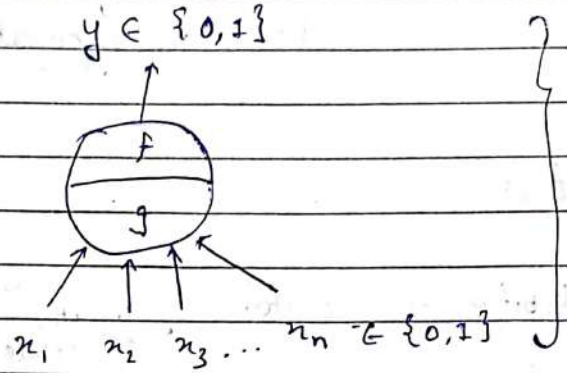
DEEP LEARNING

WEEK 1

(Brief discussion on history of deep learning)

* Artificial Neuron

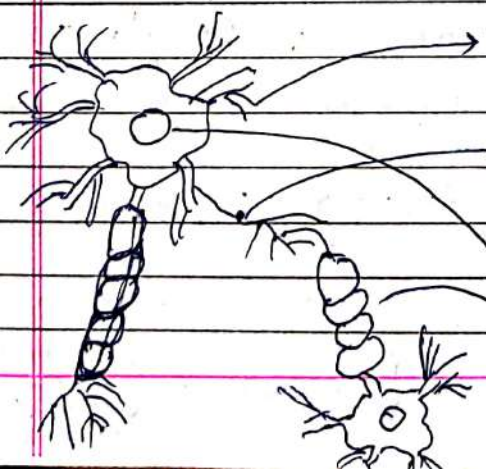


* Image Net challenge architectures

	Network	Error	Layers
(1)	Alex Net	16 %	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

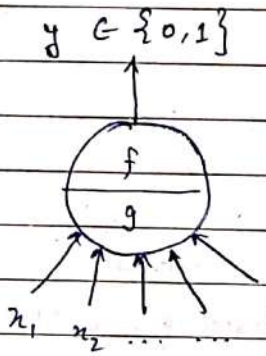
* Biological Neuron



* dendrite - recieve 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 & function f takes decision based on aggregation

- The inputs can be inhibitory or excitator → 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 input is 1 output of neuron is 0)

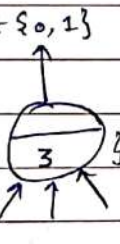
- eg $g(x_1, x_2, \ldots x_n) = g(x) = \sum_{i=1}^{n} x_i$

$$y = f(g(x)) = 1 \quad \text{if } g(x) \geq \theta$$
$$= 0 \quad \text{if } g(x) < \theta$$

θ is thresholding parameter

- 

MP Neuron

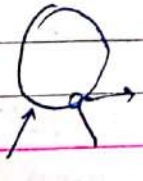


AND

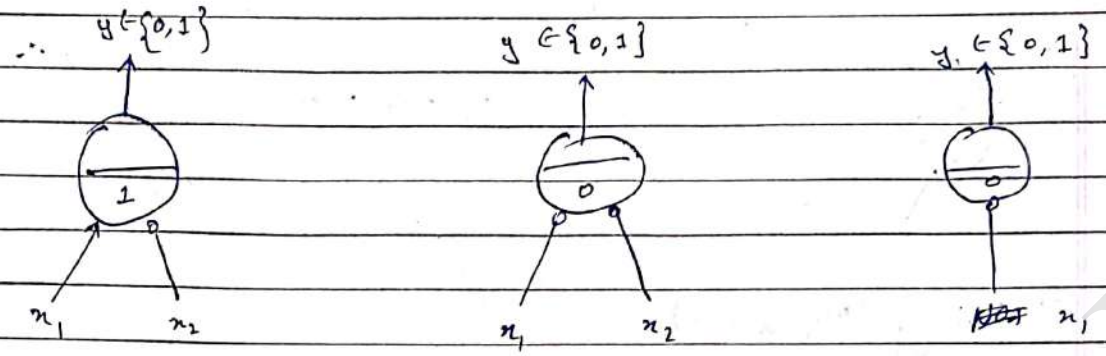
OR

→ at least one of inputs is 1 then output is 1
$(x_1 + x_2 + x_3) \geq 1$ then o/p is 1

→ All three inputs should be 1 for output to be 1
$(x_1 + x_2 + x_3) \geq 3$ then o/p = 1

- 

Here in below example
$(x_1 + x_2) \geq 1$
$(x_1 + x_2) \geq 1$
$x_1 \geq 0$
} threshold all obtained if inhibitory input is considered [≥1]

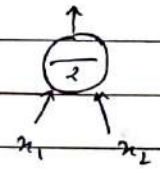


x_1 AND $!x_2$ | NOR | NOT

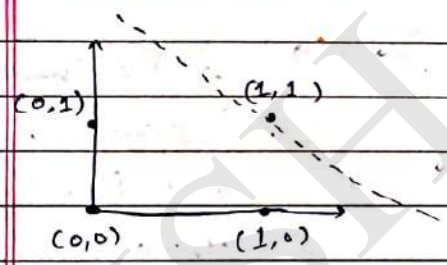
$(x_1 + x_2) \equiv 1$	$(x_1 + x_2) \equiv 0$	$x_1 \equiv 0$
then output = 1	o/p = 1	o/p = 1
(because inhibitory input)	All inputs are inhibit they	

→ Geometric Interpretation (2D)

- AND →



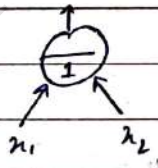
$\therefore x_1 + x_2 \geq 2$



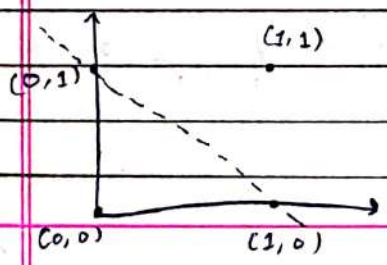
All points on or above the line have output 1

All points below the line have output 0

- OR →



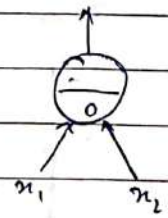
$\therefore x_1 + x_2 \geq 1$



All points on or above
o/p = 1

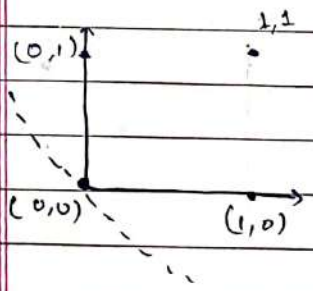
All points below
o/p = 0

≤1]



$x_1 + x_2 \geq 0$

Tautology (Always ON)

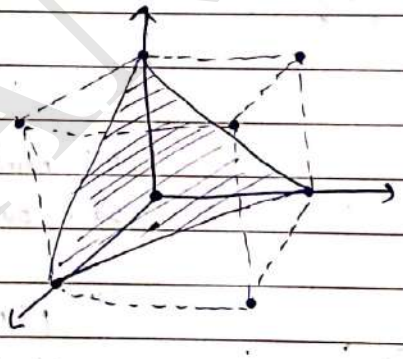


All points on or above
o/p = 1

→ Geometric Implementation (3D)

OR

$x_1 + x_2 + x_3 \geq 1$

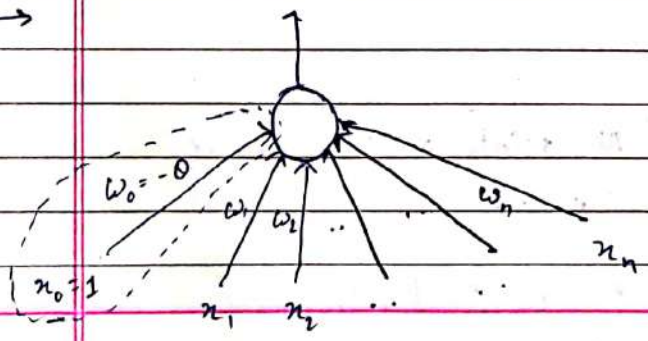


(0, 0, 0) → o/p = 0
else → o/p = 1

∴ The shaded plane seperates point (0,0,0) from others

* Perceptron

→



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

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

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

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

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

here $w_0 = -\theta$, $x_0 = 1$

→

x_1	x_2	OR	
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	$w_0 + \sum w_i x_i \geq 0$
1	1	1	$w_0 + \sum w_i x_i \geq 0$

$\therefore$ $w_0 + 0 \cdot w_1 + 0 \cdot w_2 < 0$ (x_1, x_2)

