

Deep Learning Using Python

Book: Fundamentals of Deep Learning

By Nitin Buduma

Nikhil Buduma

Joe Papa

Samir Kundu

samirkundu@soa.ac.in

9475094548

1. Fundamentals of Linear Algebra for Deep Learning

2. Fundamentals of Probability

3. The Neural Network (NN)

4. Training Feed-Forward N.N

5. Beyond Gradient Descent

6. CNN

7. Embedding & Representation Learning

8. Models for sequence Analysis

9. Generative Models

10. Implementing NN using PyTorch

01-09-25

Chapter - I

Fundamental of linear Algebra

→ Deep learning :

Deep learning, DL ⊆ Machine learning, ML

It is a subset of ML that uses algorithms or techniques called Artificial NN which are inspired by the structure & functions of the brain & are called capable self-learning.

1st ML problem solved - Identifying the PIN code

[by Y LeCun, L Bottou,
Y Bengio, P Haffner (1998)]

Column Matrix : $\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}_{m \times 1}$

$\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} \\ \vdots \\ x_{m1} \end{bmatrix}_{n \times m}$

when x_{nm} is 1 it will glow otherwise

Column Space : $\begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} = c_1 \begin{pmatrix} 2 \\ 4 \end{pmatrix} + c_2 \begin{pmatrix} 3 \\ 5 \end{pmatrix}$; c_1, c_2 are scalars (The linear combination of both Column Space)

Null Space : $\boxed{Ax=0}$ $\begin{bmatrix} 4 & 6 \\ 2 & 7 \\ 5 & 8 \end{bmatrix}_{3 \times 2} \times \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{2 \times 1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}_{3 \times 1}$

$\Rightarrow 4x_1 + 6x_2 = 0$ (i)
 $2x_1 + 7x_2 = 0$ (ii)
 $5x_1 + 8x_2 = 0$ (iii) Null Space = $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Q) $A = \begin{bmatrix} 2 & -4 \\ 1 & -2 \\ -4 & 8 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & -4 \\ 1 & -2 \\ -4 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{2 \times 1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}_{3 \times 1}$

$2x_1 - 4x_2 = 0$
 $x_1 - 2x_2 = 0$
 $-4x_1 + 8x_2 = 0$

$$x_1 = 2x_2$$

$$\text{Null Space} = c \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

Eigen Values : $A - \lambda I_n = 0$ $\det(A - \lambda I) = 0$

Eigen Vectors : $(A - \lambda I_n) \alpha = 0$

03-09-25

Q) $A = \begin{bmatrix} 1 & 2 \\ 5 & 4 \end{bmatrix}$ Find Eigen Values & Eigen Vectors.

$$A = \begin{bmatrix} 1 & 2 \\ 5 & 4 \end{bmatrix} \quad \lambda I = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}$$

$$A - \lambda I = \begin{bmatrix} 1-\lambda & 2 \\ 5 & 4-\lambda \end{bmatrix}$$

$$\begin{aligned} \Rightarrow \det(A - \lambda I) &= 0 \\ \Rightarrow (1-\lambda)(4-\lambda) - 10 &= 0 \\ \Rightarrow \lambda^2 - 5\lambda - 6 &= 0 \\ \Rightarrow \lambda^2 + \lambda - 6\lambda - 6 &= 0 \end{aligned}$$

Eigen vector corresponding to $\lambda = 6 \rightarrow (A - \lambda I) \alpha = 0$
Putting $\lambda = 6$ in $A - \lambda I \Rightarrow \begin{bmatrix} -5 & 2 \\ 5 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$$\begin{aligned} \Rightarrow -5x_1 + 2x_2 &= 0 \\ 5x_1 - 2x_2 &= 0 \\ \alpha_2 &= \frac{5}{2}\alpha_1 \end{aligned}$$

$$\text{If } x_1 = c, \text{ then } x_2 = \frac{5}{2}c$$

$$\text{So, eigen vector for } \lambda = 6 \text{ is } \begin{pmatrix} c \\ \frac{5}{2}c \end{pmatrix} = c \begin{pmatrix} 1 \\ \frac{5}{2} \end{pmatrix}$$

where c is non-zero real number

Eigen vector corresponding to $\lambda = -1$

$$\Rightarrow \begin{bmatrix} 2 & 2 \\ 5 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow \begin{cases} 2x_1 + 2x_2 = 0 \\ 5x_1 + 5x_2 = 0 \end{cases} \Rightarrow x_2 = -x_1$$

If we put $x_1 = c$, then $x_2 = -c$
So, eigen vector for $\lambda = -1$ is $\begin{pmatrix} x_1 \\ -x_1 \end{pmatrix} = \begin{pmatrix} c \\ -c \end{pmatrix} = c \begin{pmatrix} 1 \\ -1 \end{pmatrix}$
where c is non-zero real number.

Q) Find the Eigen Values & Eigen Vector, $A = \begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$

$$A - \lambda I = \begin{bmatrix} 1-\lambda & 1 & 1 \\ -1 & -1-\lambda & -1 \\ 0 & 0 & 1-\lambda \end{bmatrix}$$

$$\begin{aligned} \Rightarrow \det(A - \lambda I) &= 0 \\ \Rightarrow (1-\lambda)(\lambda^2-1) + 1(1-\lambda) &= 0 \\ \Rightarrow \lambda^2 - 1 - \lambda^3 + \lambda + 1 - \lambda &= 0 \\ \Rightarrow \lambda^2 - \lambda^3 &= 0 \\ \Rightarrow \lambda^2(1-\lambda) &= 0 \\ \Rightarrow \lambda &= 0, 0, 1 \end{aligned}$$

$$\begin{aligned} \Rightarrow x_1 + x_2 + x_3 &= 0 \text{ (i)} \\ -x_1 - x_2 - x_3 &= 0 \text{ (ii)} \\ x_3 &= 0 \text{ (iii)} \end{aligned}$$

Eigen vector corresponding to $\lambda = 0 \Rightarrow \begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

Putting $x_3 = 0$ in (i) $x_1 + x_2 = 0$
 $x_2 = -x_1$ or $x_1 = -x_2$
If we put $x_1 = c$ then $\begin{pmatrix} c \\ -c \\ 0 \end{pmatrix} = c \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}$

Eigen vector corresponding to $\lambda = 1 \Rightarrow \begin{bmatrix} 0 & 1 & 1 \\ -1 & -2 & -1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

$$\begin{aligned} x_2 + x_3 &= 0 \text{ (i)} \\ -x_1 - 2x_2 - x_3 &= 0 \text{ (ii)} \\ 0 &= 0 \text{ (iii)} \end{aligned}$$

$x_3 = -x_2$ or $x_2 = -x_3$
Putting x_3 in (ii), $-x_1 - 2x_2 + x_2 = 0$
 $-x_1 - x_2 = 0$

If we put $x_2 = c$ then $\begin{pmatrix} -c \\ c \\ -c \end{pmatrix} = c \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix}$

Chapter - 2 Fundamental of Probability

⇒ Sample Space: It is the entire set of possibilities for an experiment

Eg: For rolling a dice, $S = \{1, 2, 3, 4, 5, 6\}$ [occurrence of an event]

⇒ Event: It is the subset of Sample Space.

Eg: For getting an even no., $E = \{2, 4, 6\}$.

⇒ Probability of an event: $P(E) = \frac{n(E)}{n(S)} = \frac{3}{6} = \frac{1}{2}$

⇒ Properties of Probability:

(i) Outcomes - $O_1, O_2, \dots, O_n \rightarrow$ Probability of outcomes, $P(O_1) + P(O_2) + \dots + P(O_n) = 1$

$$\sum P(O) = 1$$

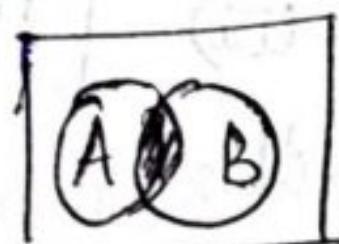
(ii) If A & B are events & $A \subseteq B$, then $P(A) \leq P(B)$.



(iii) $P(A^c) = 1 - P(A)$



(iv) $P(A \cup B) = P(A) + P(B) - P(A \cap B)$



⇒ Conditional Probability:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

(probability of A given B)
[B occurs earlier
A occurs later]

Eg: 3 coin is tossed

HHH
HHT
HTH
HTT
TTT
TTH
THT
THH

B: H in 1st trial
A: Even no. of H
 $P(A) = \frac{3}{8}$ $P(B) = \frac{4}{8} = \frac{1}{2}$
 $P(A \cap B) = \frac{2}{8}$

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$= \frac{2/8}{1/2} = \frac{4}{8} = \frac{1}{2}$$

* If A & B are independent events, $P(A \cap B) = P(A) \cdot P(B)$

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A) \cdot P(B)}{P(B)} = P(A)$$

04-09-25

Q) Total Student = 50, Girls = 30, Total no. of students passed = 35 out of which 20 are girls.
Find the probability of the event "Randomly select a student & given that the student is passed & is a girl".

Ans: $P(G) = \frac{30}{50} = \frac{3}{5}$

$$P(\text{Girl} | \text{Student who passed}) = \frac{P(G \cap Q)}{P(Q)}$$

$$= \frac{20}{35} = \frac{4}{7}$$

$$P(\text{Student who passed}) = \frac{35}{50}$$

$$P(\text{Girl} \cap \text{Student who passed}) = \frac{20}{50} = \frac{2}{5}$$

$$\begin{aligned} \text{Baye's Theorem: } & \left[P(A|B) = \frac{P(AB)}{P(B)} \right] \quad \left[P(B|A) = \frac{P(AB)}{P(A)} \right] \\ \Rightarrow P(AB) &= P(A|B) \cdot P(B) \quad \Rightarrow P(AB) = P(B|A) \cdot P(A) \\ \Rightarrow P(A|B) \cdot P(B) &= P(B|A) \cdot P(A) \\ \Rightarrow \boxed{P(B|A) = \frac{P(A|B) \cdot P(B)}{P(A)}} \end{aligned}$$

Eg: Spam filter
 Given: 1% emails are spam. If an email has a word "free" & is a spam then the probability is 0.8. Probability that email contains word "free" & is not spam is 0.1. What is the probability of email being spam given that it contains word "free".

$P(\text{"free"}|\text{spam}) = 0.8$

$P(\text{"free"}|\neg\text{spam}) = 0.1$

$P(\text{spam}|\text{"free"}) = ?$

$$P(\text{spam}|\text{"free"}) = \frac{P(\text{"free"}|\text{spam}) P(\text{spam})}{P(\text{"free"})}$$
 $[P(\text{spam}) = 1\% = 0.01]$

$$= \frac{P(\text{"free"}|\text{spam}) P(\text{spam})}{P(\text{"free"}|\text{spam}) P(\text{spam}) + P(\text{"free"}|\neg\text{spam}) \cdot P(\neg\text{spam})}$$

$$= \frac{0.8 \times 0.01}{0.8 \times 0.01 + 0.1 \times 0.99} = \underline{\underline{0.0748}}$$

\Rightarrow Random Variable:

$X : S \rightarrow R$
 Random variable Sample space Real number

Eg: X : # heads
 y : # tails

$$X(HHH) = 3$$

$$Y(HHH) = 0$$

$$Y(HTH) = 1$$

\Rightarrow Probability Density function/Mass function:

$$f(x) = \text{Prob}[X=x] \quad x = 0, 1, 2, 3$$

\Rightarrow Probability Distribution function:

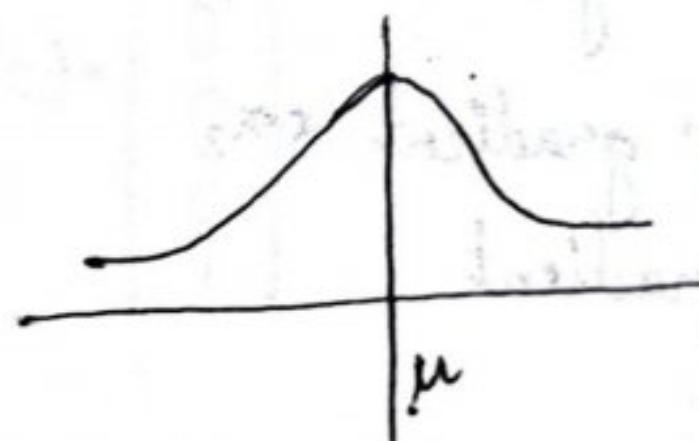
$$F(x) = \text{Prob}[X \leq x]$$

$$F(x) = \int_{-\infty}^x f(x) dx$$

$$= \sum_{i=-\infty}^{\infty} f(x) \quad [\text{distinct value}]$$

\Rightarrow Normal Distribution:

$$F(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



\Rightarrow Expectation:

$$E(x) = \sum x P(X=x)$$

Eg: $X = \# \text{Heads}$ (coin tossed)

$$X=0, X=1$$

$$E(x) = 0 \times P(X=0) + 1 \times P(X=1)$$

$$= 0 + 1 \times \frac{1}{2}$$

$$= \underline{\underline{\frac{1}{2}}}$$

Eg: $X = \# \text{6 (dice rolled)}$
 $X=0, X=1$

$$E(x) = 0 \times P(X=0) + 1 \times P(X=1)$$

$$= 0 \times \frac{5}{6} + 1 \times \frac{1}{6}$$

$$= \underline{\underline{\frac{1}{6}}}$$

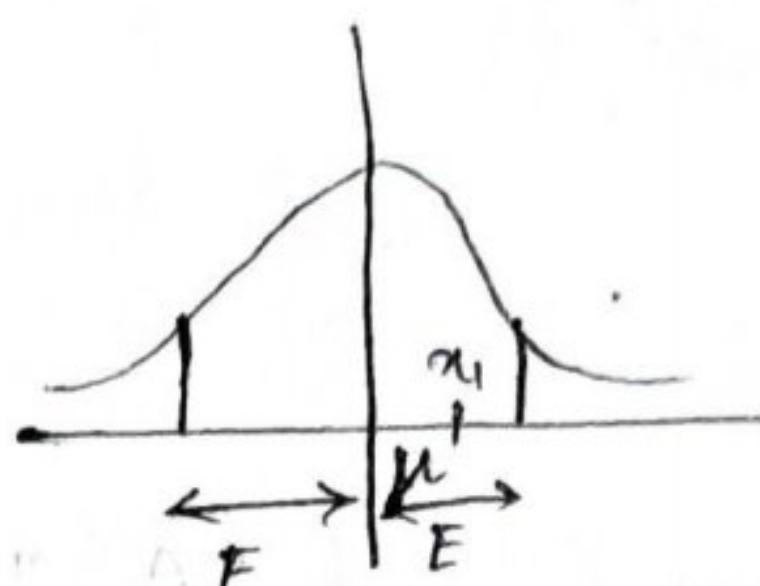
\Rightarrow Properties of Expectation:

i) $E(c) = c$ [c=constant]

