e_d \quad \left. \begin{array}{l} \parallel \\ 2 \end{array} \right\} \therefore v_n = v_{n+1}$$

Stable after some time

(2) If M is normal square matrix

$$v_{n+1} = k \lambda_d e_d$$

$$v_{n+2} = k \lambda_d^2 e_d$$

$$v_{n+m} = k \lambda_d^m e_d$$

- If  $|\lambda_d| = 1$  stable
- $|\lambda_d| < 1$  vanishing
- $|\lambda_d| > 1$  exploding

## \* Linear Algebra

→ Basis - for a set of vectors  $\in \mathbb{R}^n$  if each vector can be expressed as linear combination of these vectors and these vectors are linearly independent then they are called basis. eg for  $\mathbb{R}^3$   $(1, 0, 0), (0, 1, 0), (0, 0, 1)$

→ Linearly independent - If for a set of vectors ~~one~~ vector in the set ~~can~~ be expressed as linear combination of other vectors in that set.

$$c_1 v_1 + c_2 v_2 + \dots + c_n v_n = 0 \quad (c_1, c_2, c_3, \dots, c_n \text{ are scalars})$$

→ Let's consider  $\mathbb{R}^2$

basis  $\rightarrow (1, 0) \text{ and } (0, 1)$

or  $(2, 3) \text{ and } (5, 7)$  etc

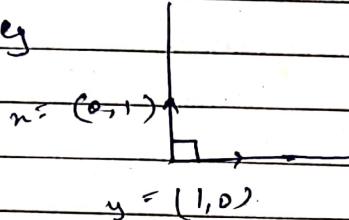
$$(x, y) = a_1 [2, 3] + a_2 [5, 7]$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 & 5 \\ 3 & 7 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \Rightarrow \begin{array}{l} x = 2a_1 + 5a_2 \\ y = 3a_1 + 7a_2 \end{array}$$

can be solved  
by elimination

(independent)  
 But if these vectors in basis are orthonormal  
 ie  $90^\circ$  angle between them (orthogonal)  
 unit vectors (normal)

e.g.



$\{0, 1\}$  &  $\{1, 0\}$  are orthonormal

many more such vectors can be formed without using 0 & 1 abs.

$$\text{Now } \mathbf{x} \cdot \mathbf{y} = \cos 90^\circ \times 1 \times 1 = 0 \times 1 \times 1$$

- (1)

$$\text{Also } \mathbf{x} \cdot \mathbf{x} = \cos 0^\circ \times 1 \times 1 = 1 \quad - (2)$$

for  $n$  dimension ( $\mathbf{z} = nx1$  vector)

$$\mathbf{z} = \alpha_1 \mathbf{u}_1 + \alpha_2 \mathbf{u}_2 + \alpha_3 \mathbf{u}_3 + \dots + \alpha_n \mathbf{u}_n$$

are the ~~vectors~~ orthonormal basis vectors

$$\mathbf{u}_1^T \mathbf{z} = \underbrace{\alpha_1}_{\text{dot product}} \mathbf{u}_1^T \mathbf{u}_1 + \underbrace{\alpha_2}_{\text{from (1)}} \mathbf{u}_1^T \mathbf{u}_2 + \dots + \underbrace{\alpha_n}_{\text{from (2)}} \mathbf{u}_1^T \mathbf{u}_n$$

$$\text{Scalar value} = \alpha_1 \times 1 + 0 + 0 + \dots + 0$$

$$\text{Scalar value} = \alpha_1$$

Similarly we can find  $\alpha_2, \alpha_3, \dots, \alpha_n$

~~Note~~ Now in case of elimination complexity is  $O(n^3)$   
 but for above it is  $O(n^2)$

Theorem

→ Eigen <sup>vectors</sup> of matrix are independent if eigen values are distinct

Theorem

→ The eigen vectors of square symmetric matrix are orthogonal

## \* Eigen Value Decomposition

Let  $u_1, u_2, \dots, u_n$  be eigen vectors. And  
 $\lambda_1, \lambda_2, \dots, \lambda_n$  be eigen values

$$Au = A \begin{bmatrix} u_1 & u_2 & \dots & u_n \end{bmatrix} = \begin{bmatrix} \lambda_1 u_1 & \lambda_2 u_2 & \dots & \lambda_n u_n \end{bmatrix}$$

↙  
collection  
of eigen vectors

$$= \begin{bmatrix} \lambda_1 u_1 & \lambda_2 u_2 & \dots & \lambda_n u_n \end{bmatrix}$$

$$= \begin{bmatrix} u_1 & u_2 & \dots & u_n \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{bmatrix}$$

$$= U \times \Lambda$$

Let's say if  $u^{-1}$  exists

$$A = U \lambda U^{-1} \quad [\text{eigen value decomposition}]$$

$$U^{-1} A U = \lambda \quad [\text{diagonalization of matrix}]$$

→ If  $A$  is square symmetric then as per theorem its eigen vectors are orthogonal

$$\therefore Q = U^T U = \begin{bmatrix} \leftarrow u_1 \rightarrow \\ \leftarrow u_2 \rightarrow \\ \vdots \\ \leftarrow u_n \rightarrow \end{bmatrix} \begin{bmatrix} \uparrow & \uparrow & \uparrow \\ u_1 & u_2 & \dots & u_n \end{bmatrix}$$

$$u_i^T u_j = 0 \text{ if } i \neq j$$

$$= 1 \text{ if } i = j$$

$$\therefore U^T U = \text{Identity} \quad \therefore \underline{\underline{U^T = U^{-1}}}$$

Thus

$$\text{eigen value decomposition} = A = U \Sigma U^T$$

$\downarrow$   
Diagonal matrix

for sequence  $x_0, Ax_0, A^2x_0, \dots$

$n^{\text{th}}$  entry  $\boxed{U \Sigma^n U^T x_0 = A^n x_0}$

Theorem

$$\rightarrow \max_n (n^T A n) \quad \text{such that } \|n\| = 1$$

Here  $n$  will be dominant eigen vector

Theorem →  $\min_n (n^T A n) \quad \text{such that } \|n\| = 1$

Here  $n$  will be smallest eigen vector of  $A$