$\Rightarrow w_0 < 0 \longrightarrow (1)$

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

$\Rightarrow w_2 > -w_0 \longrightarrow (2)$

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

$\Rightarrow w_1 > -w_0 \longrightarrow (3)$

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

$\Rightarrow w_1 + w_2 > -w_0 \longrightarrow (4)$

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

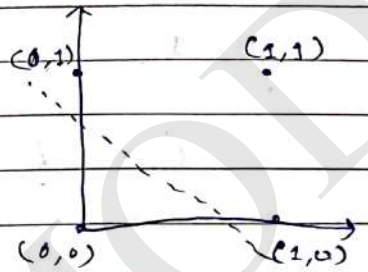
$w_0 = -1$, $w_1 = 1.1$, $w_2 = 1.1$

∴ line will be

$w_0 + w_1 x_1 + w_2 x_2 = 0$

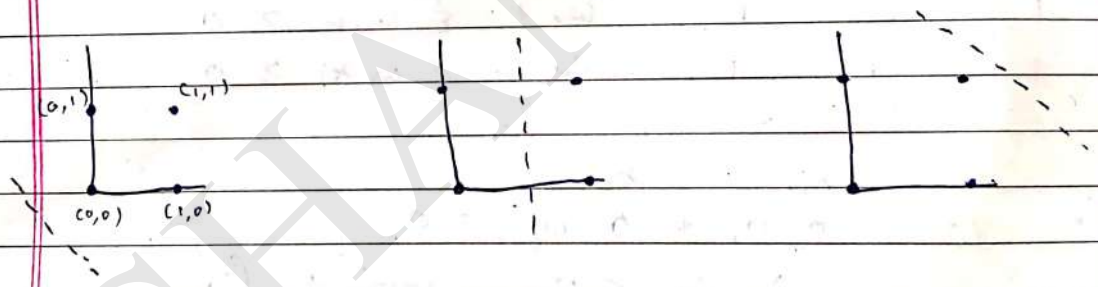
$-1 + 1.1 x_1 + 1.1 x_2 = 0$

$\Rightarrow \quad 1.1 x_1 + 1.1 x_2 = 1$



* Errors and Error Surfaces

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



Error = 1
since (0,0) is misclassified

Error = 1
since (0,1) is misclassified

Error = 3
since (0,1), (1,1), (1,0) are misclassified

Equation of line $\Rightarrow \quad w_0 + w_1 x_1 + w_2 x_2 = 0$

Thus we want values of w_0, w_1, , w_2 such that error is minimum and in best case it is 0.

* Perceptron Learning Algorithm

Lets say

$[\overset{1}{x_0}, x_1, x_2, \ldots, x_n]$ → Attributes

$[w_0, w_1, w_2, \ldots, w_n]$ → Weights

→ Algorithm

P ← when all inputs result in target class 1
N ← when all input result in target class 0
initialize random weight w

while ! convergence do → all data entries
    pick random $x \in (P \cup N)$
    if $x \in P$ and $w.x < 0$ then
        $w = w + x$ —— (1)
    end

    if $x \in N$ and $w.x \geq 0$ then
        $w = w - x$ —— (2)
    end
end

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

→ Intution behind (1) & (2) statements

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

then any point on this line say, $[x_0, x_1, x_2, x_3 \ldots x_n]$
(random numbers satisfying $w.x = 0$)

This point will be perpendicular to ω thus similar for all other points

$\therefore$ ω is vector perpendicular to the line

because $\cos\alpha = \dfrac{\omega \cdot x}{\|\omega\| \|x\|}$

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

$\cos\alpha \times \|\omega\| \|x\| = 0$

($\|\omega\| \|x\|$ can't be zero)

$\cos\alpha$ ↓

thus $\alpha = 90$

$\cos\alpha = 0$

Now, from perceptron model

$y = 1$; if $\omega^T x \geq 0$

$y = 0$; if $\omega^T x < 0$

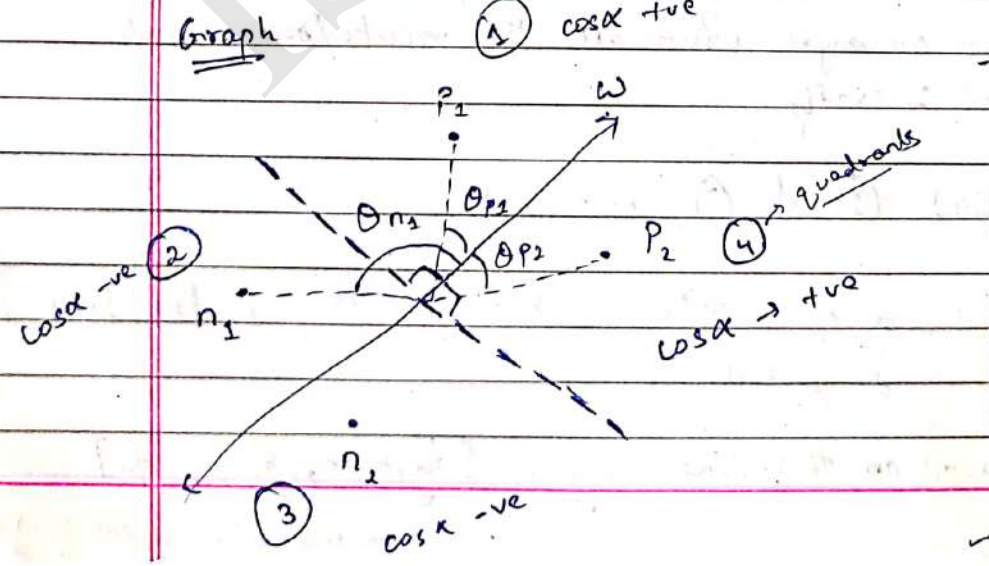
$\therefore \cos\alpha = \dfrac{\omega \cdot x}{\|\omega\| \|x\|} = \dfrac{\omega^T x}{\|\omega\| \|x\|}$

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

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

($\because$ $\|\omega\|$ & $\|x\|$ are always +ve)

Graph



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

Statement 1

- Here, in (1) → $w = w + x$

initially $\cos\alpha = w^T x$

Now

$\cos\alpha_{new} = (w + x)^T x$

$= w^T x + x^T x$

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

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

thus $\alpha_{new} < \alpha$

Here, $w \cdot x < 0$

this means that angle between x & w is greater than 90 & we want to make it less than 90 thus we reduce

- in statement (2) → $w = w - x$

$\cos\alpha = w^T x$

Now,

$\cos\alpha_{new} = w^T x - x^T x$

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

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

thus $\alpha_{new} > \alpha$

Here, $w \cdot x \geq 0$

this means angle between x & w is less than 90 & we want it to be greater than 90

→ Proof of Convergence

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

∴ We consider single set

$P' = P \cup N^-$ → N in which x is made -x

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

Improved Perceptron Learning Algorithm

P ← inputs which result into target class 1
N ← inputs which result into target class 0

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

$P' \leftarrow P \cup N^-$
initialize ω randomly
while !convergence do
    pick random $p \in P'$
    $p \leftarrow \frac{p}{\|p\|}$ normalize to make unit norm
    if $\omega \cdot p < 0$ then
        $\omega = \omega + p$
    end
end

// Algorithm converges if all points are correctly classified

similar to earlier for $x \in P$
but for $x \in N$
since now, $-x \in N^-$
$\Rightarrow \omega^T(-x) \geq 0$

condition becomes same for $P \& N^-$ and thus for N
(∵ if $\omega \cdot p < 0$ then $\omega = \omega + p$)

DELUXE
PAGE NO.:
DATE

Proof

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

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

Let β be angle between ω^* & ω_{t+1}

↳ assume that ω^* is normalized unit vector and is the solution

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

Now,

- Numerator $= \omega^* \cdot \omega_{t+1} = \omega^* \cdot (\omega_t + p_i)$

$$= \omega^* \cdot \omega_t + \omega^* \cdot p_i$$

$$\geq \omega^* \cdot \omega_t + \delta \qquad (\delta = \min\{\omega^* \cdot p_i \ \forall i\})$$

$$\geq \omega^* \cdot (\omega_{t-1} + p_j) + \delta$$

$$\geq \omega^* \cdot \omega_{t-1} + \omega^* \cdot p_j + \delta$$

$$\geq \omega^* \cdot \omega_{t-1} + 2\delta$$

$$\geq \omega^* \cdot \omega_0 + k\delta \qquad (\text{By induction})$$

Here it is assumed that at time stamp t

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

- $\text{Denominator}^2 = \|w_{t+1}\|^2$

$= (w_t + p_i) \cdot (w_t + p_i)$

$= \|w_t\|^2 + 2 w_t \cdot p_i + \|p_i\|^2$

$\leq \|w_t\|^2 + \|p_i\|^2 \quad (\because w_t \cdot p_i < 0)$

$< \|w_t\|^2 + 1 \quad (\because \|p_i\|^2 = 1)$

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

$< \|w_0\|^2 + (k) \quad (\text{By induction})$

Now

$$\cos\beta \geq \frac{w^* \cdot w_0 + k\delta}{\sqrt{\|w_0\|^2 + (k)}}$$

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

~~finite~~

* Linearly inseperable boolean functions

x_1	x_2	XOR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
0	1	1	$w_0 + \sum w_i x_i \geq 0$
1	0	1	$w_0 + \sum w_i x_i \geq 0$
1	1	0	$w_0 + \sum w_i x_i < 0$

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

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

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

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

not possible

Same can be observed graphically as well.