$$E(X) = \mu \quad [\text{Mean} = \frac{x_1 + x_2 + \dots + x_n}{n}]$$

ii) $E(cx) = cE(x)$

iii) $E(x_1 + x_2) = E(x_1) + E(x_2)$



\Rightarrow Variance:

$$\boxed{\text{Var}(x) = E[(x-\mu)^2]}$$

$$= E[x^2 - 2\mu x + \mu^2]$$

$$= E(x^2) + E(-2\mu x) + E(\mu^2) \quad [\mu = E(x)]$$

$$= E(x^2) + [-2\mu \cdot E(x)] + \mu^2$$

$$= E(x^2) - 2\mu^2 + \mu^2 \quad (\because E(x) = \mu)$$

$$= E(x^2) - \mu^2$$

$$= E(x^2) - [E(x)]^2$$

\Rightarrow Co-Variance:

$$\boxed{\text{cov}(x,y) = E(xy) - E(x) \cdot E(y)}$$

$$\text{var}(x+y) = \text{var}(x) + \text{var}(y) + 2\text{cov}(x,y)$$

\Rightarrow Activation $(x_1 w_1 + x_2 w_2 + b) \geq 0$ then 1 otherwise 0

$$\text{Error} = y - \hat{y}$$

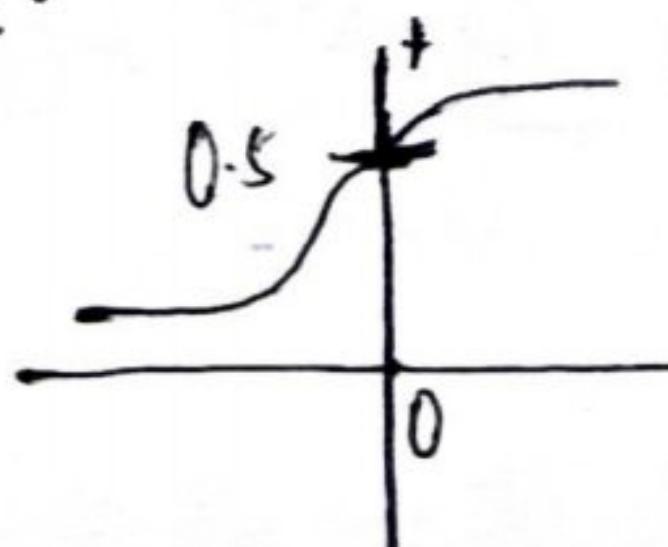
$$\text{Gradient} = \text{Error} \times \text{learning rate} \quad \text{positive constant}$$

$$w_{1,\text{new}} = w_{1,\text{old}} + \text{gradient} \cdot x_1$$

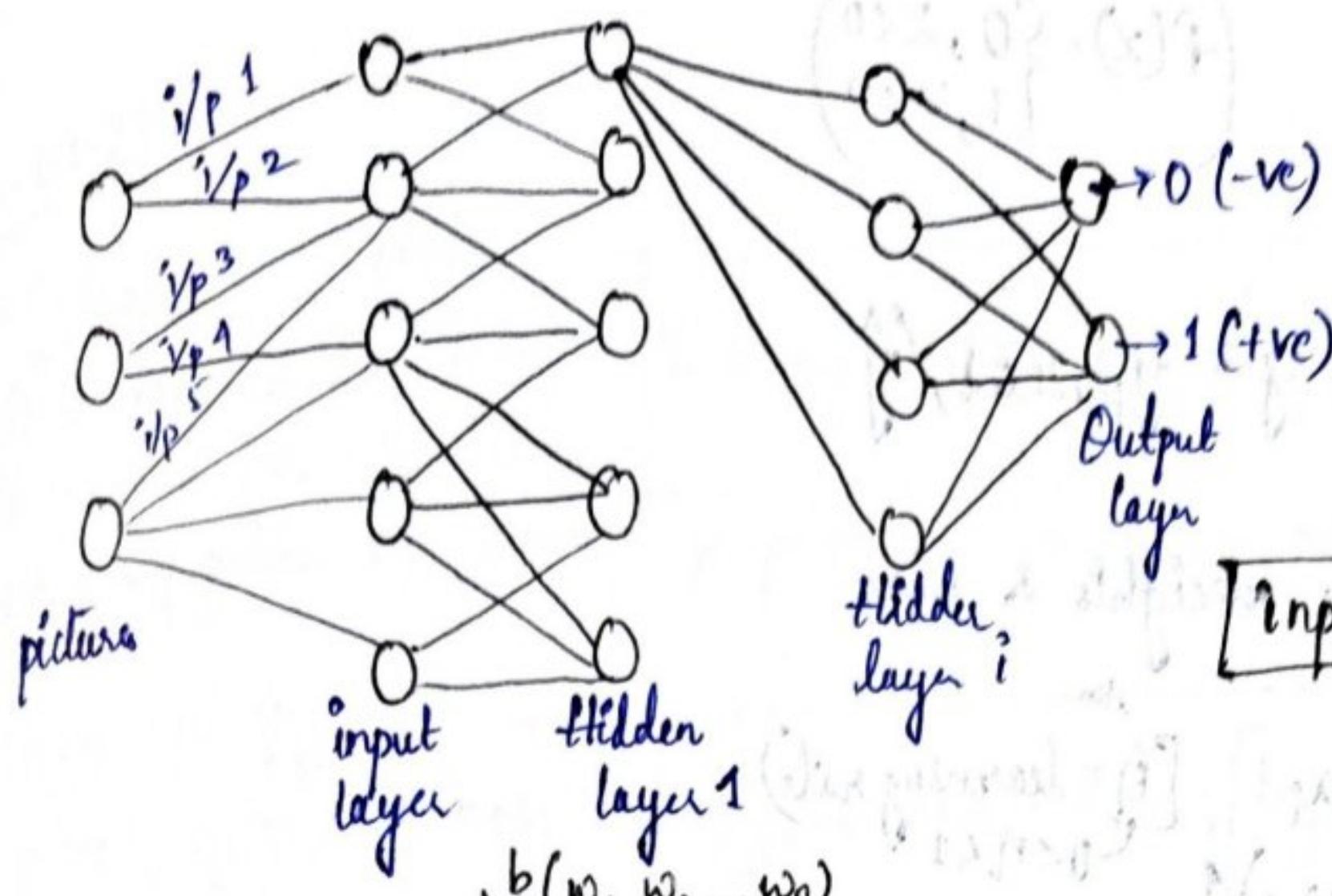
$$w_{2,\text{new}} = w_{2,\text{old}} + \text{gradient} \cdot x_2$$

$$b = b + \text{gradient}$$

\Rightarrow Sigmoid Function:

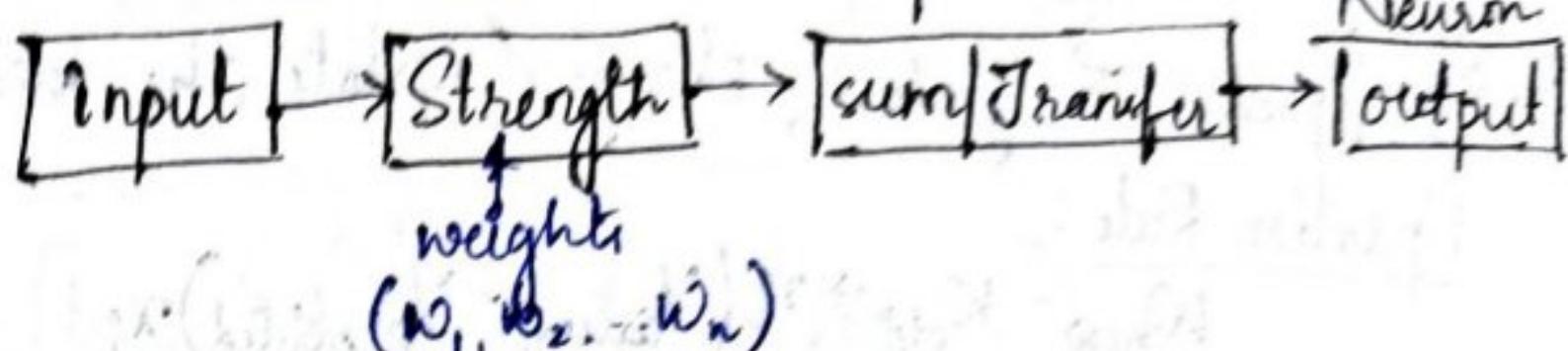
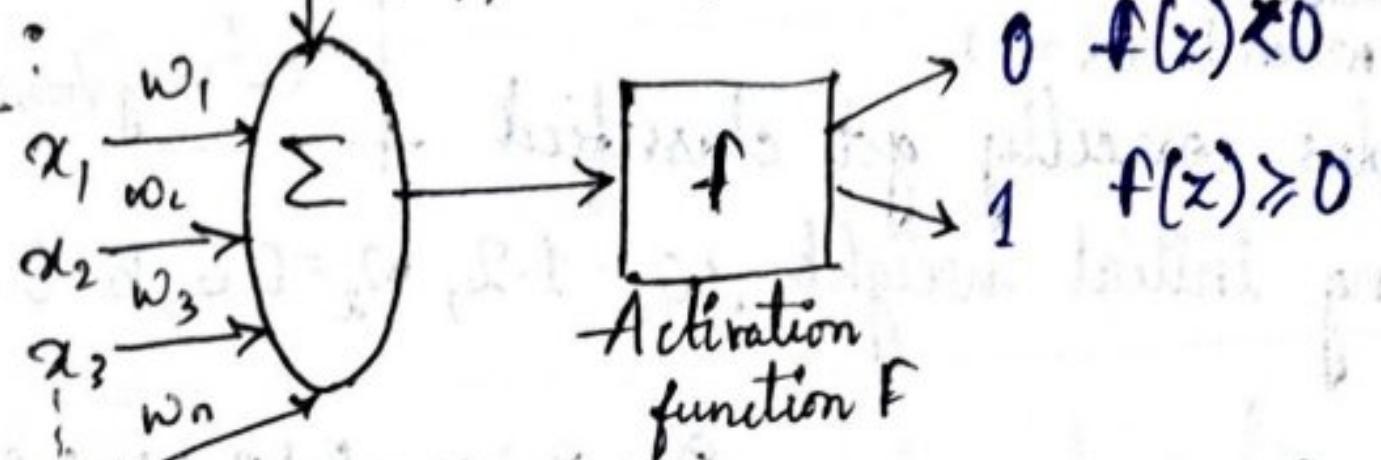


08-09-25

Chapter 3Building Block of DL → ANNFeed-Forward Neural Network

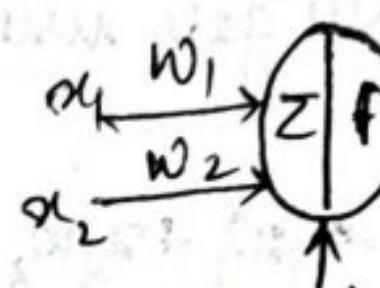
Activation Function: Activates the nodes of hidden layer as per requirement of the selection of the function.

The nodes are called perceptrons / Artificial Neuron

Perceptron:

$$z = b + w_1 x_1 + w_2 x_2 + \dots + w_n x_n$$

$$w_1 x_1 + w_2 x_2 + b = 0$$

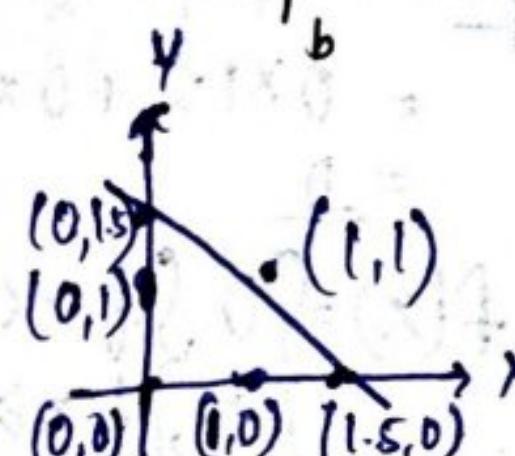


Eg: $w_1 = 1, w_2 = 1, b = -1.5$, AND Gate

x ₁	x ₂	y
0	0	0
0	1	0
1	0	0
1	1	1

$$z = -1.5 + 1.x_1 + 1.x_2$$

$$f(z) = \begin{cases} 1, & f(z) \geq 0 \\ 0, & f(z) < 0 \end{cases}$$



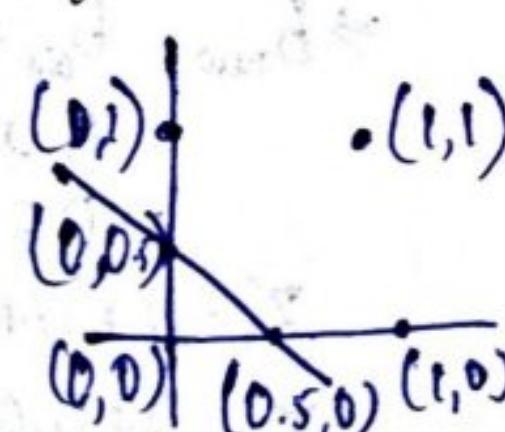
$$f(z) = z = \begin{cases} -1.5 + 0 + 0 = -1.5 < 0 \\ -1.5 + 1 \times 0 + 1 \times 1 = -0.5 < 0 \\ -1.5 + 1 \times 1 + 0 \times 1 = -0.5 < 0 \end{cases}$$

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ b & x_1 & x_2 \end{bmatrix}$$

$$x = XW^T = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}_{4 \times 3} \begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix}_{3 \times 1} = \begin{bmatrix} -1.5 \\ -0.5 \\ -0.5 \\ 0.5 \end{bmatrix} \rightarrow f(z) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Eg: Above eg in OR Gate

$$\begin{aligned} x_1 + x_2 - 0.5 &= 0 \\ [w_1 x_1 + w_2 x_2 + b = 0] \\ x = -0.5 + 1 \cdot x_1 + 1 \cdot x_2 \end{aligned}$$

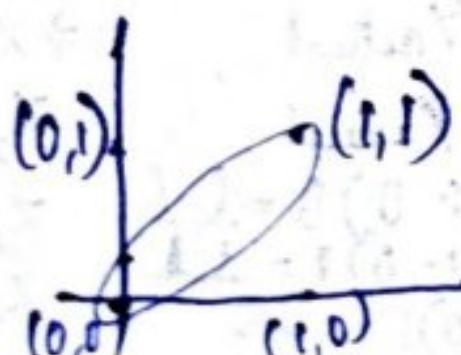


$$f(z) = \begin{cases} 1, & \text{if } f(z) \geq 0 \\ 0, & \text{if } f(z) < 0 \end{cases}$$

$$f(z) = xW^T \text{ for AND & OR gate.}$$

Eg: Above eg in XOR Gate

x ₁	x ₂	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0



We cannot get linear graph for XOR gate.
So, we can't find perceptron for XOR gate..

$$W = \begin{bmatrix} -0.5 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

$$xW^T = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 0.5 \\ 0.5 \\ 1.5 \end{bmatrix} \rightarrow f(z) = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

10-09-25

Perception training Rule

1. Randomly choose initial weights & bias.
2. Find the z . ($z = b + \sum_{i=1}^n x_i w_i = \sum_{i=1}^n x_i w_i$)

$$f(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$$

3. Apply the activation function on z to get $y_{predicted}$ / \hat{y} .

4. Compare $y_{predicted}$ with y_{actual} .

5. If $y_{predicted} \neq y_{actual}$, change/update the weights & bias.

Updation Rule:

$$w_{new} = w_{old} + \eta (y_{actual} - y_{predicted}) x_i \quad [\eta = \text{learning rate}, 0 < \eta < 1]$$

$$\text{bias}_{new} = \text{bias}_{old} + \eta (y_{actual} - y_{predicted}) \quad \text{when } b = x_0 \text{ & } x_0 = 1$$

6. This process continued till all the samples correctly got classified.

- Q) Design a perception for AND gate using initial weight, $w_1 = 1.2$, $w_2 = 0.6$, $b = 0$. threshold.

x ₁	x ₂	y
0	0	0
0	1	0
1	0	0
1	1	1

Truth table for AND gate.