→ Boolean functions for two inputs

x_1	x_2	f_1	f_2	f_3	. . . . .	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			

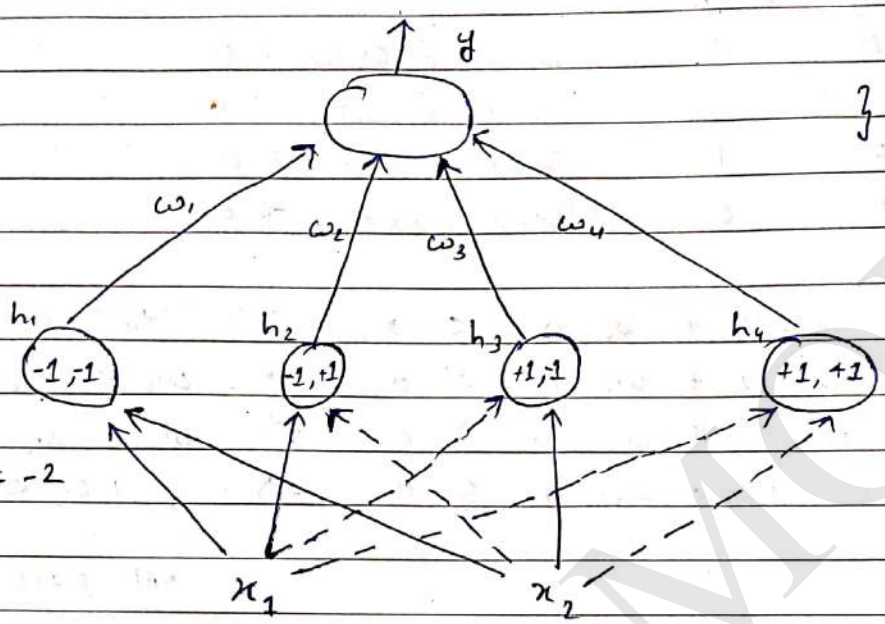
only XOR & !XOR from these 16 function are linearly inseperable

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

* Network of Perceptrons

(implementing linearly inseperable boolean functions)

figure 1



Here assume that x_1 & x_2 can assume two values

1 ⇒ true

-1 ⇒ False instead of zero.

Since bias = -2 ∴ neurons will hit only if weighted sum ≥ 2

these lines have weights as mentioned

- dotted line = $w_1 = +1$
- normal line = $w_2 = -1$

} connected with input layer

Here,

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

thus it fires

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

thus it fires

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

thus it fires

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

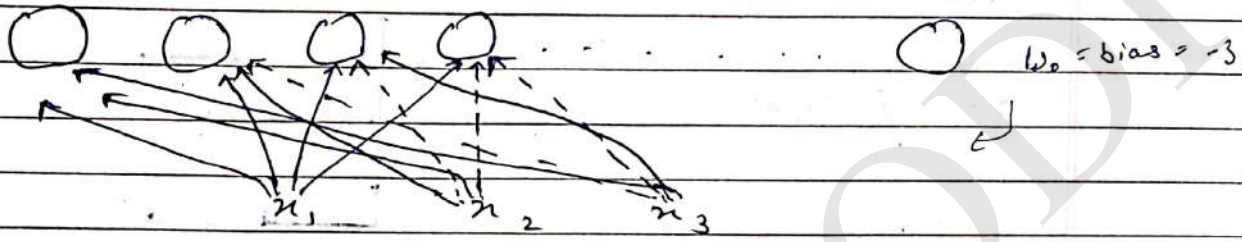
thus it fires

Concept is called MLP (Multilayered Perceptrons)

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

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

Lets 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 $\Sigma w_i h_i \geq w_0$ for XOR = 1

$\Sigma w_i h_i < w_0$ for XOR = 0

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

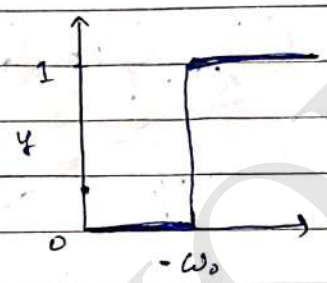
now we can randomly fix w_0 & choose w_1, w_2, w_3, w_4 and classification is done

WEEK 3

* Sigmoid Neurons

→ Need ?

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



based on w_0 it is classified

if $z = 0.51$ ans = 1
$z = 0.49$ ans = 0

} values are close but different ans ⇒ Harsh classification

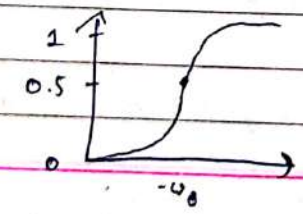
Thus we need sigmoid

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

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

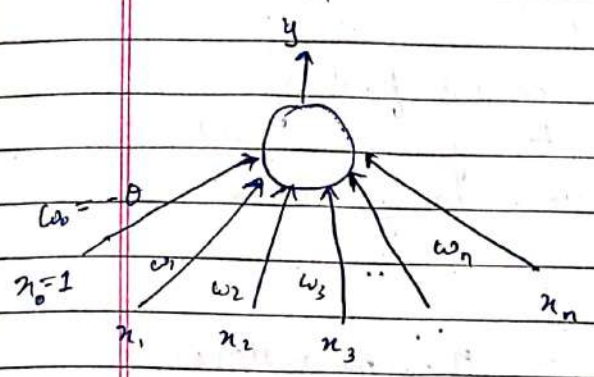
- at $z = \infty \Rightarrow$ sigmoid $(z) = 1$
- at $z = -\infty \Rightarrow$ sigmoid $(z) = 0$
- at $z = 0 \Rightarrow$ sigmoid $(z) = 0.5$

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



Smooth function
(no sharp transition around threshold)

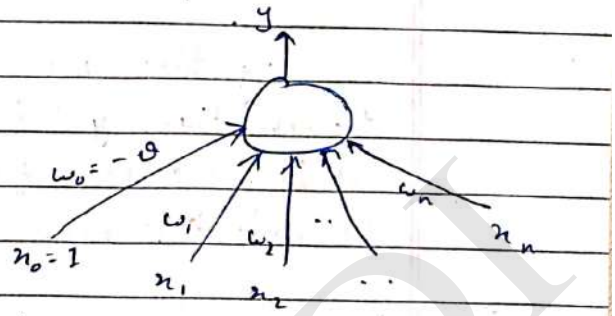
Perceptron



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

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

Sigmoid



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

(Smooth, continuous & differentiable)

* Supervised Machine learning Setup

→

- Data : $\{ x_i, y_i \}_{i=1}^{n}$
  - x_i ↓ set of all inputs
  - y_i ↓ target output
- Model : Our approximation of relation between x & y

eg → $\hat{y} = \frac{1}{1 + e^{-(\omega^T x)}}$ (logistic regression)

$\hat{y} = \omega^T x$ (Linear regression)

$\hat{y} = x^T W x$ (quadratic form)

$1 \times n \quad n \times n \quad n \times 1$

refer machine learning notes

ω 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 would not be able to adjust to get close to y because we don't have parameter like w

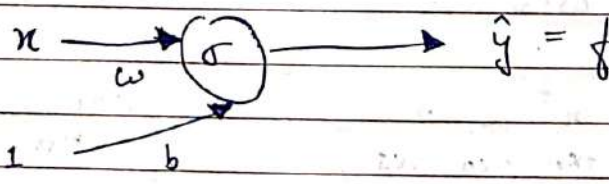
- Parameters → all the w are the parameters which is to be learned from data
- Learning Algorithm → An algorithm for learning parameters of a model (eg - perceptron learning algo, gradient descent etc)
- Objective / Loss / Cost / Error function : guides learning algorithm

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

Why we use sum of square error
- It is differentiable
- square is done because the positive & negative error 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,

$(x, y) = (0.5, 0.2)$ & $(2.5, 0.9)$ } two data points } as training data

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

$f(0.5) \rightarrow 0.2$ & $f(2.5) \rightarrow 0.9$

→ lets take an initial random guess

$w = 0.5$, $b = 0$

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

$$L(w, b) = \frac{1}{2} \sum_{i=1}^{N} (y_i - f(x_i))^2 = \frac{1}{2} * (0.9 - f(2.5))^2 + \frac{1}{2} * (0.2 - f(0.5))^2$$

$$= 0.073$$

w	b	$L(w, b)$	
0.5	0	0.0730	
-0.10	0	0.1481	} next random guess Loss increases so now we increase w beyond 0.5
0.94	-0.94	0.0214	} So we keep on making random guesses in direction where Loss is decreasing & stop at desired value of Loss
1.42	-1.73	0.0028	
1.65	-2.08	0.0003	
1.78	-2.27	0.0000	

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

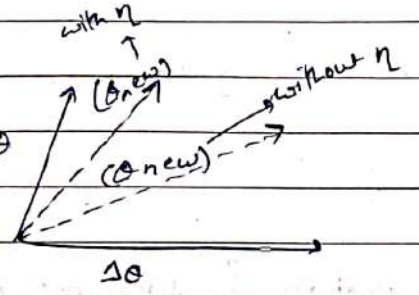
$\theta = [w, b] \in R^2$

↳ since two parameters it is 2 dimensional vector

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

↳ change in values of w, b

∴



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

$\therefore \quad \theta_{new} = \theta + \eta \cdot \Delta\theta$ ← this vector here denotes the direction

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

Ans - direction opposite to that of gradient

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

$$L(\theta + \eta u) = L(\theta) + \eta * u^T \nabla L(\theta) + \frac{\eta^2}{2!} u^T \nabla^2 L(\theta)\, u + \frac{\eta^3}{3!} \ldots + \frac{\eta^4}{4!} \ldots + \ldots$$

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

(ignoring $\eta^2, \eta^3, \ldots$ because η is a small value & $\eta^2, \eta^3, \ldots$ are close to 0)

Formulas

Refer Machine Learning notes for better Understanding

- Gradient → $\nabla L[w, b]$

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

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

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

- Taylor Series →

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

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

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

$\therefore$ It can be said

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

↓

+ve because we wanted to move in direction of $\Delta\theta$ thus we add +ve constant

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

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

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

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 u that we intend to move in should be at 180° wrt the gradient
- We move in direction opposite to the gradient

$$\therefore \begin{bmatrix} w_{t+1} \\ b_{t+1} \end{bmatrix} = \begin{bmatrix} w_t \\ b_t \end{bmatrix} - \eta \begin{bmatrix} \nabla w_t \\ \nabla b_t \end{bmatrix}$$

θ_{t+1} $\quad \theta_t \quad \nabla L(\theta_t)$

Here,

$$\nabla w_t = \frac{\partial L(w,b)}{\partial w} \quad \& \quad \nabla b = \frac{\partial L(w,b)}{\partial b}$$

$w = w_t, \; b = b_t$

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

→ Algorithm

$t \leftarrow 0$
max_iterations $\leftarrow$ 1000;
while $t <$ max_iterations do
    $w_{t+1} = w_t - \eta \nabla w_t$;
    $b_{t+1} = b_t - \eta \nabla b_t$;
end

→ Calculating ∇w_t & ∇b_t

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

$$\nabla w_t = \frac{\partial \frac{1}{2}\sum (y_i - f(x_i))^2}{\partial w} = \left[\frac{x}{x} y_i - f(x_i)\right] \frac{\partial f(x_i)}{\partial w} \quad x-1$$

$$= \sum \left(y_i - f(x_i)\right) x - \left(\frac{\partial \sigma(wx+b)}{\partial w}\right)$$

$$= \frac{\partial \frac{1}{1+e^{-(wx+b)}}}{\partial w} \times \left(y_i - f(x_i)\right)$$

$$= \left(y_i - f(x_i)\right) x$$

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

$= -\frac{\left(-e^{-(wx+b)}\right)}{\left(1+e^{-(wx+b)}\right)^2} \times x$ by quotient rule

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

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

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

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

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

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

$\therefore f(x) \times (1 - f(x)) \times x$

$\therefore \Rightarrow \Sigma\left[(y_i - f(x_i)) \times f(x_i) \times (1-f(x_i)) \times x_i\right]$

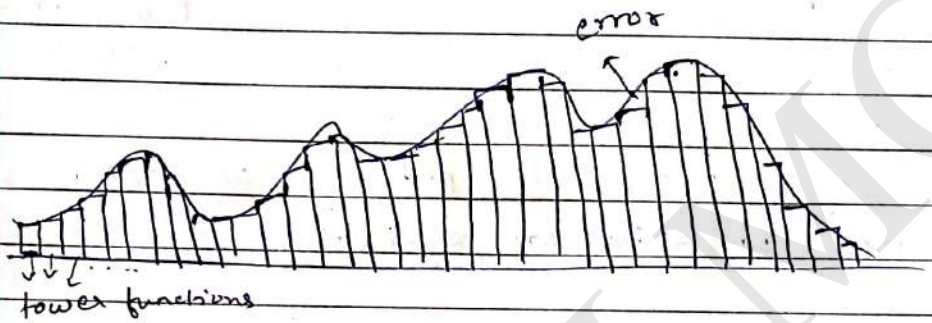
$\nabla b_t = \Sigma\left[(y_i - f(x_i)) \times f(x_i) \times (1-f(x_i))\right]$

* Representation Power of Multilayer Network of Sigmoid Neurons

→ Universal Approximation Theorem

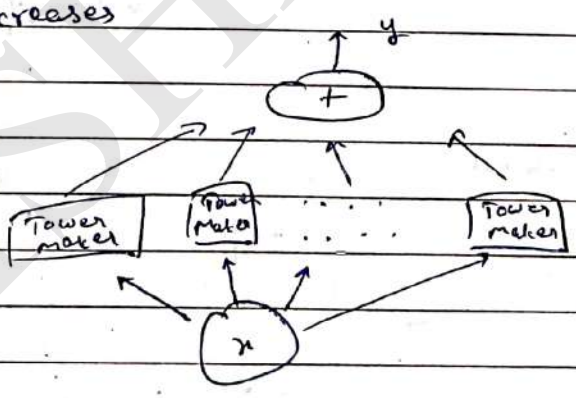
There is a guarantee that for any function $f(x) : R^n \to R^m$ we can always find a neural network (with 1 hidden layer containing enough neurons) whose output $g(x)$ satisfies $|g(x) - f(x)| < \varepsilon$!!

→



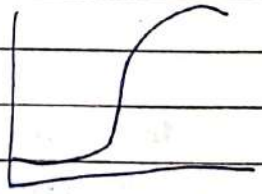
for any function denoted by the curved line above there are tower functions which help us approximate the curve

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



} generate tower function of different magnitude for different instances

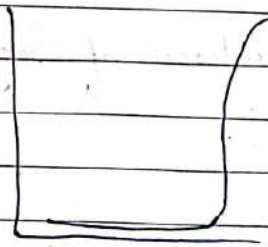
→



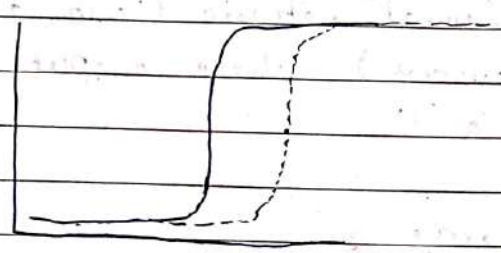
in a sigmoid curve when the value of w, b are normal

∞ normal sigmoid curve

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

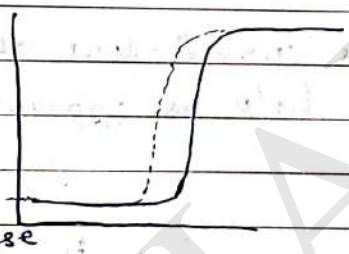


sigmoid function becomes similar to step function when the value of w become very high (because w is the slope)

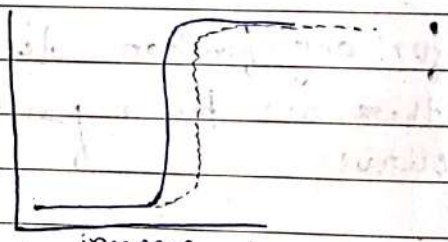


sigmoid function shifts its position when b becomes very high or very low (because b is the intercept)

→ This is a single dimensional input ie; w, there is only one input parameter w against b

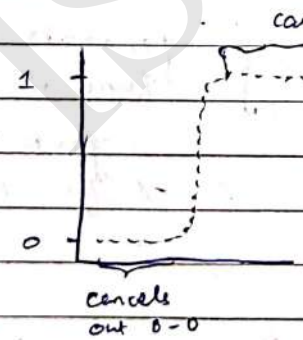


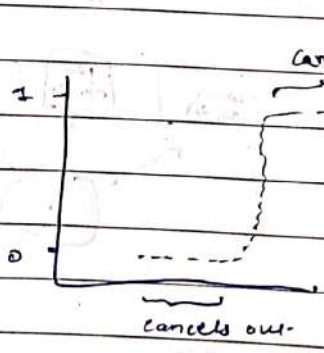
decrease ~~increase~~ ~~decrease~~ the value of b



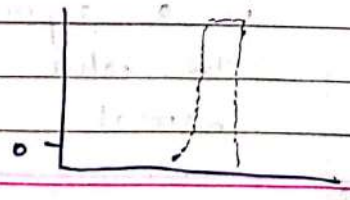
~~decrease~~ increase ~~increase~~ the value of b

Now

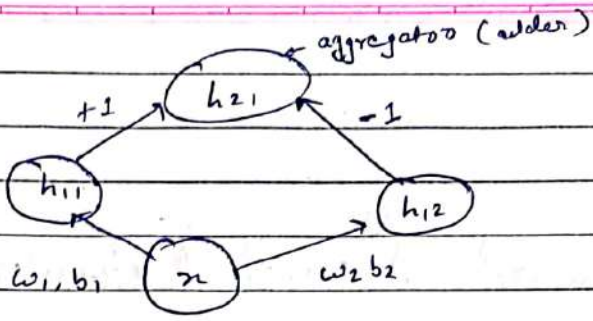




=



tower function is obtained



Refer (notes folder in course material

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
" " " " network 2D tower_maker.py

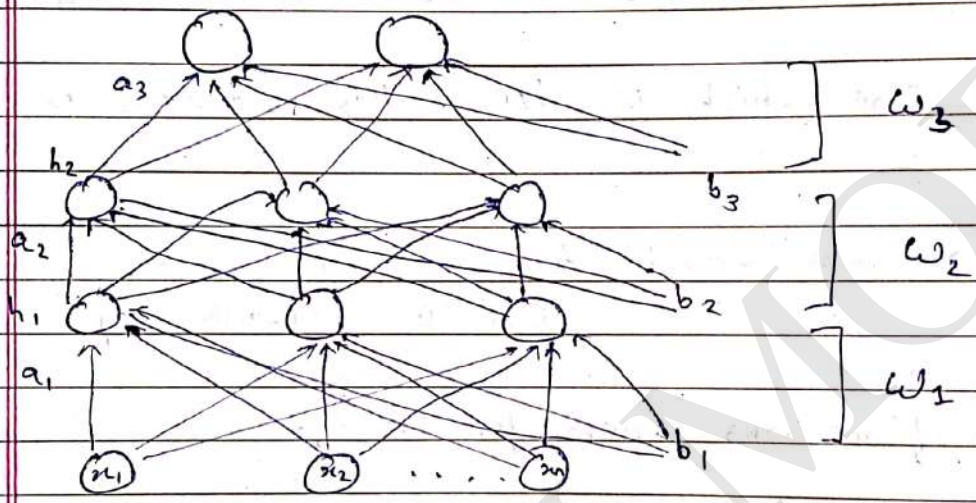
→ ∴ No of neurons required for 1D = 2
2D = 4

∴ No. of neurons for nD = O(2n)

WEEK 4

* Feed Forward Neural Networks

→



- Pre-activation at layer i (aggregation ⇒ pre-activation) in this case

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

eg $a_1 = b_1 + W_1 h_0$

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

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

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

- Activation Function at i layer

$$h_i(x) = g(a_i(x))$$

$$\begin{bmatrix} h_{i1} \\ h_{i2} \\ h_{i3} \end{bmatrix} = g\left(\begin{bmatrix} a_{i1} \\ a_{i2} \\ a_{i3} \end{bmatrix}\right) = \begin{bmatrix} g(a_{i1}) \\ g(a_{i2}) \\ g(a_{i3}) \end{bmatrix}$$

g = can be anything eg σ

- Activation at final layer

$$\therefore f(x) = (h_L(x)) = O(a_L(x))$$

→ can be sigmoid or softmax as per requirement

→ Now, if

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

$$\hat{y}_i = f(x_i) = O(w^3 g(w^2 g(w^1 x + b_1) + b_2) + b_3)$$

- Parameters

$$\theta = w_1, w_2, \ldots, w_L, b_1, b_2, \ldots b_L \quad \{L = 3\}$$

- Algorithm - Gradient Descent with back prop
- Objective Function

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

N → no. of data points; k → predicted output vector

PAGE NO.:
DATE:

* Learning Parameters Feed Forward N.N

→ In gradient descent

$\theta \leftarrow [\omega_1 \dots \omega_n, b_1 \dots b_n]$

while . . .