Step 1 : $z = x_1 w_1 + x_2 w_2 + b$
 $= 0 \times 1.2 + 0 \times 0.6 + 0$
 $= 0$
 $f(z) = \begin{cases} 0, & \text{if } z < 1 \\ 1, & \text{if } z \geq 1 \end{cases}$
 $y_{predicted} = 0 = y_{actual} (\because z < 1)$
 $\Rightarrow \text{No weight updation.}$

Step 2 : $z = 0 \times 1.2 + 1 \times 0.6 + 0$
 $= 0.6$

$y_{predicted} = 0 = y_{actual} (\because z < 1)$

\Rightarrow No weight updation.

Step 3 : $z = 1 \times 0.2 + 0 \times 0.6 + 0$

$z = 0.2$
 $y_{predicted} = 1 \neq y_{actual} (\because z > 1.2)$

$\Rightarrow w_{1,new} = w_{1,old} + \eta (y_{actual} - y_{pred}) x_1$

$= 1.2 + 0.5(0-1)$
 $= 0.7$

$\Rightarrow w_{2,new} = w_{2,old} + \eta (y_{actual} - y_{pred}) x_2$

$= 0.6$

$\Rightarrow b_{new} = b_{old} + \eta (y_{actual} - y_{pred})$

$= 0 + 0.5(0-1)$
 $= -0.5$

$z_1 = 0 + 0 + (-0.5) = -0.5 < 1, \hat{y} = 0 = y_{actual}$

$z_2 = 0 + 0.6 - 0.5 = 0.1 < 1, \hat{y} = 0 = y_{actual}$

$z_3 = 0.7 \times 1 + 0 \times 0.6 - 0.5 = 0.2 < 1, \hat{y} = 0 = y_{actual}$

\Rightarrow No updation.

Step 4 : $z = 0.7 \times 1 + 0.6 \times 1 = 0.5$
 $= 0.8 < 1$

$y_{predicted} = 0 \neq y_{actual}$

$\Rightarrow w_{1,new} = 0.7 + 0.5(1-0)$

$= 1.2$

$\Rightarrow w_{2,new} = 0.6 + 0.5(1-0)$

$= 1.1$

$\Rightarrow b_{new} = -0.5 + 0.5(1-0)$

$= 0$

$z_1 = 0 \Rightarrow y_{predicted} = y_{actual}$

$z_2 = 1.1 \Rightarrow y_{predicted} \neq y_{actual}$

\Rightarrow Updation needed.

$w_{1,new} = 1.2 + 0.5(0-1) = 0.7 < 0, y_{predicted} \neq y_{actual}$

- Q) OR gate using $w_1 = 0.6$, $w_2 = 0.6$, $b = 0$

$$f(z) = \begin{cases} 0, & \text{if } z < 1 \\ 1, & \text{if } z \geq 1 \end{cases}$$

x_1	x_2	$z = x_1 + x_2$
0	0	0
0	1	1
1	0	1
1	1	2

$\Rightarrow z = 0 \Rightarrow y_{predicted} = y_{actual}$

$\Rightarrow z = 0.6 \Rightarrow y_{predicted} \neq y_{actual}$

$\Rightarrow w_{1,new} = 0.6 + (1-0)0 = 0.6$

$\Rightarrow w_{2,new} = 0.6 + 0(1-0)1 = 0.6$

$\Rightarrow b_{new} = 0 + 0(1-0)1 = 0.5$

$\Rightarrow z_1 = 0.5 \Rightarrow y_{predicted} \neq y_{actual}$

$\Rightarrow z_2 = 1.1 + 0.5 = 1.6 \Rightarrow y_{predicted} \neq y_{actual}$

$\Rightarrow z_3 = 0.6 + 0.5 = 1.1 \Rightarrow y_{predicted} \neq y_{actual}$

$\Rightarrow z_4 = 1.1 + 0.6 + 0.5 = 2.2 \Rightarrow y_{predicted} \neq y_{actual}$

$$\alpha_2 \xrightarrow{1.1} \left(\sum_{i=1}^3 f_i \right) \alpha_1$$

$$0.8\alpha_1 + 1.1\alpha_2 + 0.5 = 0$$

11-09-25

\Rightarrow Entropy: $p(x_i)$

It is denoted by $H(x)$.

It is used for calculating

back propagation or forward propagation.

$$\begin{aligned} \text{Entropy: } H(x) &= E_{p(x)} \left(\log_2 \frac{1}{p(x)} \right) \\ &= \sum_i p(x_i) \log_2 \frac{1}{p(x_i)} \\ &= -E \left(\log_2 p(x) \right) \\ &= -\sum_i (p(x_i) \log_2 p(x_i)) \end{aligned}$$

$$E(x) = \sum_{x_i} p(x_i)$$

\Rightarrow Cross Entropy:

$p(x_i), q(x_i)$
[2 distribution function]

$$\begin{aligned} \text{Cross Entropy, } CE(p||q) &= E_{p(x)} \left[\log_2 \frac{1}{q(x)} \right] \\ &= \sum_i (p(x_i) \log_2 \frac{1}{q(x_i)}) \\ &= -\sum_i p(x_i) \log_2 q(x_i) \end{aligned}$$

\Rightarrow KL divergence:

$p(x)$
[2 distribution function]

$$KL(p||q) = E_{p(x)} \left[\log_2 \frac{1}{q(x)} - \log_2 \frac{1}{p(x)} \right]$$

$$KL(p||q) = E_{p(x)} \left[\log \frac{p(x)}{q(x)} \right]$$

$$KL(p||q) = \sum_i \left[p(x_i) \log_2 \frac{p(x_i)}{q(x_i)} \right]$$

Q) 3 classes - X, Y, Z, $p(X) = 0.7$, $p(Y) = 0.2$, $p(Z) = 0.1$. Compute Entropy of the distribution.

$$\text{Entropy} = -\sum p(x_i) \log_2 p(x_i)$$

$$= [p(X) \log_2 p(X) + p(Y) \log_2 p(Y) + p(Z) \log_2 p(Z)]$$

$$= [0.7 \log_2 (0.7) + 0.2 \log_2 (0.2) + 0.1 \log_2 (0.1)]$$

Q) $P(X) = 0.7 \quad | \quad q(X) = 0.6$ Compute Cross Entropy & KL divergence.

$$\begin{array}{ll} P(Y) = 0.2 & | \quad q(Y) = 0.3 \\ P(Z) = 0.1 & | \quad q(Z) = 0.1 \end{array}$$

$$CE(p||q) = -\sum p(x_i) \log_2 q(x_i)$$

$$= - [0.7 \times \log_2 0.6 + 0.2 \times \log_2 0.3 + 0.1 \times \log_2 0.1]$$

$$KL(p||q) = \sum \left[p(x_i) \log_2 \frac{p(x_i)}{q(x_i)} \right]$$

$$= 0.7 \times \log_2 \frac{p(x_i)}{q(x_i)} + 0.2 \log_2 \frac{p(x_i)}{q(x_i)} + 0.1 \log_2 \frac{p(x_i)}{q(x_i)}$$

$$= 0.7 \times \log_2 \left(\frac{7}{6} \right) + 0.2 \times \log_2 \left(\frac{2}{3} \right) + 0.1 \log_2 1$$

Q) Consider a perceptron of 3 inputs $x_1=1, x_2=1, x_3=0$, the weights are $w_1=2, w_2=-4, w_3=1$ & activation function is step function. Find the output value Y of the perceptron.

$$b=0, z = x_1w_1 + x_2w_2 + x_3w_3 = 2 - 4 + 0 = -2$$

$$f(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

$$Y = f(z) = 0$$

→ Activation function

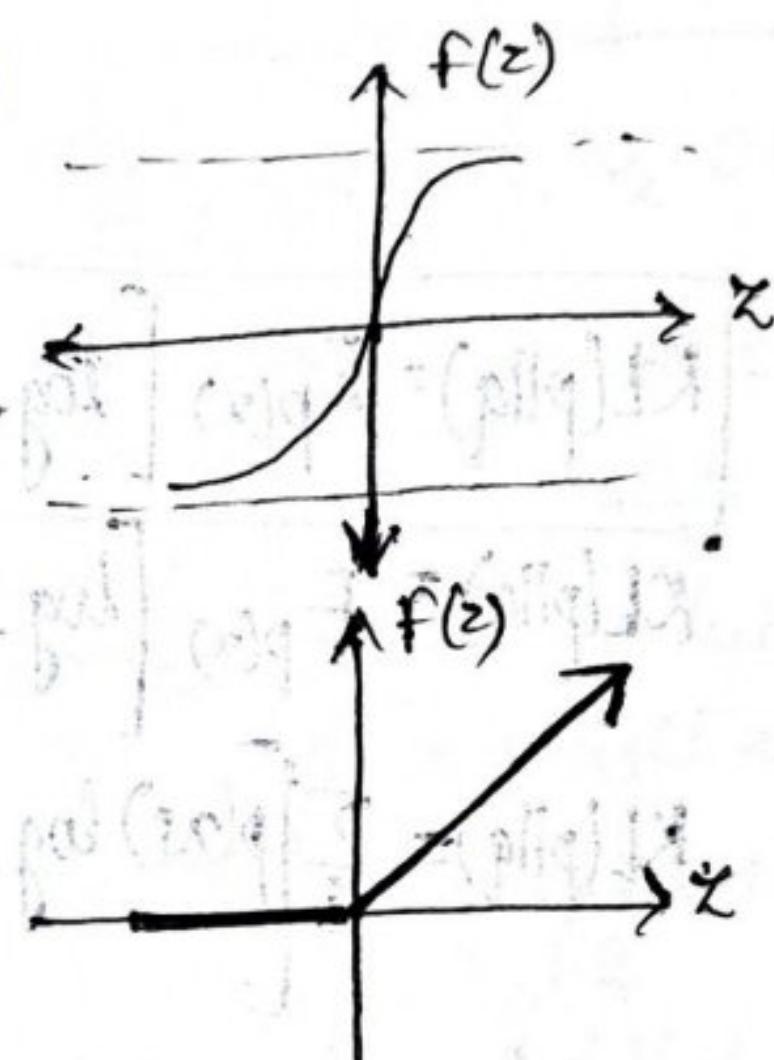
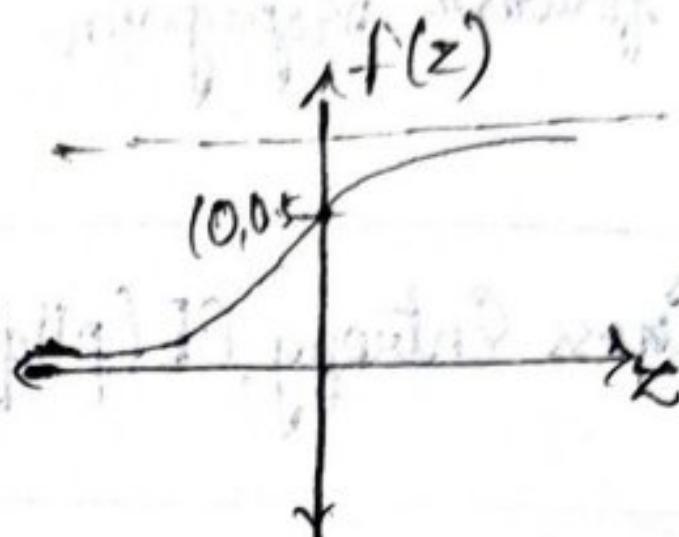
- i) Step function
- ii) Sigmoid function

- iii) Tanh function
- iv) ReLU function

→ Sigmoid function: $f(z) = \frac{1}{1+e^{-z}}, 0 < f(z) < 1$

→ Tanh function: $f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, -1 < f(z) < 1$

→ ReLU function: $f(z) = \max\{0, z\}, f(z) = \begin{cases} 0, & \text{if } z < 0 \\ z, & \text{if } z \geq 0 \end{cases}$



13-09-25

→ Entropy: It is the quantitative assessment of the unpredictability of the distribution of your class labels. We can use it to understand how evenly or unevenly your data points are distributed across different classes. The higher the entropy, the harder it will be for our supervised ML algorithm to make accurate prediction. (less is better)

$$H(x) = E_{p(x)} \left[\log_2 \frac{1}{p(x)} \right]$$

→ Cross Entropy: It is used as the loss function that quantifies how well or poorly a model performs. It measures the discrepancy between the predicted values & the actual values from the dataset. If the model predicts a probability that is far from the actual value, the cross-entropy value will be large, indicating a poor prediction. (less is better)

$$CE(p||q) = E_{p(x)} \left[\log_2 \frac{1}{q(x)} \right]$$

$p(x)$ - actual distribution
 $q(x)$ - predicted distribution

KL-divergence: It is defined as the number of bits required to convert one distribution into another distribution. The lower bound value is 0 & is achieved when the distributions are identical.