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

. . .

end

Now,

Here value of $\nabla\theta$ will be as follows since $\omega_1 \dots \omega_n$ are matrix and $b_1 \dots b_n$ are vector

$$\nabla\theta = \begin{bmatrix} \frac{\partial L(\theta)}{\partial \omega_{111}} & \dots & \frac{\partial L(\theta)}{\partial \omega_{11n}} & \frac{\partial L(\theta)}{\partial \omega_{211}} & \dots & \frac{\partial L(\theta)}{\partial \omega_{21n}} & \dots & \frac{\partial L(\theta)}{\partial \omega_{L11}} & \dots & \frac{\partial L(\theta)}{\partial \omega_{L1k}} \\ \frac{\partial L(\theta)}{\partial \omega_{121}} & \dots & \frac{\partial L(\theta)}{\partial \omega_{12n}} & \frac{\partial L(\theta)}{\partial \omega_{221}} & \dots & \frac{\partial L(\theta)}{\partial \omega_{22n}} & \dots & \frac{\partial L(\theta)}{\partial \omega_{L21}} & \dots & \frac{\partial L(\theta)}{\partial \omega_{L2k}} \\ \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots \\ \frac{\partial L(\theta)}{\partial \omega_{1n1}} & \dots & \frac{\partial L(\theta)}{\partial \omega_{1nn}} & \frac{\partial L(\theta)}{\partial \omega_{2n1}} & \dots & \frac{\partial L(\theta)}{\partial \omega_{2nn}} & \dots & \frac{\partial L(\theta)}{\partial \omega_{Ln1}} & \dots & \frac{\partial L(\theta)}{\partial \omega_{Lnk}} \\ \underbrace{\qquad\qquad}_{\omega_1} & & & \underbrace{\qquad\qquad}_{\omega_2} & & & \dots & \underbrace{\qquad\qquad}_{\omega_L} & & \end{bmatrix} \quad \begin{matrix} \frac{\partial L(\theta)}{\partial b_{11}} & \dots & \frac{\partial L(\theta)}{\partial b_{L1}} \\ \frac{\partial L(\theta)}{\partial b_{12}} & \dots & \frac{\partial L(\theta)}{\partial b_{L2}} \\ \vdots & & \vdots \\ \frac{\partial L(\theta)}{\partial b_{1n}} & \dots & \frac{\partial L(\theta)}{\partial b_{Ln}} \end{matrix} \quad \begin{matrix} \} b_1 \\ \} b_2 \\ \vdots \\ \} b_L \end{matrix}$$

$\therefore$ total parameters

			Last layer	
$(L-1)$	$(n \times n)$	ω	$1 \times (n \times k)$	ω
$(L-1)$	(n)	b	$1 \times (k \times 1)$	b

Assuming that we multiply ω^T with the input

* Output Function

→ Expectation

$\sum p_{\text{probability}}(x_i) \times v(x_i)$

Refer notes / week 4 / n

now for certain & outc
gain is 0. And for
information gain is

gain := I(A

$= - \sum p(x_i)$

→ These probability dis
function called soft

softmax(z) =

* Output Function & Loss Function

→ Expectation

$$\sum p_{\text{probability}}(x_i) \times v_{\text{value}}(x_i) = \text{Expectation}$$

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

$$= -\sum P(x_i) \log_2 (P(x_i))$$

Now lets say,

predicted $\hat{y} = q$ (probability distribution)

$y = p$ (actual probability distribution)

$$\frac{\partial L(\theta)}{\partial b_{11}} \quad \cdots \quad \frac{\partial L(\theta)}{\partial b_{L1}} \} b_1$$

$$\frac{\partial L(\theta)}{\partial b_{12}} \quad \cdots \quad \frac{\partial L(\theta)}{\partial b_{L2}} \} b_2$$

$$\vdots \qquad\qquad \vdots \qquad\qquad \vdots$$

$$\frac{\partial L(\theta)}{\partial b_{1n}} \quad \cdots \quad \frac{\partial L(\theta)}{\partial b_{Ln}} \} b_L$$

$i = k$

So, expected information

$$= -\sum p(x_i) \log_2 (q(x_i))$$] 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

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

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

eg

$$a_L = [a_{L_1}^{10} \quad a_{L_2}^{-20} \quad a_{L_3}^{30}]$$

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

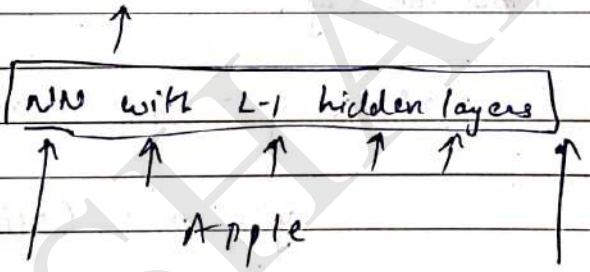
$$\rightarrow \hat{y}_1 = \frac{e^{10}}{e^{10} + e^{-20} + e^{30}}$$

Note → $\sum \hat{y} = 1$ } since $\hat{y}$ is probability distribution

→ Now in Classification task

apple mango orange Banana

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



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,

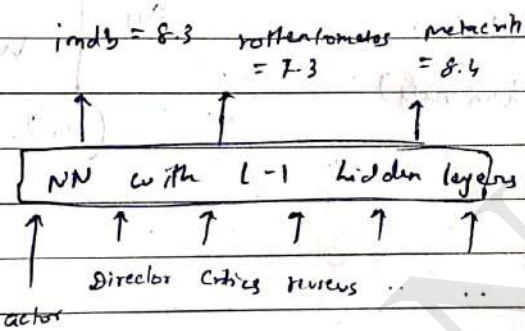
$$-\sum_{c=1}^{k} y_c \log \hat{y}_c \Rightarrow -\log \hat{y}_c$$ (y_c is correct classification 1)

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

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



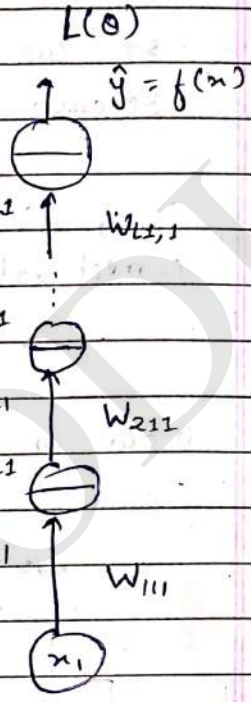
In this case output function can't be sigmoid because it gives value between 0 and 1

So we don't apply sigmoid & straight away let the linear function go in output. ie - a_L

→ imp

	Outputs	
	Real Values	Probabilities
Output Activation	Linear	Softmax
Loss Function	Squared error	Cross Entropy

* Back Propogation (Intuition)



Normal chain rule

$$\frac{\partial L(\theta)}{\partial W_{111}} = \left(\frac{\partial L(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{L1}} \frac{\partial a_{L1}}{\partial h_{21}} \frac{\partial h_{21}}{\partial a_{21}} \frac{\partial a_{21}}{\partial h_{11}}\right) \times \left(\frac{\partial h_{11}}{\partial a_{11}} \frac{\partial a_{11}}{\partial W_{111}}\right)$$

$$\frac{\partial L(\theta)}{\partial W_{111}} = \frac{\partial L(\theta)}{\partial h_{11}} \times \frac{\partial h_{11}}{\partial W_{111}}$$

(compressing chain rule)

similarly

$$\frac{\partial L(\theta)}{\partial W_{211}} = \frac{\partial L(\theta)}{\partial h_{21}} \frac{\partial h_{21}}{\partial W_{211}}$$

$$\frac{\partial L(\theta)}{\partial W_{L11}} = \frac{\partial L(\theta)}{\partial a_{L1}} \frac{\partial a_{L1}}{\partial W_{L11}}$$

→ Quantities of interest

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

$$\frac{\partial L(\theta)}{\partial W_{111}} = \frac{\partial L(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_3} \frac{\partial a_3}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_1}{\partial W_{111}}$$

- $\frac{\partial L(\theta)}{\partial W_{111}}$ — talk to weight
- $\frac{\partial L(\theta)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_3}$ — Talk to o/p layer
- $\frac{\partial a_3}{\partial h_2} \frac{\partial h_2}{\partial a_2}$ — talk to previous hidden layer
- $\frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1}$ — talk to previous hidden layer
- $\frac{\partial a_1}{\partial W_{111}}$ — Now talk to activation

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

DELUXE PAGE NO.: DATE:

* Computing Gradients wrt Output Units

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

$\frac{\partial L(\theta)}{\partial \hat{y}_{oi}}$

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

Now $\frac{\partial L(\theta)}{\partial \hat{y}_{oi}} = \frac{\partial -\log(y_l)}{\partial \hat{y}_{oi}}$

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

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

Differentiating loss with each element (i) in y_k vector

⇓ can be written as

$\frac{\mathbb{I}(i=l)}{\hat{y}_l}$ } is indicator function gives 1 if $i=l$, 0 otherwise

$$\nabla_{\hat{y}} L(\theta) = \frac{\partial L(\theta)}{\partial \hat{y}_{oi}} = \frac{\mathbb{I}(i=l)}{\hat{y}_l} \Big\} = \begin{bmatrix} \frac{\partial L(\theta)}{\partial \hat{y}_1} \\ \frac{\partial L(\theta)}{\partial \hat{y}_2} \\ \vdots \\ \frac{\partial L(\theta)}{\partial \hat{y}_k} \end{bmatrix} = -\frac{1}{y_l} \begin{bmatrix} \mathbb{I}(l=1) \\ \mathbb{I}(l=2) \\ \vdots \\ \mathbb{I}(l=k) \end{bmatrix}$$

$= e_{o(l)}$ (one hot vector)

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

$\frac{\partial L(\theta)}{\partial a_{Li}}$

$= \frac{\partial -\log \hat{y}_l}{\partial \hat{y}_l} \times \frac{\partial \hat{y}_l}{\partial a_{Li}}$

$= -\frac{1}{\hat{y}_l} \times \frac{\partial \left(\frac{\exp(a_{Ll})}{\sum_{i=1} \exp(a_{Li})} \right)}{\partial a_{Li}}$

derivative of softmax

$$\frac{\partial}{\partial a_{Li}} \frac{\exp(a_{Ll})}{\sum \exp(a_{Li})} = \frac{\exp(a_{Ll})\, \mathbb{I}(i=l) \times \sum \exp(a_{Li}) - \exp(a_{Ll}) \exp(a_{Li})}{\sum \exp(a_{Li})^2}$$

$[a_{L1}, a_{L2}, a_{L3}, \ldots, a_{Ln}]$

$$= \mathbb{I}(i=l)\, \text{softmax}(a_{Ll}) - \text{softmax}(a_{Ll}) \times \text{softmax}(a_{Li})$$

(a_{Li}: i^{th} element in a_L)

$$= -\frac{1}{\hat{y}_l} \mathbb{I}(i=l) \times \left[\mathbb{I}(i=l)\, \text{softmax}(y_l) - \text{softmax}(y_l) \times \text{softmax}(y_i) \right]$$

$$= \mathbb{I}(i=l)\, \hat{y}_l - \hat{y}_l \hat{y}_i$$

$$= -\frac{1}{\hat{y}_l} \left(\mathbb{I}(i=l)\, \hat{y}_l - \hat{y}_l \hat{y}_i \right)$$

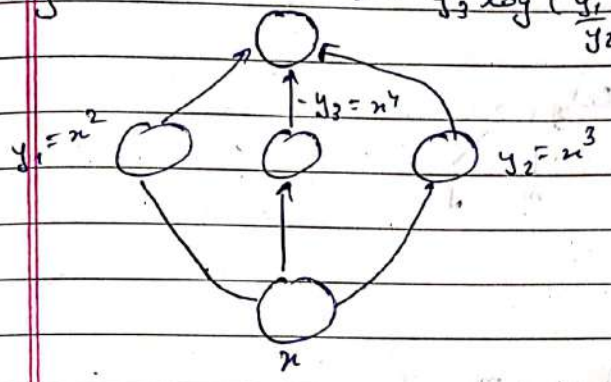
$$= -\left(\mathbb{I}(i=l) - \hat{y}_i \right)$$

$$\therefore \nabla a_L = \begin{bmatrix} \frac{\partial L(\theta)}{\partial a_{L1}} \\ \vdots \\ \frac{\partial L(\theta)}{\partial a_{Lk}} \end{bmatrix} = \begin{bmatrix} -\left(\mathbb{I}(l=1) - \hat{y}_1 \right) \\ -\left(\mathbb{I}(l=2) - \hat{y}_2 \right) \\ \vdots \\ -\left(\mathbb{I}(l=k) - \hat{y}_k \right) \end{bmatrix}$$

$= -(e(l) - f(x))$ ← All here computed $[\nabla_{a_L} L(\theta)]$

* Computing Gradients wrt. Hidden Units

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



$$\frac{dz}{dx} = \sum_{i=1}^{3} \frac{dz}{dy_i} \frac{dy_i}{dx}$$

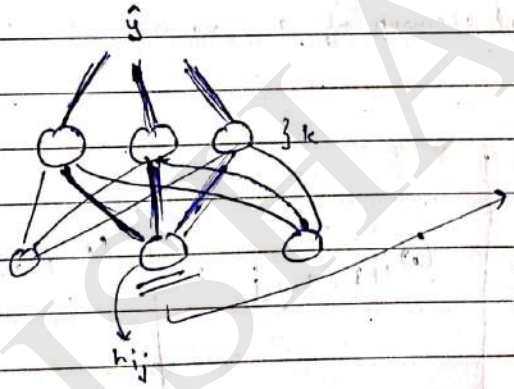
↓

Same concept is used for N.N (adding chained derivatives)

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

→ refer ~~back prop~~ notes / week4 / backpropdiagram1.png

→



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

i^{th} hidden layer; j^{th} neuron in hidden layer

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

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

$h_{ij} = h_{22}$

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

Now, $a_{3_1}^{i+1\ \ m}$

$$\frac{\partial a_{3_1}}{\partial h_{22}} = \frac{\partial \left(w_{311} h_{21} + w_{312} h_{22} + w_{313} h_{23} + b_{31} \right)}{\partial h_{22}}$$

h_{22} : i j

$$= w_{3+2}$$

$i+1$ m j

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

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

$\frac{\partial L(\theta)}{\partial a_{i+1,m}}$ → individual elements of a_{i+1} vector

Now,

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

$$\therefore \nabla_{h_{ij}} L(\theta) = \left(w_{i+1,\cdot,j} \right)^T \nabla_{a_{i+1}} L(\theta)$$

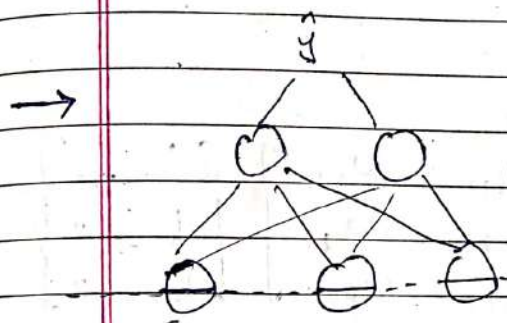
single neuron of hidden layer

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

h_i → all hidden neurons

$$= (w_{i+1})^T (\nabla_{a_{i+1}} L(\theta))$$

matrix · vector



→ we are here with $\frac{\partial L(\theta)}{\partial h_{ij}}$

Now,

$$\frac{\partial L(\theta)}{\partial a_{ij}} = \frac{\partial L(\theta)}{\partial h_{ij}} \frac{\partial h_{ij}}{\partial a_{ij}} = \frac{\partial L(\theta)}{\partial h_{ij}} g'(a_{ij}) \quad [\because h_{ij} = g(a_{ij})]$$

$\frac{\partial L(\theta)}{\partial h_{ij}}$ ↙ in case of whole vector ie for all neurons

j is from $(1 \ldots n)$ ← $\left(\frac{\partial L(\theta)}{\partial h_{ij}} \odot g'(a_{ij})\right)$ ↘ hadamard product

for all neurons

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

* Computing Gradient wrt Parameters

$$a_k = b_k + W_k h_{k-1}$$

Weights →

$$\frac{\partial a_{ki}}{\partial W_{kij}} = h_{k-1,j}$$

eg -

$$\begin{bmatrix} a_{k1} \\ a_{k2} \\ a_{k3} \end{bmatrix} = \begin{bmatrix} b_{k1} \\ b_{k2} \\ b_{k3} \end{bmatrix} + \begin{bmatrix} W_{k11} & W_{k12} & W_{k13} \\ W_{k21} & W_{k22} & W_{k23} \\ W_{k31} & W_{k32} & W_{k33} \end{bmatrix} \begin{bmatrix} h_{k-1,1} \\ h_{k-1,2} \\ h_{k-1,3} \end{bmatrix}$$

$$\frac{\partial a_{k1}}{\partial W_{kij}} = \frac{\partial b_{k1}}{\partial W_{k12}} + W_{k11} h_{k-11} + W_{k12} h_{k-12} + W_{12} h_{k13}$$