$$KL(p||q) = \underset{\text{Expectation}}{E_{p(x)}} \left[\log_2 \frac{1}{q(x)} - \log_2 \frac{1}{p(x)} \right]$$

$$\cdot E_{p(x)} \left[\log_2 \frac{1}{q(x)} \right] - E_{p(x)} \left[\log_2 \frac{1}{p(x)} \right] \quad \because E(X+Y) = E(X) + E(Y)$$

$$KL(p||q) = CE(p||q) - H(p)$$

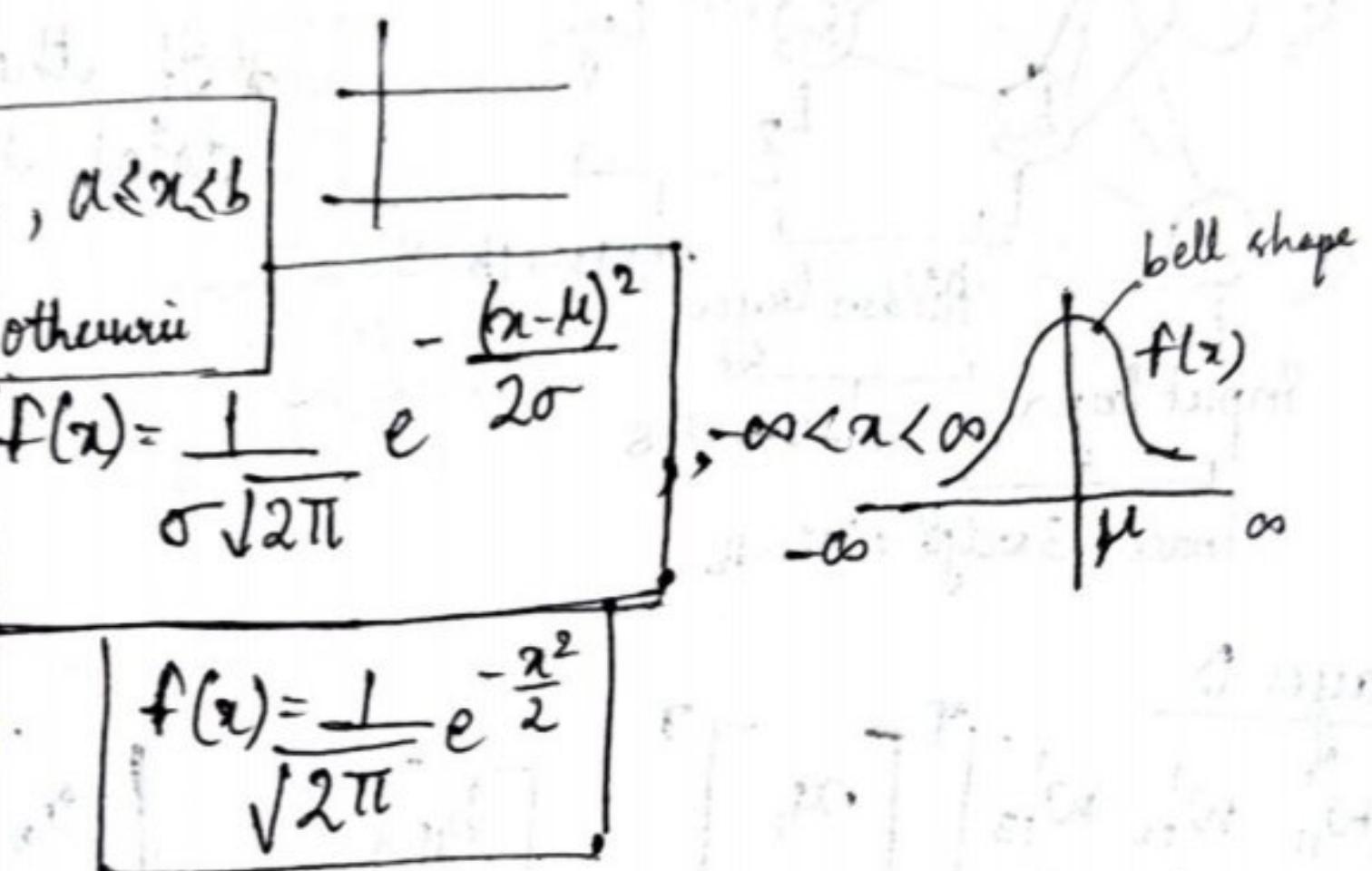
→ Continuous distribution function:

1. Uniform distribution function:

$$f(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & \text{otherwise} \end{cases}$$

2. Normal distribution function (Gaussian df):

μ = mean
 σ^2 = Variance
 If $\mu=0$ & $\sigma^2=1$, then it is called Standard Normal distribution.



$$E(x) = \int_a^b x f(x) dx \quad [\text{Expectation}]$$

$$H(f(x)) = - \int_a^b f(x) \log_2 f(x) dx \quad [\text{Entropy}]$$

$$\text{Var}(x) = E[(x-\mu)^2] \quad [\text{Variance}]$$

$$CE(f(x)||g(x)) = - \int_a^b f(x) \log_2 g(x) dx \quad [\text{Cross Entropy}]$$

$$= \int_a^b (x-\mu)^2 f(x) dx$$

$$KL(f(x)||g(x)) = \int_a^b f(x) \log_2 \left(\frac{f(x)}{g(x)} \right) dx \quad [\text{KL-divergence}]$$

Q) $f(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & \text{otherwise} \end{cases}$. Find $E(x)$, $\text{var}(x)$.

$$E(x) = \int_a^b x f(x) dx = \int_a^b x \frac{1}{b-a} dx = \frac{1}{b-a} \int_a^b x dx = \frac{1}{b-a} \left[\frac{x^2}{2} \right]_a^b = \frac{1}{2(b-a)} \cdot b^2 - a^2 = \frac{(b+a)(b-a)}{2(b-a)} = \frac{b+a}{2}$$

$$\mu = E(x) = \frac{b+a}{2}$$

$$\text{var}(x) = E[(x-\mu)^2] = \int_a^b (x-\mu)^2 f(x) dx = \int_a^b (x^2 + \mu^2 - 2x\mu) \cdot \frac{1}{b-a} dx = \frac{1}{b-a} \left[\frac{x^3}{3} + \mu^2 x - \frac{2\mu x^2}{2} \right]_a^b$$

$$= \frac{1}{3(b-a)} [x^3]_a^b + \mu^2 [x]_a^b - \mu [x^2]_a^b$$

Q) $f(x) = \begin{cases} 1, & \text{if } 0 \leq x \leq 1 \\ 0, & \text{otherwise} \end{cases}$

$$g(x) = \begin{cases} 1/2, & \text{if } 0 \leq x \leq 2 \\ 0, & \text{otherwise} \end{cases}$$

$$H(g(x)), CE(f(x)||g(x)), KL(f(x)||g(x))$$

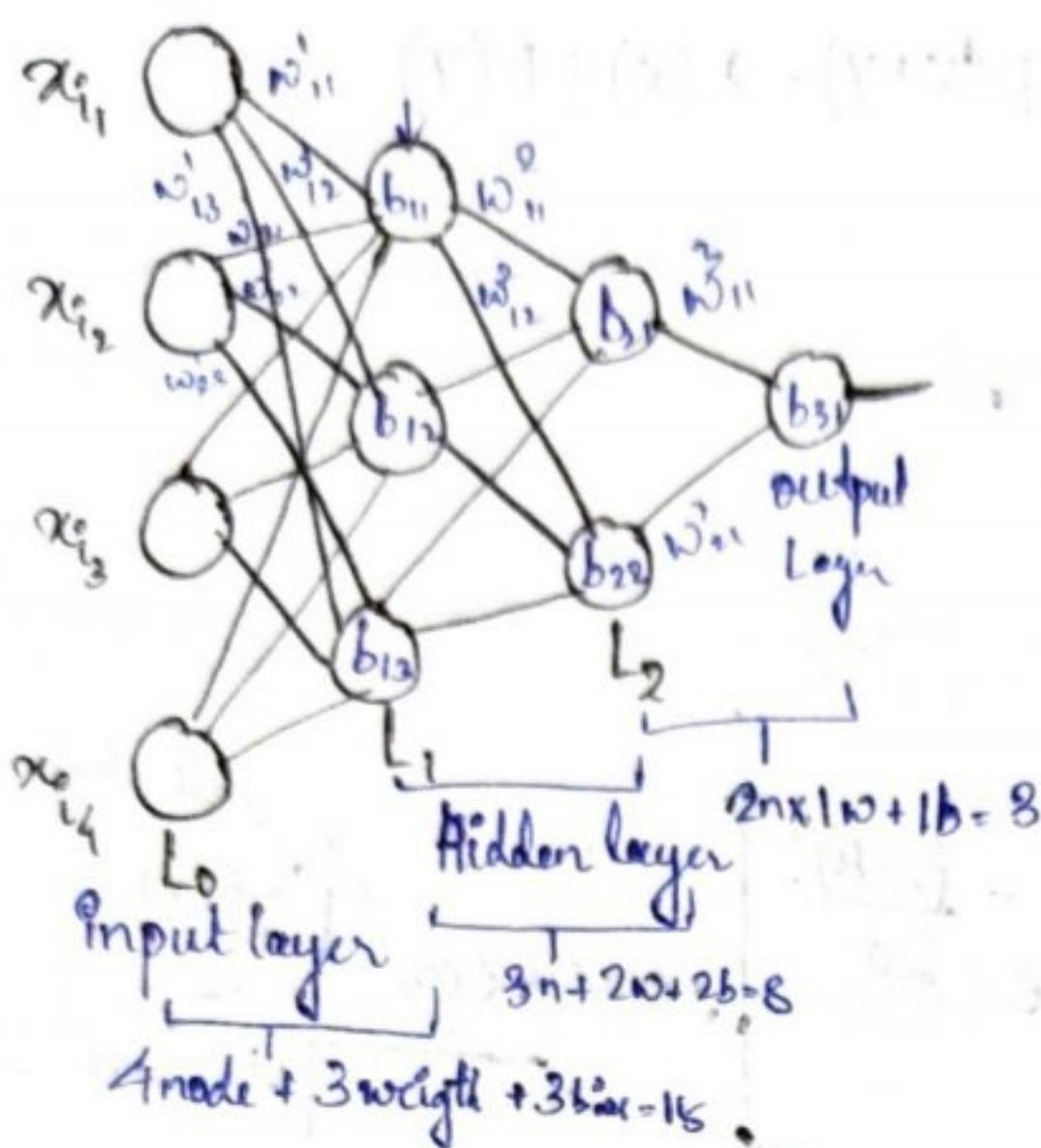
$$\Rightarrow H(g(x)) = - \int \frac{1}{2} \log_2 \left(\frac{1}{2} \right) dx = -\frac{1}{2} (-1) [x]_0^\infty = \frac{1}{2} [2-0] = 1$$

$$CE(f(x)||g(x)) = - \int_a^b 1 \cdot \log_2 \left(\frac{1}{2} \right) dx = - [x]_a^\infty (-1)$$

15-09-25

Chapter-4

→ Multi layer Perceptron: Single layer Perceptron can only deal with linear separability. Capacity of single layer perceptron is not there to deal with non-linear separability. So, we need Multi-layer perceptron or multiple neurons.



$Age(x_1)$	$ig(x_2)$	$attendence(x_3)$	$Hours(x_4)$
$8x_{11}$	$1x_{21}$	$5x_{31}$	6
$9x_{12}$	$8x_{22}$	$3x_{32}$	8
$5x_{13}$	$5x_{23}$	$3x_{33}$	3

$$\text{Total learnable parameter} = 15 + 18 + 3 = 36$$

* If there are 'n' no. of nodes & 'm' no. of neurons,
total learnable parameter = $[n \times m + m]$

Layer 0

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix}^T \begin{bmatrix} x_{i_1} \\ x_{i_2} \\ x_{i_3} \\ x_{i_4} \end{bmatrix} + \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix}_{3 \times 1} = \begin{bmatrix} o_{11} \\ o_{12} \\ o_{13} \end{bmatrix} \Rightarrow \begin{bmatrix} w_{11}x_{i_1} + w_{12}x_{i_2} + w_{13}x_{i_3} + w_{41}x_{i_4} \\ w_{21}x_{i_1} + w_{22}x_{i_2} + w_{23}x_{i_3} + w_{42}x_{i_4} \\ w_{31}x_{i_1} + w_{32}x_{i_2} + w_{33}x_{i_3} + w_{43}x_{i_4} \end{bmatrix} + \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix} = \begin{bmatrix} o_{11} \\ o_{12} \\ o_{13} \end{bmatrix}$$

no. of input no. of neurons in hidden layer

Layer 1

$$\begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \\ w_{31}^2 & w_{32}^2 & w_{33}^2 \end{bmatrix} \begin{bmatrix} o_{11} \\ o_{12} \\ o_{13} \end{bmatrix} + \begin{bmatrix} b_{21} \\ b_{22} \end{bmatrix} = \begin{bmatrix} o_{21} \\ o_{22} \end{bmatrix}$$

Layer 2

$$\begin{bmatrix} w_{11}^3 & w_{21}^3 \end{bmatrix}^T \begin{bmatrix} o_{21} \\ o_{22} \end{bmatrix} + b_{31} = o_{31} \rightarrow \text{way of forward propagation}$$

$$\Rightarrow o_{31} = w_{11}^3 o_{21} + w_{21}^3 o_{22} + b_{31}$$

$$a^{[1]} = a [w^{[1]}. a^{[0]} + b^{[1]}]$$

$$a^{[2]} = a [w^{[2]}. a^{[1]} + b^{[2]}]$$

$$a^{[3]} = a [w^{[3]}. a^{[2]} + b^{[3]}]$$

$$a^{[1]} = w^{[1]}. a^{[0]} + b^{[1]}$$

$$a^{[2]} = w^{[2]}. a^{[1]} + b^{[2]}$$

$$a^{[3]} \rightarrow o_{31} = a \{ w^{[3]}. a (w^{[2]}. a^{[1]} + b^{[2]}) + b^{[3]} \}^2$$

$$= a \{ w^{[3]}. a (w^{[2]}. a (w^{[1]}. a^{[0]} + b^{[1]}) + b^{[2]}) + b^{[3]} \}^2$$

So, multilayer perception is a cascade of functions

$$a^{[2]} = w^{[2]} (w^{[1]}. a^{[0]} + b^{[1]}) + b^{[2]}$$

$$= w^{[2]}. w^{[1]}. a^{[0]} + w^{[2]}. b^{[1]} + b^{[2]}$$

$$= w^{[2]}. a^{[0]} + b^{[2]}$$

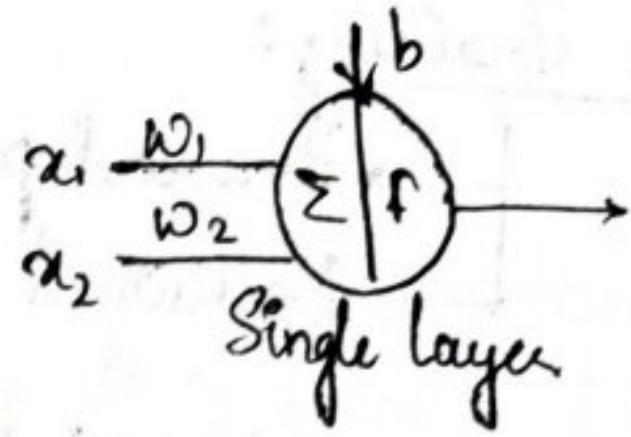
Root mean square

$$\sqrt{\frac{\sum (Y_{\text{pred}} - Y_{\text{actual}})^2}{n}}$$

* Without using objective functions in the hidden layers, it becomes linear.
So, we need something non-linear.
Based on the example derived, no matter how many layers we use, we shall be getting similar kind of equations. So, we need to use non-linear obj. func to get benefit of different hidden layers.

17-09-25

Multi-layer Perception: Used for non-linear separation.