$i = 1, 2$

$$= h_{k-1\,2}$$

$\downarrow$

j

$$\therefore \frac{\partial a_{ki}}{\partial W_{kij}} = h_{k-1\,j}$$

Now,

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

Let's take 3×3 matrix as weight for example

$$\nabla_{W_k} L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial W_{k00}} & \frac{\partial L(\theta)}{\partial W_{k01}} & \frac{\partial L(\theta)}{\partial W_{k02}} \\ \frac{\partial L(\theta)}{\partial W_{k10}} & \frac{\partial L(\theta)}{\partial W_{k11}} & \frac{\partial L(\theta)}{\partial W_{k12}} \\ \frac{\partial L(\theta)}{\partial W_{k20}} & \frac{\partial L(\theta)}{\partial W_{k21}} & \frac{\partial L(\theta)}{\partial W_{k22}} \end{bmatrix}$$

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

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

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

(3×1) (1×3) → (3×3)

Biases →

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

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

$$\nabla_{b_k} L(\theta) = \left[\frac{\partial L(\theta)}{\partial a_{k0}} \quad \frac{\partial L(\theta)}{\partial a_{k1}} \quad \cdots \quad \frac{\partial L(\theta)}{\partial a_{kn}} \right]^T$$

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

* Back Propagation (Algorithm)

→ $t \leftarrow 0$

max_iterations $\leftarrow$ 1000

Initialize $\theta_0 = [w_1^0, \ldots w_L^0, b_1^0 \ldots b_L^0]$;

while t++ < max_iterations do

$h_1, h_2 \ldots h_{L-1}, a_1, a_2 \ldots a_L, \hat{y} = \text{forwardprop}(\theta_t)$;

$\nabla\theta_t = \text{backwardprop}(h_1, h_2 \ldots h_{L-1}, a_1, a_2 \ldots a_L, \hat{y})$;

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

end

forward prop →

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

$a_k = b_k + W_k h_{k-1}$;

$h_k = g(a_k)$;

end

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

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

backward prop →

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

for $k = L$ to 1 do

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

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

} // Compute gradients wrt parameters

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

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

end

* Derivative of Activation Function

→ Sigmoid

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

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

$$= \frac{1}{1+e^{-z}} \left(\frac{1+e^{-z}-1}{1+e^{-z}} \right)$$

$$= g(z)(1-g(z))$$

→ Tanh

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

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

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

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

* Information Content, Entropy & cross Entropy

→ Information Gain / content $\propto \frac{1}{P(A)}$ — (1)

Let X & Y be random variables of independent events

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

$f(P(X \cap Y)) = f(P(X)) + f(P(Y))$ — from (1)

(∵ events are independent)

IG can be expressed as function of probability

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

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

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

→ Entropy = Expected information content

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

→ Cross Entropy

		bits required	
A →	$\frac{1}{4}$	00	$IG(A) = -\log_2 \frac{1}{4} = 2$ bits required
B →	$\frac{1}{4}$	01	$IG(B) = " = 2$
C →	$\frac{1}{4}$	10	$IG(C) = " = 2$
D →	$\frac{1}{4}$	11	$IG(D) = " = 2$

Avg bits required = 2 } Entropy

Similarly for 8 variables

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

Now if,

X

$A \to 1/2$	$IG(A) = 1$	less bits required for more frequenty occuring event
$B \to 1/4$	$IG(B) = 2$	
$C \to 1/8$	$IG(C) = 3$	
$D \to 1/8$	$IG(D) = 3$	

Avg bits required = 1.75 } Entropy

Thus

Cross entropy

$$-\sum p_i \log_2 q_i$$

q_i → estimated distribution

$\log_2 q_i$ → bits required for estimated distribution

p_i → it depends on the actual distribution

Now minimize

$-\sum p_i \log_2 q_i$ (take partial derivative wrt q_i & equate to 0)

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

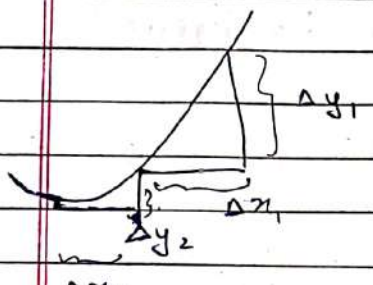
$\therefore -\sum p_i \log q_i + (\lambda(\sum q_i - 1))$ → laplace multiplier } minimum at $q_i = 1$

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

$\downarrow$

$p_i = q_i$ (if $\lambda = 1$)

WEEK 5



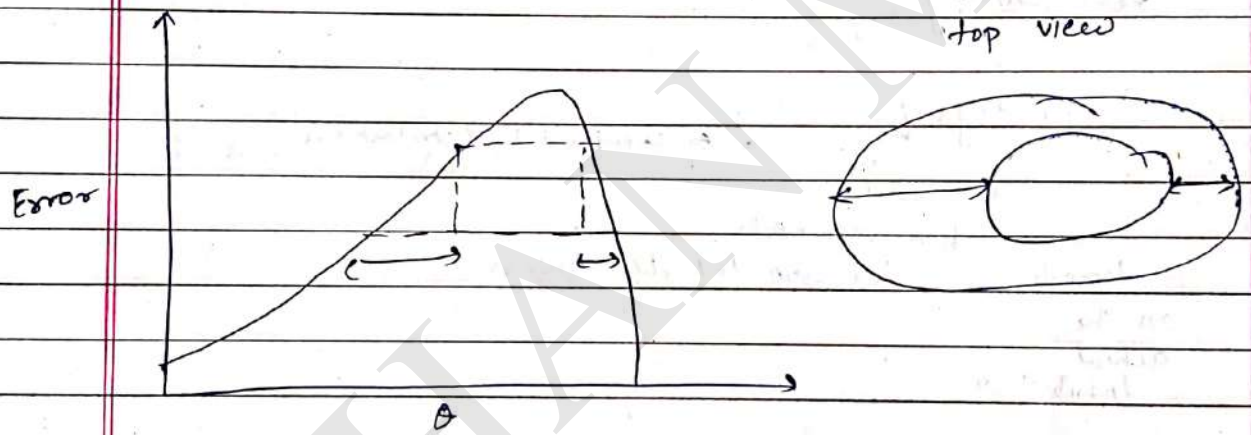
Now whenever there is steep slope derivative is high

Whenever ~~shallow~~ gentle slope derivative is low

eg $\frac{\Delta y_1}{\Delta x_1}$ = low & $\frac{\Delta y_2}{\Delta x_2}$ = 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

$$update_t = \gamma\, update_{t-1} + \eta \nabla w_t$$

$$w_{t+1} = w_t - update_t$$

working

$update_0 = 0$ } initially

$update_1 = \gamma \cdot update_0 + \eta \nabla w_1 = \eta \nabla w_1$

$update_2 = \gamma \cdot update_1 + \eta \nabla w_2 = \gamma \cdot \eta \nabla w_1 + \eta \nabla w_2$

$update_3 = \gamma \cdot update_2 + \eta \nabla w_3 = \gamma(\gamma \cdot \eta \nabla w_1 + \eta \nabla w_2) + \eta \nabla w_3$
$= \gamma^2 \cdot \eta \nabla w_1 + \gamma \cdot \eta \nabla w_2 + \eta \nabla w_3$

$update_4 = \gamma \cdot update_3 + \eta \nabla w_4 = \gamma^3 \cdot \eta \nabla w_1 + \gamma^2 \eta \nabla w_2 + \gamma \eta \nabla w_3 + \eta \nabla w_4$

exponentially weighted average
$\gamma < 1$

$\therefore update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$

$= \gamma^{t-1} \cdot \eta \nabla w_1 + \gamma^{t-2} \cdot \eta \nabla w_2 + \dots \eta \nabla w_t$

→ 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 the oscillations decrease. It is better than normal gradient descent

* Nesterov's Gradient Descent

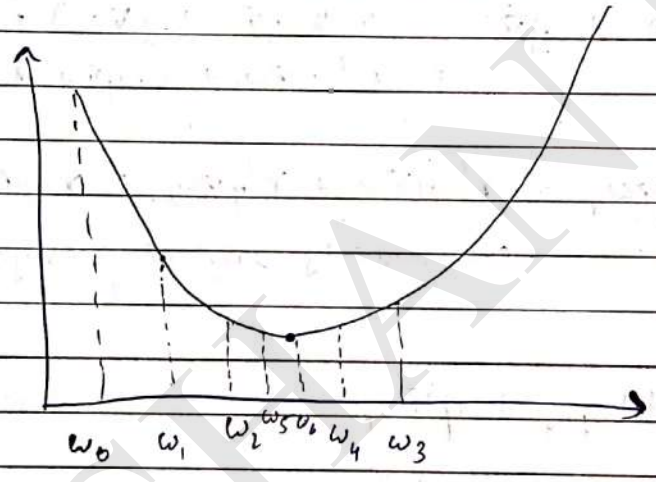
Momentum based GD → $w_{t+1} = \left(w_t - \underbrace{\gamma\, update_{t-1}}_{\text{history}}\right) - \underbrace{\eta \nabla w_t}_{\text{gradient}}$

$w_{look\_ahead} = w_t - \gamma \cdot update_{t-1}$

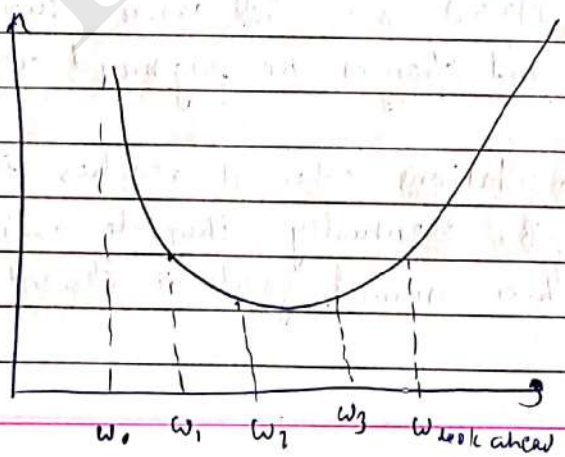
$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_{look\_ahead}$

$w_{t+1} = w_t - update_t$

Explaination



This is momentum based gradient descent



Nesterov gradient Descent

One can have stochastic or mini batch versions of Momentum base GD & Nesterov's GD

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

Then find the gradient wrt $w_{lookahead}$ and calculate $update_t$ and then update w_t with $update_t$

* 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.