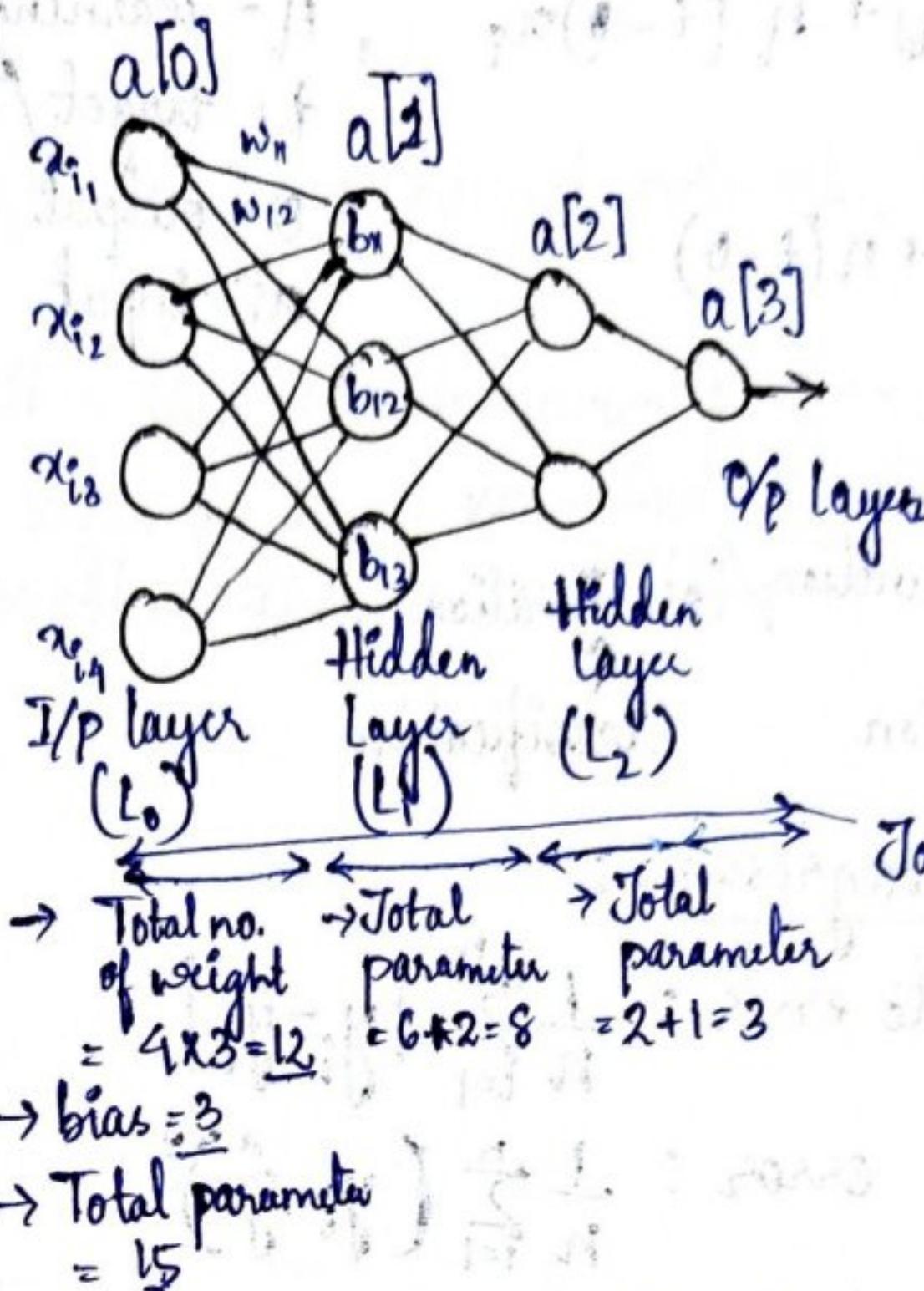


Ex:

	x_1	x_2	x_3	x_4	y
x_{11}	8	$7x_{21}$	$5x_{31}$	9	1
x_{12}	9	$8x_{22}$	$6x_{32}$	8	1
x_{13}	5	$5x_{23}$	$3x_{33}$	4	0

w_{ij}^k = weight of edge connected by i^{th} node in $(k-1)^{th}$ layer & j^{th} node in k^{th} layer

b_{ij} = bias of j^{th} node in i^{th} layer



$$\text{Total learnable parameters} = 15 + 8 + 3 = \underline{\underline{26}}$$

$$\rightarrow \text{bias} = 3$$

$$\rightarrow \text{Total parameters} = 15$$

For layer 1:

$$\begin{bmatrix} w_{11}^1 & w_{21}^1 & w_{31}^1 & w_{41}^1 \\ w_{12}^1 & w_{22}^1 & w_{32}^1 & w_{42}^1 \\ w_{13}^1 & w_{23}^1 & w_{33}^1 & w_{43}^1 \end{bmatrix}_{3 \times 4} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \end{bmatrix}_{4 \times 1} + \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix} = \begin{bmatrix} O_{11} \\ O_{12} \\ O_{13} \end{bmatrix}$$

For layer 2:

$$\begin{bmatrix} w_{11}^2 & w_{21}^2 & w_{31}^2 \\ w_{12}^2 & w_{22}^2 & w_{32}^2 \end{bmatrix} \begin{bmatrix} O_{11} \\ O_{12} \\ O_{13} \end{bmatrix} + \begin{bmatrix} b_{21} \\ b_{22} \end{bmatrix} = \begin{bmatrix} O_{21} \\ O_{22} \end{bmatrix}$$

For layer 3:

$$\begin{bmatrix} w_{11}^3 \\ w_{21}^3 \end{bmatrix} \begin{bmatrix} O_{21} \\ O_{22} \end{bmatrix} + b_{21} = O_{31}$$

$$\begin{aligned} a^{[1]} &= f(w^{[1]} a^{[0]} + b^{[1]}) \\ a^{[2]} &= f(w^{[2]} a^{[1]} + b^{[2]}) \\ a^{[3]} &= f(w^{[3]} a^{[2]} + b^{[3]}) \end{aligned}$$

$a_i = f(w^{[3]} + f(w^{[2]} + f(w^{[1]} a^{[0]} + b^{[1]}) + b^{[2]})) \rightarrow$ So, multi-layer perception can be written as cascade of function.

$$a^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$$

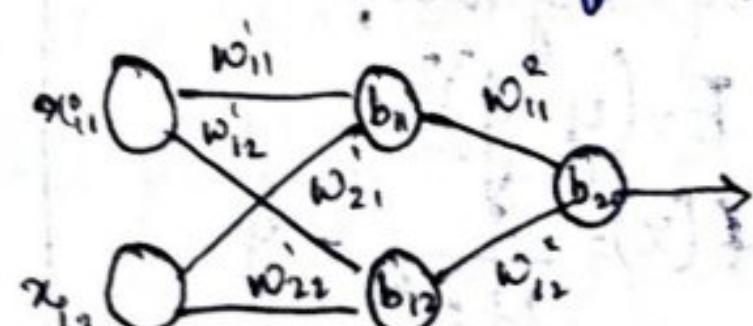
$$a^{[2]} = w^{[1]} a^{[1]} + b^{[2]} = w^{[2]} [w^{[1]} a^{[0]} + b^{[1]}] + b^{[2]}$$

$$= w^{[2]} w^{[1]} a^{[0]} + w^{[2]} b^{[1]} + b^{[2]}$$

$$= w^{[1]} a^{[0]} + b^{[2]}$$

\Rightarrow If activation is not present then we cannot find non-linearity.

Ex:



$$\text{Total learnable parameters} = 6 + 3 = \underline{\underline{9}}$$

⇒ Perceptron Training:

→ Forward propagation $\underline{z = w^T x + b} \rightarrow f(z) = y_{\text{pred}} = \hat{y}$ [loss = $y - \hat{y}$]

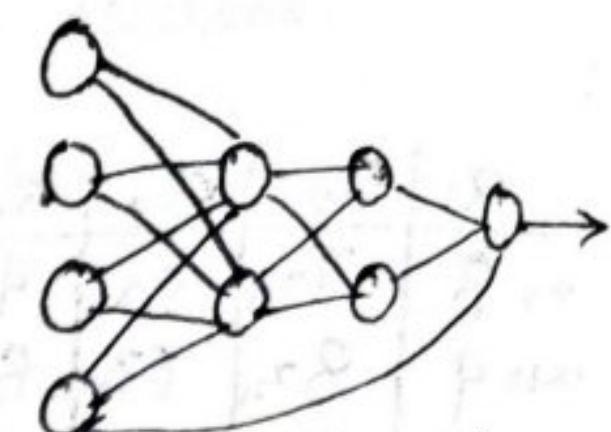
→ Backward propagation Updates weight & bias, when loss is high.

$$w_{\text{new}} = w_{\text{old}} + \eta (t - \hat{y}) x_i, \quad \eta = \text{learning rate}$$

$t = \text{target/actual value}$

$o = \text{output}$

$x_i = \text{input}$



20-09-25

→ Cost Function: Error function/Loss Function

Regression Classification

Cost function in case of regression:

→ Mean absolute error: $\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$

→ Mean square error: $\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$

Cost function in case of classification:

→ Binary Cross-Entropy Loss/Log Loss: $L = -\frac{1}{N} \sum_{i=1}^N y_i \log \hat{y}_i + (1-y_i) \log (1-\hat{y}_i)$

Activation function is Sigmoid.

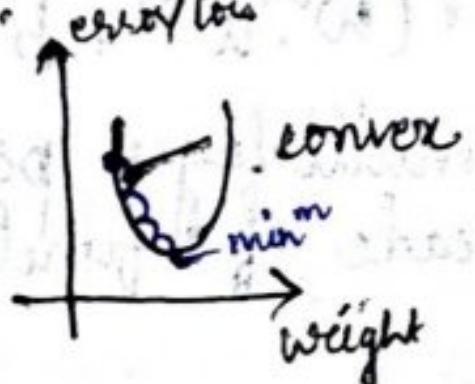
→ Categorical Cross-Entropy Loss: $L = -\frac{1}{N} \sum_{i=1}^N \left(\sum_{j=1}^C y_{ij} \log \hat{y}_{ij} \right), C = \text{no. of classes}$

Activation function is SoftMax.

→ Objective of Cost function:

- Minimize the error

$$E = \frac{1}{n} \sum_{i=1}^n (y - \hat{y})^2$$



convex
min

weight

$-\eta \left(\frac{\partial E}{\partial w} \right)$ → By this the distance is decided.

direction

learning rate, by which steps are decided, $\eta \in [0, 1]$

For sigmoid,

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w}$$

$$= -2(y - \hat{y}) \cdot e^{-z} \cdot \frac{e^{-z}}{(1+e^{-z})^2} \times \alpha_1$$

$$= -2y - \hat{y} \cdot \hat{y}(1-\hat{y}) - \alpha \left[\frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}} \left(1 - \frac{1}{1+e^{-z}} \right) = g(z)g'(z) \right]$$

$$\begin{aligned} E &= (y - \hat{y})^2 \\ \hat{y} &= f(z) \\ z &= w^T x + b \\ \frac{\partial E}{\partial w} &= \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w} \end{aligned}$$

For sigmoid,

$$\begin{aligned} E &= (y - \hat{y})^2 \\ f(z) &= \frac{1}{1+e^{-z}} = \hat{y} \end{aligned}$$

$$z = w_1 x_1 + w_2 x_2 + b$$

Q2.09-25 → Activation Function :

In ANN, each neuron form a weighted sum of its inputs & passes the resulting scalar values through a function referred to as an activation function or transfer function.

Uses:

- If we not apply the activation function in a neural network, it can't capture the non-linear data.
- If activation function is linear, then the neuron performs linear regression/classification.
- In general, activation function is taken to be a non-linear function to do non-linear regression & solve classification problem that are not linearly separable.

→ Ideal activation function:

- Non-linear
- Differentiable
- Computationally inexpensive
- Normalized
- Non-saturating function

→ Sigmoid function:
$$f(x) = \frac{1}{1+e^{-x}}, x \in (-\infty, \infty); 0 < f(x) < 1.$$

Advantage:

- It squeezes the data from $(-\infty, \infty)$ to $(0, 1)$ & we can treat it as a probability function & we can apply it on the output layer when we are dealing with binary classification function.
- It is a non-linear, so we can capture the non-linear pattern in the data.

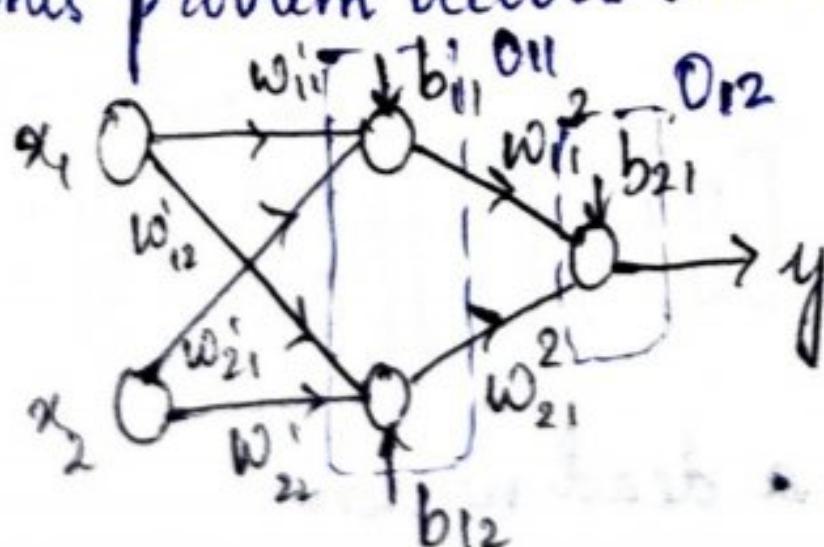
Disadvantage:

- It is a saturating function, i.e., it ranges from $-a$ to a & squeezes & due to saturating properties it faces vanishing gradient problem.
- The sigmoid function is not normalized function or it is a non-zero centered & it is expensive.

→ Vanishing Gradient Problem:

When we perform back propagation & move back, the gradient gradually gets smaller & at a point the weights will not update & the training of the model stops.

This problem occurs in deep neural network where we have multiple hidden layers.



Learnable parameter = 9

$$\begin{aligned} \frac{\partial L}{\partial w_{11}} &= \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z_1} \cdot \frac{\partial z_1}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial z_{11}} \cdot \frac{\partial z_{11}}{\partial w_{11}} \\ &= 0.1 \times 0.1 \times 0.1 \times 0.1 \times 0.1 \\ &= 10^{-5} \end{aligned}$$

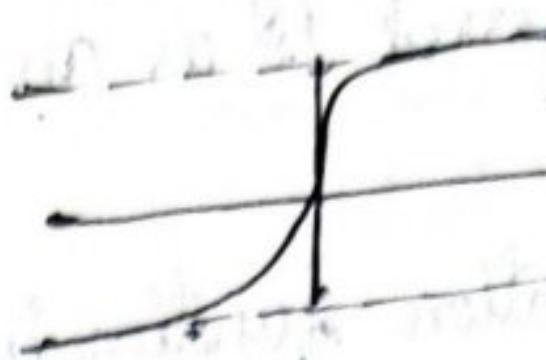
$$\begin{aligned} z_{11} &= x_1 w_{11} + x_2 w_{12} + b_{11} \quad [w_{11} = w_{11}^* - \eta \frac{\partial L}{\partial w_{11}}] \\ z_{21} &= o_{11} w_{21} + o_{12} w_{22} + b_{21} \end{aligned}$$

So, we can't apply Sigmoid function in the hidden layer.