Imp Notes

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

Algorithm	No. of steps in 1 epoch
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

* Adaptive Learning Rate

→ Tuning

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

pick learning rate that gives ~~the~~ 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

$\eta = \eta_0^{-kt}$ after every epoch update η
here I have assumed $t=1$

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

→ 1/t Decay

$\eta = \frac{\eta_0}{1+kt}$ k is hyperparameter
t is step size

* Momentum

→ $\gamma_t = \min\left(1 - 2^{-1-\log_2([t/250]+1)}, \mu_{max}\right)$

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

$\gamma_0 = 0.5$
$\gamma_{250} = 0.75$
$\gamma_{750} = 0.875$

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

* Line Search

→ At every epoch try n no. of learning rates
eg lets 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 choosen based on error surface

if surface is gentle higher learning rate
if surface is steep lower learning rate

* Gradient Descent with Adaptive Learning Rate

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

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

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

if x_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 wrt all gradients / elements in ∇w_t

Thus size of v_t matrix = size of ∇w_t matrix

→ RMS prop

parameters corresponding to dense inputs

(b here means the parameters updated more no. of time)

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

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

$$\omega_{t+1} = \omega_t - \frac{\eta}{\sqrt{V_t + \epsilon}} * \nabla \omega_t$$

Deromineter grows very slowly

→ Adam

$$m_t = \beta_1 * m_{t-1} + (1-\beta_1) * \nabla \omega_t$$ } similar to momentum based GD

$$V_t = \beta_2 * V_{t-1} + (1-\beta_2) * (\nabla \omega_t)^2$$ } similar to Adagrad / RMS prop

$$\hat{m} = \frac{m_t}{1-\beta_1^t} \qquad \hat{V}_t = \frac{V_t}{1-\beta_2^t} \quad — (1)$$

$$\omega_{t+1} = \omega_t - \frac{\eta}{\sqrt{\hat{V}_t + \epsilon}} * \hat{m}_t$$

- Bias Correction (obtaining statement (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

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

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

∴ 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 $\nabla w_t = g_t$

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

$m_0 = 0$

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

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

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

∴ General =

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

We need $E[m_t]$

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

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

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

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

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

$$= E[g] (1-\beta) \sum_{i=1}^{t} \beta^{t-i}$$ → is a GP

$$= E[g] (1-\beta) \left(\beta^{t-1} + \beta^{t-2} + \beta^{t-3} + \dots + \beta^{0} \right)$$

$$= E[g] (1-\beta) \left(\frac{1-\beta^t}{1-\beta} \right)$$ [formula $\left(\frac{1-r^n}{1-r}\right)$ sum of GP]

$$\therefore E[m_t] = E[g] (1-\beta^t)$$

$$E\left[\frac{m_t}{1-\beta^t}\right] = E[g]$$

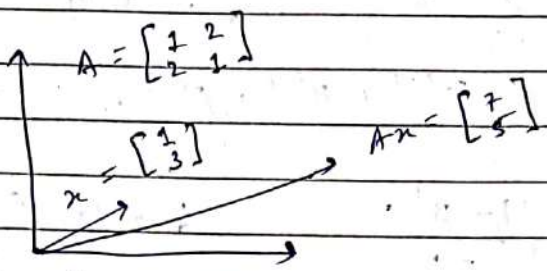
$$E[\hat{m}_t] = E[g] \qquad \left(\because \hat{m}_t = \frac{m_t}{1-\beta^t} \right)$$

similar for $\hat{v}_t$

WEEK 6

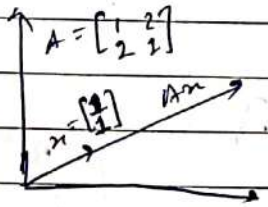
* Eigen Values & Eigen Vectors

→



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

→



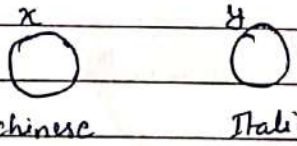
When we multiply a matrix with Eigen vector its ~~directo~~ direction doesn't change, magnitude may change.

$Ax = \lambda x$ [direction remains same]

λ → Eigen value

x → Eigen vector

→ lets consider an example



initially $\begin{bmatrix} x \\ y \end{bmatrix} = V_0$

for first timestamp

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

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

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

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

$$\therefore \quad v_n = M^n v_0 \rightarrow \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} p & 1-q \\ 1-p & q \end{bmatrix}$$

- Dominant eigen vectors & values

If $\lambda_1, \lambda_2, \ldots \lambda_n$ are eigen values of $n \times n$ matrix A then λ_1 is called dominant eigen value if

$$|\lambda_1| \geq |\lambda_i| \quad i = 2, \ldots, n$$

& corresponding vector of λ_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

$$Av_0, A^2v_0, A^3v_0, \ldots A^nv_0, \ldots$$

at some point n

$$A^nv_0 = k \times \text{dominant eigen vector}$$

Now

$$v_n = M^nv_0 = k \times \text{dominant eigen vector } (e_d)$$

$$v_{n+1} = Mv_n = M\,k\,e_d$$

Now,

$$Me_d = \lambda_d e_d$$

(λ_d: dominant eigen value; e_d: dominant eigen vector)

$$\therefore = kMe_d = k\lambda_d e_d$$

(1) If M is stochastic matrix

$$k\lambda_d e_d = ke_d \quad (\because \lambda_d = 1) \quad \Rightarrow \quad v_n = v_{n+1}$$

Stable after some time

(2) If M is normal square matrix

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

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

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

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

* Linear Algebra

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

→ Linearly independent - If for a set of vectors no vector in the set can be expressed as linear combination of other vectors in that set.

$$c_1 v_1 + c_2 v_2 + \ldots + c_n v_n = 0 \quad (c_1 = c_2 = c_3 = \ldots = c_n = 0)$$ are scalers

→ Lets consider R^2

basis → (1,0) & (0,1)
OR (2,3) & (5,7) etc

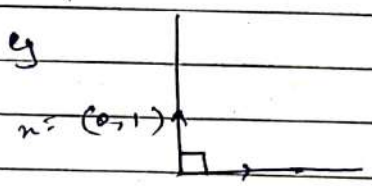
$$(x, y) = a_1 [2,3] + a_2 [5,7]$$

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

can be solved by elimination

(independent)

But if these vectors in basis are orthonormal
ie 90° angle between them (orthogonal)
unit vectors (normal)

eg 

∴ 0,1 & 1,0 are orthonormal

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

Now $x \cdot y = \cos 90 \times |x| \times |y| = 0 \times 1 \times 1$

$= 0$ — (1)

$x \cdot x = \cos 0 \times |x| |y| = 1$ — (2)

for n dimension ($z = n \times 1$ vector)

$$z = \alpha_1 u_1 + \alpha_2 u_2 + \alpha_3 u_3 + \ldots \alpha_n u_n$$

are the ~~vectors~~ orthonormal basis vectors

$$u_1^T z = \alpha_1 u_1^T u_1 + \alpha_2 u_1^T u_2 + \ldots \alpha_n u_1^T u_n$$

$u_1^T z$ ~ dot product

$\Downarrow$ from (1) $\Downarrow$ from (2)

Scalar value $= \alpha_1 \times 1 + 0 + \ldots + 0$

Scalar value $= \alpha_1$

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

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

Theorem

→ Eigen ~~value~~ vectors of matrix are independent if eigen values are distinct

Theorem

→ The eigen vectors of square symmetric matrix are orthogonal

* Eigen Value Decomposition

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

$$Au = A\begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ u_1 & u_2 & \cdots & u_n \\ \downarrow & \downarrow & & \downarrow \end{bmatrix} = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ Au_1 & Au_2 & \cdots & Au_n \\ \downarrow & \downarrow & & \downarrow \end{bmatrix}$$

Au ↓ collection of eigen vector

$$= \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ \lambda_1 u_1 & \lambda_2 u_2 & \cdots & \lambda_n u_n \\ \downarrow & \downarrow & & \downarrow \end{bmatrix}$$

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

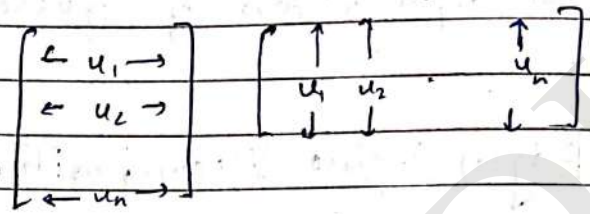
$$= U \times \lambda$$

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

$\therefore \quad Q = U^T U =$ 

$u_i^T u_j = 0 \quad$ if $\quad i \neq j$

$= 1 \quad$ if $\quad i = j$

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

Thus

eigen value decomposition = $A = U \Sigma U^T$

↓ diagonal matrix

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

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

Theorem

→ $\max_x (x^T A x)$ such that $\|x\| = 1$.

Here x will be dominant eigen vector

Theorem

→ $\min_x (x^T A x)$ such that $\|x\| = 1$

Here x will be smallest eigen vector of A