24-09-25

$$\rightarrow \text{Janh: } f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, x \in (-\infty, \infty) \Rightarrow -1 < f(x) < 1$$

squeezes the range $(-\infty, \infty)$ to $(-1, 1)$



Advantage:

- i) It is a non-linear function.
- ii) It is differentiable.
- iii) It is a zero-centered function.

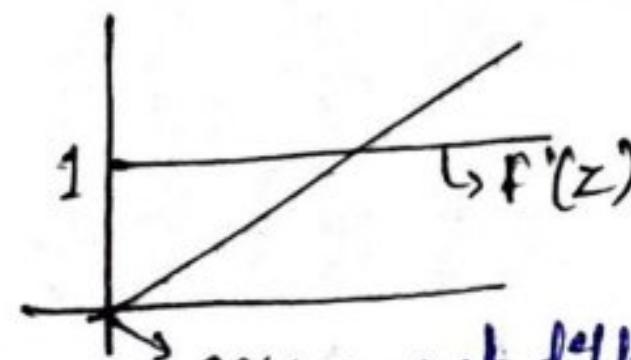
Disadvantage:

- i) It is a saturating function.
- ii) It is computationally expensive.
- iii) It has vanishing gradient problem

\rightarrow RelU: Used in hidden layer

$$f(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ x, & \text{if } x > 0 \end{cases}$$

$$\frac{df}{dx} = \begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{if } x > 0 \end{cases}$$



corner-not differentiable at this point

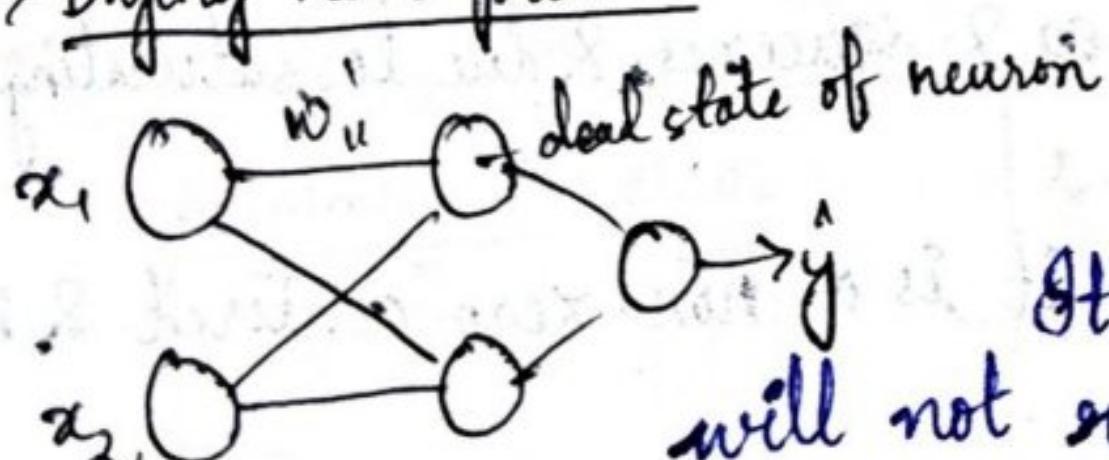
Advantage:

- i) It is non-linear.
- ii) It doesn't face vanishing gradient problem if all, consider the positive region.
- iii) It is computationally in-expensive.
- iv) It converges faster.

Disadvantage

- i) It is not differentiable at $x=0$.
- ii) It is not a zero-centered function.
- iii) It has Dying-RelU problem.

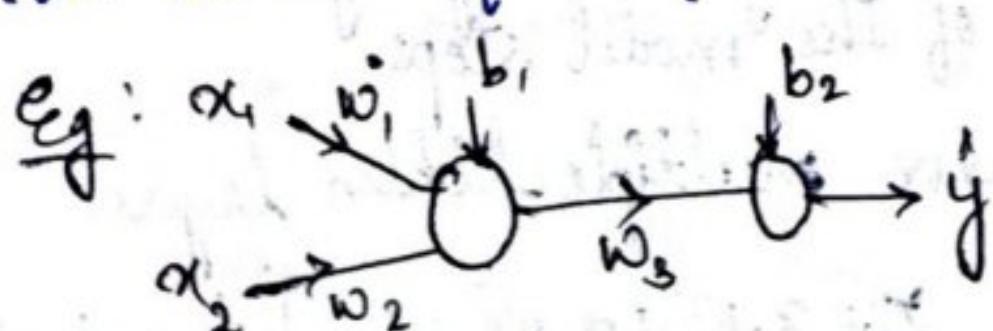
\rightarrow Dying RelU problem:



$$w_{1,\text{new}} = w_{1,\text{old}} + n \frac{\partial L}{\partial w_1}$$

It is a problem where a particular neuron get dead & will not recover from further state by training state.

If 50% of the neuron are dead, then the network will not perform well & not able to find pattern.



$$z_1 = w_1 x_1 + w_2 x_2 + b_1$$

If $x_i < 0$, then $f(z_i) = 0$

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z_i} \cdot \frac{\partial z_i}{\partial w_i} = 0$$

$$w_{1,\text{new}} = w_{1,\text{old}} + n \frac{\partial L}{\partial w_1} = w_{1,\text{old}}$$

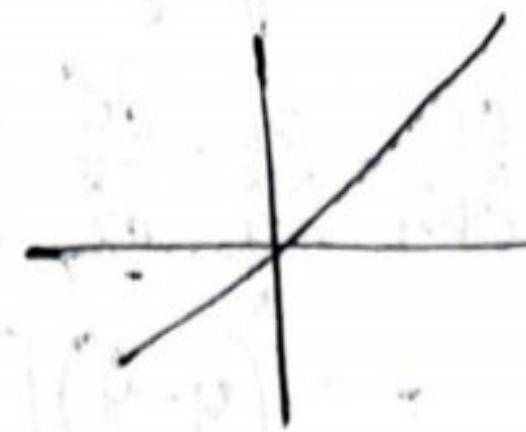
So, no weight updation happens, & this neuron is treated as a dead neuron

Leaky ReLU

$$f(z) = \max\{0.01z, z\}$$

$$= \begin{cases} z, & \text{if } z > 0 \\ 0.01z, & \text{if } z \leq 0 \end{cases}$$

$$f'(z) = \begin{cases} 1, & \text{if } z > 0 \\ 0.01, & \text{if } z \leq 0 \end{cases}$$



Advantage

- i) It prevents dead neuron by allowing a small gradient for negative value.
- ii) Improve gradient flow during back propagation.
- iii) Helps in faster & more stable training compared to ReLU.
- iv) Useful in deep networks where ReLU fails.

Softmax Activation function :

$$f(z) = \frac{e^{a_i}}{\sum_k e^{a_k}}, a_i = \text{value of output layer}$$

X	Class 1	Class 2	Class 3	Class 4
1	0.97	0.3	0.4	0.4 → Class 1
11	0.8	0.4	0.98	0.2 → Class 2
22	0.3	0.97	0.2	0.5 → Class 3
9	0.8	0.4	0.98	0.7 → Class 4

output layer

Used in multiclass classification problem. It will tell us probability of each class. & we can find the class with the highest probability.

$$x_1 \rightarrow 0.5, f(z) = \frac{e^{0.5}}{e^{0.5} + e^{1.5} + e^{0.1}}$$

$$x_2 \rightarrow 1.5$$

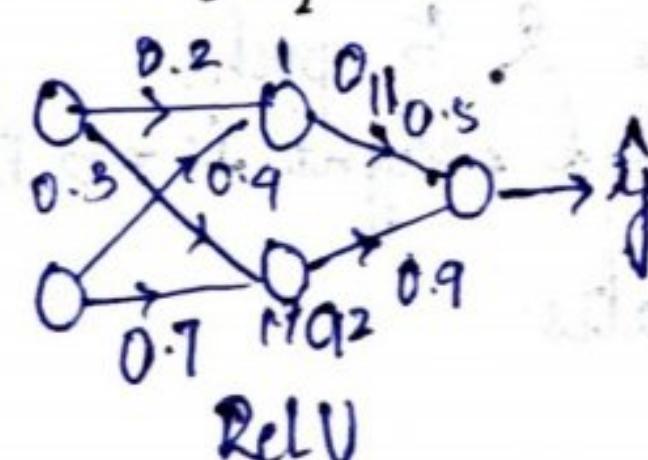
$$x_3 \rightarrow 0.1$$

27-04-25

- Q1) Consider a small network : Input $x = [1 \ 2]^T$, First layer weight, $w_1 = \begin{bmatrix} 0.2 & 0.4 \\ 0.3 & 0.7 \end{bmatrix}$, $b_1 = [0]$. Second layer weight, $w_2 = \begin{bmatrix} 0.5 \\ 0.9 \end{bmatrix}$, $b_2 = 0$. Activation function in 1st layer is ReLU & in the 2nd layer is linear. Loss function is Mean Square error with target $y = 5$, Learning rate, $\eta = 0.01$.
- (1) Perform forward pass (2) Compute loss (3) Find $\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}$ (4) Update w_1, w_2, b_1, b_2

$$z_1 = w_1 x + b_1$$

$$= \begin{bmatrix} 0.2 & 0.4 \\ 0.3 & 0.7 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1.7 \end{bmatrix}$$



$$O_1 = f(z_1) \Rightarrow O_1 = \begin{bmatrix} O_{11} \\ O_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 1.7 \end{bmatrix}$$

$$\hat{y} = 1 \times 0.5 + 1.7 \times 0.7$$

$$z_2 = w_2 \cdot 0 + b_2 = [0.5 \ 0.7] \begin{bmatrix} 1 \\ 1.7 \end{bmatrix} + 0$$

$$= 0.5 + 1.53 = \underline{2.03}$$

$$\hat{y} = \underline{2.03}$$

$$\text{Loss, } L = (y - \hat{y})^2 = (5 - 2.03)^2 = (2.97)^2 = \underline{8.8209}$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2} = \left[\frac{\partial z_2}{\partial w_2} = \cancel{w_2 \cdot 0 + b_2} = 0 \right]$$

$$= -2(y - \hat{y}) \begin{bmatrix} 1 \\ 1.7 \end{bmatrix}^2$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b_2} = -2(y - \hat{y}) \Rightarrow \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1} = \frac{\partial z_1}{\partial w_1} = \frac{\partial}{\partial w_1}$$

$$= -2(y - \hat{y}) \cdot w_2 \cdot 1 \cdot x$$

$$= \begin{bmatrix} 0.5 \\ 0.7 \end{bmatrix}_{2 \times 1} \begin{bmatrix} 1 & 2 \end{bmatrix}_{1 \times 2} = \begin{bmatrix} \dots \\ \dots \end{bmatrix}_{2 \times 2}$$

~~6-10-25~~

→ Types of Gradient Descent :

- i) Stochastic Gradient Descent
- ii) Batch Gradient Descent
- iii) Mini-Batch Gradient Descent

→ Epoch in Neural Network :

When we train a neural network on a training data set, we perform forward & backward propagation. Using gradient descent to update weights. When network has seen every single training input once that is Epoch.

- We can either i/p training examples one by one in mini-batches or entire batches.
- When we have i/p either dataset once, entire dataset has passed to n/w.

→ Batch Optimization/GD :

If we i/p multiple training egs at a time (entire training dataset) to neural n/w & calculate \hat{y} [pred o/p], for each individual sample & collect sample which are not correctly classified.

- When we need to calculate loss function for each individual sample which are not correctly classified & combine them loss funt to get overall loss function.
- In batch optimisation, it uses huge amount of memory & computationally not efficient.

Eg Suppose there are 50 samples & 5 epochs:

No. of times we have updated weight is = 5 times.

* No. of epoch = No. of updates.

Stochastic Optimization:

If we if one training sample at a time & carry out forward & backward propagation to update corresponding weights, then it's called Stochastic Optimization or Stochastic Gradient Descent.

Eg: 50 samples & 5 epochs

No. of times update of weight = 250 times

1 epoch = 50 sample

Total no. of update = $5 \times 50 = 250$.

Mini-Batch optimization / Gradient Descent :

- Here, we should have carried out two passes to compute an epoch.
- Mini batch GD is somewhere between batch & stochastic GD.
- We need ' n ' passes to compute an epoch where n is no. of batches.
- In mini-batch GD, training samples are divided into small sub dataset called mini-batches allowing model to update weights more frequently compared to using entire dataset at once.
- Instead of updating weight after calculating error for each data point (in stochastic GD) or after entire dataset (is batch GD), mini-batch GD updates the weight after processing a mini-batch of data.
- It provides balance between computational efficiency & convergence stability.

Eg: 50 samples & 5 epochs, 10 mini-batch

Total no. of weight update =

1 epoch = 10 update

5 epoch = 50 update

B-10-25

(a) Suppose we are training

x	y
1	2
2	4
3	6

a simple linear regression model $y = wx$ using gradient descent
Loss function is, $L = \frac{1}{2} (y - \hat{y})^2 = \frac{1}{2} (y - wx)^2$, Current weight, $w = 0$ &
learning rate, $\eta = 0.1$.

- (a) Batch Gradient Descent: Compute the gradient using all samples & update w once.
(b) Stochastic Gradient Descent: Compute & update w after each training sample (in given order).

$$(a) \frac{\partial L}{\partial w} = \frac{1}{N} \sum (-y_i - wx_i)x_i$$

$$= \frac{1}{3} (-2-8-18) = -9.33$$

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w} = 0 - 0.1(-9.33) = 0.933$$

$$\frac{\partial L}{\partial w} = -(y_1 - wx_1)x_1, \text{ At } x=1, \frac{\partial L}{\partial w} = -(y_1 - wx_1)x_1 = -(2-0)1 = -2$$

$$\text{At } x=2, \frac{\partial L}{\partial w} = -(y_2 - wx_2)x_2 = -(4-0)2 = -8$$

$$\text{At } x=3, \frac{\partial L}{\partial w} = -(y_3 - wx_3)x_3 = -18$$

$$(b) w_{\text{old}} = 0$$

$$w_{\text{new}} = 0 - 0.1(-2) = 0.2$$

$$\frac{\partial L}{\partial w} \Big|_{x=x_2} = -(y_2 - wx_2)x_2 = -(4 - 0.2 \times 2)2 = -(4 - 0.4)2 = -7.2.$$

$$w = 0.2 - 0.1(-7.2) = 0.92.$$

$$\frac{\partial L}{\partial w} \Big|_{x=x_3} = -(y_3 - wx_3)x_3 = -(6 - 0.92 \times 3)3 = -9.72.$$

$$w = 0.92 - 0.1(-9.72) = 1.892.$$

So, after one full pass or epoch using stochastic GD, the weight $w = 1.892$.

Q) $(x, y) = (1, 2), (2, 4), (3, 6), (4, 8)$; $\hat{y} = wx + b$. Use MSE with factor $\frac{1}{2m}$ per batch, i.e., $L = \frac{1}{2m} \sum_{i=1}^m (y_i - \hat{y}_i)^2$. Initial weight, $w=0$, $b=0$, learning rate, $\eta=0.1$

Mini-batch size = 2. (batch 1: sample 1 & 2)
(batch 2: sample 3 & 4)

(a) Do one epoch: Compute gradients & update w & b after each.

(b) Compute the loss on the full dataset before & after the epoch.

$$(a) \frac{\partial L}{\partial w} = \frac{1}{2m} \cdot 2 \sum_{i=1}^m (y_i - \hat{y}_i)x_i$$

$$= \frac{1}{m} \sum_{i=1}^m (y_i - \hat{y}_i)x_i$$

$$= \frac{1}{2} \sum_{i=1}^2 (y_i - \hat{y}_i)x_i$$

$$= \frac{1}{2} [(2-0)1 + (4-0)2] = \frac{1}{2} [2+8] = -5$$

$$L = \frac{1}{2m} \sum_{i=1}^m (y_i - \hat{y}_i)^2 = \frac{1}{2m} \sum_{i=1}^m (y_i - wx_i - b)^2$$

$$\frac{\partial L}{\partial w} = \frac{1}{2m} \cdot 2(y - wx - b) = -\frac{1}{m} (y - wx - b)x$$

$$\hat{y}_1 = wx_1 + b$$

$$= 0 \times 1 + 0 = 0$$

$$\hat{y}_2 = wx_2 + b = 0$$

$$\frac{\partial L}{\partial b} = -\frac{1}{2} [(y_1 - \hat{y}_1) + (y_2 - \hat{y}_2)] = -\frac{1}{2} [2+0] = -1$$

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{\partial L}{\partial w} = 0 - 0.1(-5) = 0.5$$

$$b_{\text{new}} = b_{\text{old}} - \eta \frac{\partial L}{\partial b} = 0 - 0.1(-1) = 0.3.$$

For batch 2: $\hat{y}_3 = wx_3 + b = 0.5 \times 3 + 0.3 = 1.8$.

$$\hat{y}_4 = wx_4 + b = 0.5 \times 4 + 0.3 = 2.3.$$

$$\frac{\partial L}{\partial w} = -\frac{1}{2} [(y_3 - \hat{y}_3)x_3 + (y_4 - \hat{y}_4)x_4] = -\frac{1}{2} [(6-1.8)3 + (8-2.3)4] = -\frac{1}{2} [12.6 + 22.8] = \frac{1}{2} (35.4) = 17.7$$

$$\frac{\partial L}{\partial b} = -\frac{1}{2} [(y_3 - \hat{y}_3) + (y_4 - \hat{y}_4)] = -\frac{1}{2} [(6-1.8) + (8-2.3)] = -\frac{1}{2} [4.2 + 5.7] = -\frac{9.9}{2} = -4.95$$

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{\partial L}{\partial w} = 0.5 - 0.1 \times (-17.7) = 2.27.$$

$$b_{\text{new}} = b_{\text{old}} - \eta \frac{\partial L}{\partial b} = 0.3 - 0.1 \times (-4.95) = 0.795.$$

$$(b) \text{ Initially, } L = \frac{1}{2 \times 4} \sum_{i=1}^4 (y_i - \hat{y}_i)^2 = \frac{1}{8} [2^2 + 4^2 + 6^2 + 8^2] = \frac{1}{8} [4 + 16 + 36 + 64] = \frac{120}{8} = 15.$$

$$\text{After epoch, } L = \frac{1}{8} \sum_{i=1}^4 (y_i - \hat{y}_i)^2$$

$$= \frac{1}{8} [(2-3.065)^2 + (4-5.335)^2 + (6-7.605)^2 + (8-9.875)^2]$$

$$= \frac{1}{8} [(1.065)^2 + (1.335)^2 + (1.605)^2 + (1.875)^2]$$

$$\approx \frac{1}{8} [9.0081] \approx 1.126$$

$$\begin{cases} \hat{y}_1 = w_1 x_1 + b = 2.27 \times 1 + 0.795 = 3.065 \\ \hat{y}_2 = 2.27 \times 2 + 0.795 = 5.335 \\ \hat{y}_3 = 2.27 \times 3 + 0.795 = 7.605 \\ \hat{y}_4 = 2.27 \times 4 + 0.795 = 9.875 \end{cases}$$

Q 10-25

Overfitting:

It happens when a model learns the noise & random fluctuations in the training data instead of the true underlying patterns.

Symptoms of overfitting:

- (i) Training accuracy is very high.
- (ii) Testing accuracy or validation accuracy is lower.
- (iii) Loss curve: training loss keeps decreasing, but validation loss increases after some epochs.

Causes of overfitting:

- (i) Model is complex, i.e. too many parameters, no. of hidden layer is too large.
- (ii) Insufficient training data.
- (iii) Training for too many epochs.
- (iv) Poorly tuned learning rate or hyperparameters.
- (v) Lack of regularization (no penalty for large weights).

Techniques to prevent overfitting:

- i) Regularization: Add a penalty to the loss function to discourage overly large weights.
 - L2 regularization: $L' = L + \lambda \sum_i w_i^2$ where λ is regularization hyper parameter $[L = (y_i - \hat{y}_i)^2]$
 - L1 regularization: Force some weights to become zero.
- ii) Dropout: Randomly "turn off" a fraction of neurons during training (e.g. 20% - 50%). This prevents the network from co-adapting to much on specific neurons.
 - If dropout rate is 0.5, then half of the neurons are randomly ignored each forward pass. At test time, all neurons are used but scaled by 0.5.
- iii) Data Augmentation: Artificially increase dataset size using transformations like
 - (a) Rotation, flipping, cropping (for image).
 - (b) Noise injection (for text/audio).
 - (c) Scaling, translation.
- iv) Early Stopping: Monitor validation loss - stop training when it starts increasing even though the training loss still decreasing. It prevent memorization of the training data.
- v) Batch Normalization: Normalize the activations across a mini-batch & adding slight regularization. It helps models converge faster & resist overfitting.

- vi) Cross-Validation: Instead of single validation set, use k -fold cross validation:
- Split data into k -subsets.
 - Train k -times, each time using a different subset as validation.
 - Average the result.

11-10-25

→ Matrix for evaluation of classifier:

1. Confusion Matrix:

Actual output			
		1	0
Predicted output	1	TP	FP
	0	FN	TN

True Positive (TP): Positive sample classified as positive.

True Negative (TN): Negative sample classified as negative.

False Positive (FP): Negative sample classified as positive.

False Negative (FN): Positive sample classified as negative.

x_1	x_2	\hat{y}	y
0.5	2.3	0	1
2.5	4.3	1	1
0.2	3.2	0	1
0.5	1.5	1	0
3.2	0.3	0	0

Confusion Matrix:

1	1	0
1	1	1
0	2	1

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \rightarrow \text{Total sample classified as positive}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Total no. of sample which was actually positive

$$\text{F1 score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}$$

Actual		Actual	
		spam	not spam
Pred.	spam	40	10
Pred.	not spam	5	45

$$\text{Recall} = \frac{10}{10+5} = \frac{10}{15} = 0.667$$

Find Accuracy & F1 score:

$$\text{Accuracy} = \frac{40+45}{40+45+10+5} = \frac{85}{100} = 0.85. \text{ Precision} = \frac{40}{40+10} = \frac{40}{50} = 0.8$$

$$\text{F1 score} = \frac{2 \times 0.8 \times 0.83}{0.8 + 0.83} = \frac{1.33}{1.65} = 0.80$$

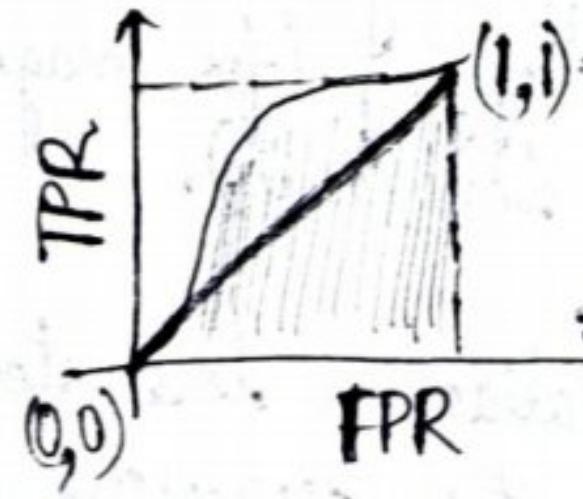
2. ROC Curve: ROC curve is a graphical plot that shows how a classifier's performance changes as the decision threshold varies.

$$x\text{-axis: FPR (False Positive Rate)} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

$$y\text{-axis: TPR (True Positive Rate)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Perfect Classifier, AUC=1

Random Classifier, AUC=1/2



AUC: Area under curve

ROC: Random Classifier

Q5. 10 Marks

Write a python code to train a perceptron for AND gate.

Activation function : Step function $f(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{otherwise.} \end{cases}$

Initial weight : $w_1 = 0, w_2 = 0$

Initial bias : $b = 0.1$

Learning rate : $\eta = 0.1$

No. of epoch : 10

Display the final weights & bias & predicted value.

$$X = \{(0,0), (0,1), (1,0), (1,1)\}$$

$$Y = \{0, 0, 0, 1\}$$

Step 1: Define activation function

Step 2: Define AND gate truth table

Step 3: Initialize weights & bias

Step 4: Train using perceptron rule

for epoch in range(10):

 for i in range(len(X)):

$$x = X[i]$$

$$y = Y[i]$$

$$z = w \cdot x + b$$

$$y_{pred} = f(z)$$

update w & b

Step 5: Display the output

25-10-25

Chapter 6 Beyond Gradient Descent

→ Challenges in Gradient Descent :

- 1) Local minima in Deep Network: Local minima are points where the gradients are zero but not necessarily they are global.
- 2) Model Identifiability: Multiple parameter configurations can produce same network output Eg: In a layer with n neurons, there are $n!$ ways to permute neurons without changing the final outputs. If the network has L layers, then there will be $(n!)^L$ many such configurations.
- 3) Flat regions in the error surface: Flat region (plateaus) occurs when gradients are near zero. In this region optimizer moves very slowly, so the training process stagnates here. It is common in deep networks due to ReLU saturation or poor weight initialization.
- 4) When gradient points in the wrong direction: In high-dimensional parameter spaces, the gradients may not always point directly toward the minimum. This is called ill-conditioning of the loss function. This ill-conditioning of the loss function leads to zigzagging behaviour, so the training process slows down.

→ Standard Gradient Descent: $\theta_{t+1} = \theta_t - \eta \frac{\partial L}{\partial \theta_t}$

→ Momentum based Gradient Descent: $v_t = \beta v_{t-1} - \eta \frac{\partial L}{\partial \theta_t}$

$$\theta_{t+1} = \theta_t + v_t = \theta_t - \eta \frac{\partial L}{\partial \theta_t} + \beta v_t$$

where,

β = momentum coefficient
(commonly $\beta = 0.9$)

v_t = velocity or running avg of gradients

η = learning rate

27-10-25

→ AdaGrad (Adaptive Gradient):

- Motivation:

- Some parameters learn faster than others.
- Some features appear more frequently than others.

- Problem:

One learning rate for all parameters is insufficient.

- Solution:

AdaGrad adapts the learning rate per parameter automatically.

- Idea:

AdaGrad adjusts the learning rate for each parameter based on its previous gradients:

- Parameters that receive large gradients get smaller learning rates (slow down).
- Parameters with small gradients get larger learning rates (speed up).

Formula:

$$r_i = r_{i-1} + g_i^2$$
$$\theta_{i+1} = \theta_i - \frac{\eta}{\sqrt{r_i + \epsilon}} g_i$$

g_i^2 = gradient in i^{th} iteration

$$r_i = r_{i-1} + g_i^2 = r_{i-2} + g_{i-1}^2 + g_i^2$$
$$= r_{i-2} + g_{i-2}^2 + g_{i-1}^2 + g_i^2$$
$$= g_1^2 + g_2^2 + \dots + g_i^2$$

$$\epsilon = 10^{-8}$$

= a small constant, to avoid division by zero.

= Sum of the squares of previous gradients

Limitations:

- i) Learning rate decays too fast.
- ii) Poor performance for non-convex problems.
- iii) Not suitable for long training runs.

RMS Prop (Root Mean Square Prop):

It uses exponentially weighted moving average instead of sum of the square of the past gradient, which avoids the vanishing learning rate.

$$r_i = \beta r_{i-1} + (1-\beta) g_i^2$$

$$\theta_{i+1} = \theta_i - \frac{\eta}{\sqrt{r_i + \epsilon}} g_i$$

β = decay factor

If $\beta=0 \Rightarrow r_i = g_i^2$

If $\beta=1 \Rightarrow r_i = r_{i-1}$

$$r_i = \beta r_{i-1} + (1-\beta) g_i^2$$

$$= \beta(r_{i-2} + (1-\beta)g_{i-1}^2) + (1-\beta)g_i^2$$

$$= \beta^2 r_{i-2} + \beta(1-\beta)g_{i-1}^2 + (1-\beta)g_i^2$$

$$= \beta^3 r_{i-3} + \beta^2(1-\beta)g_{i-2}^2 + \beta(1-\beta)g_{i-1}^2 + (1-\beta)g_i^2$$

$$= \dots$$

$$= \beta^i r_0 + \beta^{i-1}(1-\beta)g_1^2 + \dots$$

$$r_i = \beta(1-\beta) \sum_{k=1}^i \beta^{i-k} g_k^2 \quad (\because r_0 = 0)$$

→ Adam (Adaptive Moment Estimation):

It combines the advantages of Momentum & RMS Prop.

- Momentum keeps moving average of the past gradient (direction).
- RMS Prop keeps moving average of the squared gradient (magnitude).

Formula: First moment (mean of gradients) → capture direction

$$m_t = \beta m_{t-1} + (1-\beta_1) g_t$$

β_1 = decay rate for first moment.

Second moment (uncentered variance) → capture scale

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

β_2 = decay rate for second moment

Bias correction (to fix the initialization bias at start)

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

Parameter update:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \cdot \hat{m}_t$$

where generally $\beta_1 = 0.1$

$\beta_2 = 0.999$

$\eta = 0.01$

$\epsilon = 10^{-8}$

30-10-25

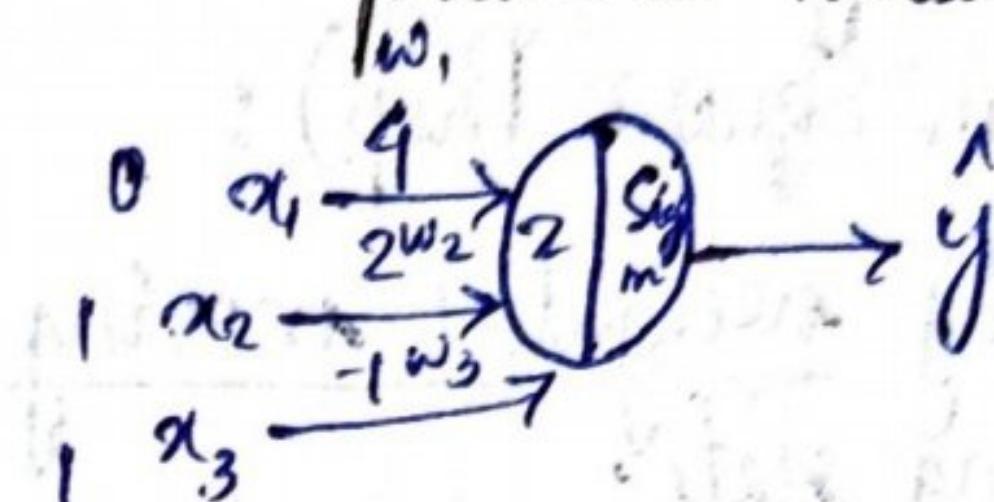
1. Q) Consider a perceptron with 3 inputs $x_1=0, x_2=1, x_3=1$, the corresponding weights are $w_1=4, w_2=2, w_3=-1$ & the activation function is sigmoid. Find the predicted value \hat{y} .

A perceptron = 1 neuron \Rightarrow Single-layer Perceptron

$$\begin{aligned} z &= \sum w_i x_i \\ &= (4 \times 0) + (2 \times 1) + (-1 \times 1) \\ &= 0 + 2 - 1 = 1. \end{aligned}$$

$$\hat{y} = f(z)$$

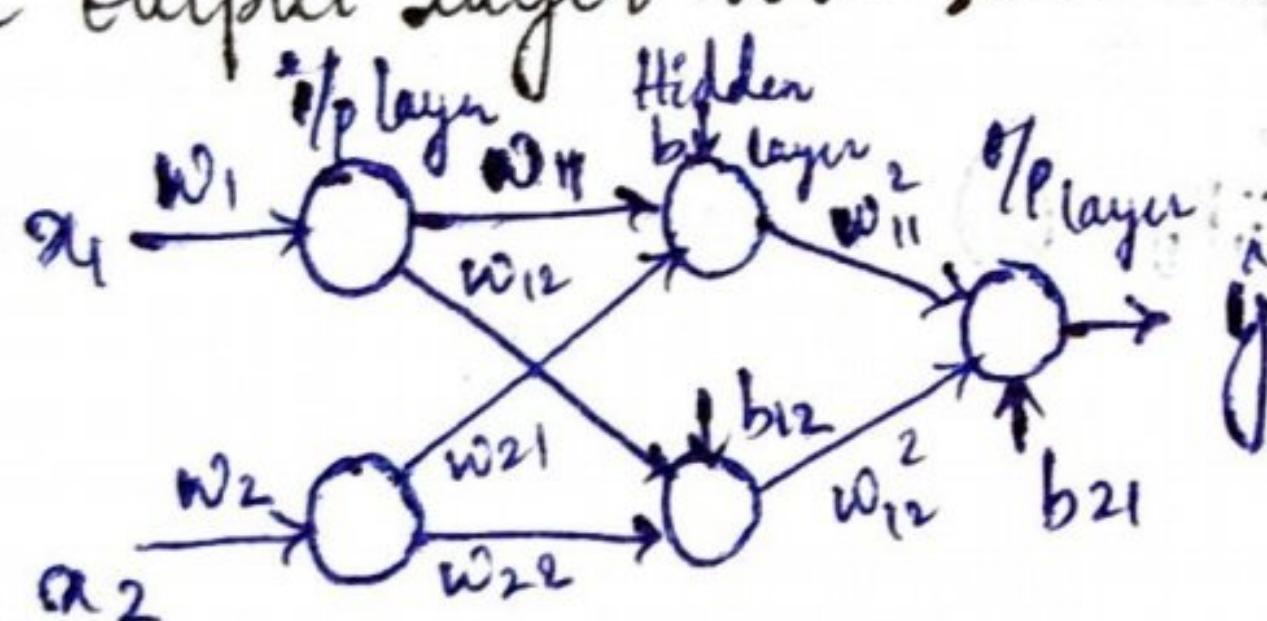
$$f(z) = \frac{1}{1+e^{-z}}$$



$$f(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+0.3678} = \underline{\underline{0.7310}}$$

$$\hat{y} = f(z) = 0.7310$$

2. Q) Consider a neural network with one input layer, one hidden layer with 2 neurons & one output layer with one neuron. Find total no. of learnable parameters.



$$\begin{aligned} \text{Total no. of learnable parameters} &= n \times m + m \\ &= 6 + 3 = 9 \quad [m = \text{no. of neurons}] \\ &\quad \begin{matrix} \checkmark \\ \text{weight} \end{matrix} \quad \begin{matrix} \checkmark \\ \text{bias} \end{matrix} \quad [n = \text{no. of nodes}] \end{aligned}$$

~~$$\text{cosine sim}(A, B) = \frac{0.2 \times 0.3 + 0.5 \times 0.4 + 0.8 \times 0.7 + 0.1 \times 0.2}{\sqrt{0.2^2 + 0.5^2 + 0.8^2 + 0.1^2} * \sqrt{0.3^2 + 0.4^2 + 0.7^2 + 0.2^2}} = \frac{0.84}{0.96 * 0.8839} = 0.98 (98\%)$$~~
~~$$\text{cosine sim}(A, C) = \frac{0.42}{0.9695} = 0.45 (45\%)$$~~
~~$$\text{cosine sim}(B, C) = \frac{0.51}{0.88 * 0.97} = 0.59 (59\%)$$~~

9-10-25

① Given: Initial weight, $w = 0$

learning rate, $\eta = 0.1$

Momentum factor, $\beta = 0.9$

Gradient over three steps: $[0.5, 0.4, -0.2]$

Compute updated weights for each step using momentum based gradient descent.

A: The velocity update is given by: $v_t = \beta v_{t-1} + \eta g_t$ [t=step]

The weight update is given by: $w_{t+1} = w_t - v_t$

For 1st step (t=1): $w_0 = 0, \eta = 0.1, \beta = 0.9, g_1 = 0.5$

$$v_1 = \beta v_0 + \eta g_1 = 0.9(0) + (0.1)(0.5) = 0.05$$

$$w_1 = w_0 - v_1 = 0 - 0.05 = -0.05$$

For 2nd step (t=2): $w_1 = -0.05, g_2 = 0.4$

$$v_2 = \beta v_1 + \eta g_2 = (0.9)(0.05) + (0.1)(0.4) = 0.045 + 0.04 = 0.085$$

$$w_2 = w_1 - v_2 = -0.05 - 0.085 = 0.135$$

For 3rd step (t=3): $w_2 = -0.135, g_3 = -0.2$

$$v_3 = \beta(v_2) + \eta g_3 = (0.9)(0.085) + (0.1)(-0.2) = 0.0765 - 0.02 = 0.0565$$

$$w_3 = w_2 - v_3 = -0.135 - 0.0565 = -0.1915$$

So, updated weights: $w_1 = -0.05$

$$w_2 = -0.135$$

$$w_3 = -0.1915$$

② Given: learning rate, $\eta = 0.1$

Gradients over iterations: $[0.2, 0.4, 0.1]$

Compute effective learning rate & updated parameter values for each steps:

Effective learning rate decreases per step $\Rightarrow \eta_t = \eta / \sqrt{t}$

For 1st step (t=1): $\eta_1 = 0.1 / \sqrt{1} = 0.1$

$$\Delta w_1 = 0.1(0.2) = 0.02$$

For 2nd step (t=2): $\eta_2 = 0.1 / \sqrt{2} = 0.0707$

$$\Delta w_2 = 0.0707(0.4) = 0.0283$$

For 3rd step (t=3): $\eta_3 = 0.1 / \sqrt{3} = 0.0577$

$$\Delta w_3 = 0.0577(0.1) = 0.00577$$

So, updated weights: $w_1 = -0.02$

$$w_2 = -0.0483$$

$$w_3 = -0.0591$$

③ Given: $\eta = 0.01$

$$\beta_1 = 0.9$$

$$\beta_2 = 0.999$$

$$\epsilon = 10^{-8}$$

Gradient over iterations = [0.3, 0.2, 0.1]

Initial $m_0 = 0, v_0 = 0$

Calculate $m_t, v_t, \hat{m}_t, \hat{v}_t$ for the first three iterations & update θ_t .

A: Formulas: $m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2$$

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

$$\Delta w_t = \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Iteration 1: $m_1 = 0.9(0) + 0.1(0.3) = 0.03$

$$v_1 = 0.999(0) + 0.001(0.3)^2 = 0.00009$$

$$\hat{m}_1 = 0.03 / (1 - 0.9) = 0.3$$

$$\hat{v}_1 = 0.00009 / (1 - 0.999) = 0.09$$

$$\Delta w_1 = 0.01 \times \frac{0.3}{\sqrt{0.09} + 10^{-8}} = 0.01$$

Iteration 2: $m_2 = 0.9(0.03) + 0.1(0.2) = 0.047$

$$v_2 = 0.999(0.00009) + 0.001(0.2)^2 = 0.00008991 + 0.00004 = 0.0001299$$

$$\hat{m}_2 = 0.047 / (1 - 0.9^2) = 0.247$$

$$\hat{v}_2 = 0.0001299 / (1 - 0.999^2) = 0.065$$

$$\Delta w_2 = 0.01 \times \frac{0.247}{\sqrt{0.065} + 10^{-8}} \approx 0.0097$$

Iteration 3: $m_3 = 0.9(0.047) + 0.1(0.1) = 0.0423 + 0.01 = 0.0523$

$$v_3 = 0.999(0.0001299) + 0.001(0.1)^2 = 0.0001298 + 0.00001 = 0.0001398$$

$$\hat{m}_3 = 0.0523 / (1 - 0.9^3) = 0.191$$

$$\hat{v}_3 = 0.0001398 / (1 - 0.999^3) = 0.0467$$

$$\Delta w_3 = 0.01 \times \frac{0.191}{\sqrt{0.0467} + 10^{-8}} = 0.0088$$

Iteration

$$1 \quad m_t$$

$$2 \quad v_t$$

$$3 \quad \Delta w_t$$

1

$$0.03$$

$$0.00009$$

$$0.0100$$

2

$$0.047$$

$$0.00013$$

$$0.0097$$

3

$$0.0523$$

$$0.00014$$

$$0.0088$$

Hessian Matrix:

$$L = L(w_1, w_2, \dots, w_n)$$

$$\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \dots, \frac{\partial L}{\partial w_n}$$

Hessian Matrix (H) =

$$\begin{bmatrix} \frac{\partial^2 L}{\partial w_1^2} & \frac{\partial^2 L}{\partial w_1 \partial w_2} & \dots & \frac{\partial^2 L}{\partial w_1 \partial w_n} \\ \frac{\partial^2 L}{\partial w_2 \partial w_1} & \frac{\partial^2 L}{\partial w_2^2} & \dots & \frac{\partial^2 L}{\partial w_2 \partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 L}{\partial w_n \partial w_1} & \frac{\partial^2 L}{\partial w_n \partial w_2} & \dots & \frac{\partial^2 L}{\partial w_n^2} \end{bmatrix}$$

$$\boxed{\frac{\partial^2 L}{\partial w_i \partial w_j} = \frac{\partial}{\partial w_i} \left(\frac{\partial L}{\partial w_j} \right)}$$

- The gradient vector (∇L) tells you the direction of steepest ascent.
- The Hessian matrix tells you how curvature changes, i.e., how fast or how slow the slope changes in each direction.

$$\nabla L = \frac{\partial L}{\partial w} = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \\ \frac{\partial L}{\partial w_n} \end{bmatrix}$$

Eg: $L = 100w_1^2 + w_2^2$

$$H = \begin{bmatrix} \frac{\partial^2 L}{\partial w_1^2} & \frac{\partial^2 L}{\partial w_1 \partial w_2} \\ \frac{\partial^2 L}{\partial w_2 \partial w_1} & \frac{\partial^2 L}{\partial w_2^2} \end{bmatrix}$$

$$= \begin{bmatrix} 200 & 0 \\ 0 & 2 \end{bmatrix}$$

$$\frac{\partial L}{\partial w_1} = 200w_1, \quad \frac{\partial^2 L}{\partial w_1^2} = 200$$

$$\frac{\partial^2 L}{\partial w_2 \partial w_1} = \frac{\partial}{\partial w_2} \left(\frac{\partial L}{\partial w_1} \right) = \frac{\partial}{\partial w_2} (200w_1) = 0$$

$$\frac{\partial^2 L}{\partial w_2^2} = 2, \quad \frac{\partial^2 L}{\partial w_1 \partial w_2} = 0$$

- The eigen values of H are 200 and 2.

Note: If A is a diagonal matrix, then the eigen value of A are the diagonal elements of A .

• Large eigen value (200): very steep direction.

• Small eigen value (2): shallow direction.

Thus, the loss surface is ill-conditioned.

• Conditioned number, $K(H) = \frac{\lambda_{\max}}{\lambda_{\min}} = \frac{200}{2} = 100$

• Conditioned number high means:

- optimization will be slow

- Gradients oscillate between steep & flat direction.

- The system is ill-conditioned.

01-11-25

Given dataset :

x	1	2	3
y	3	5	7

, Model: $\hat{y} = w_0x + b$, Initial: $w=0, b=0, \eta=0.1$

Perform two full epoch (batch gradient descent) & update w & b & do same for Stochastic Gradient descent.

$$L = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \frac{1}{n} \sum (y_i - (w_0 x_i + b))^2$$

$$\hat{y}_1 = w_0 x_1 + b$$

$$\hat{y}_2 = w_0 x_2 + b$$

$$\hat{y}_3 = w_0 x_3 + b$$

$$\frac{\partial L}{\partial w} = -\frac{2}{n} \sum (y_i - (w_0 x_i + b)) x_i$$

$$\frac{\partial L}{\partial b} = -\frac{2}{n} \sum (y_i - (w_0 x_i + b))$$

$$w = w - \eta \frac{\partial L}{\partial w}$$

$$b = b - \eta \frac{\partial L}{\partial b}$$

27-11-25

Convolutional Neural Network (CNN)

CNN has mainly three types of layer:

i) Convolutional layer:

- Uses Kernel/filters to scan the images.
- Detects edges, corners, textures, etc.
- Output a feature map showing where patterns appear.

ii) Pooling layer:

- Reduces the size of the feature maps.
- Makes the network faster, more stable & less sensitive to noise.

iii) Fully Connected layer:

- Works like a normal neural network.
- Take the extracted features & performs classification.

→ Why CNN works so well? :

CNN uses three key ideas:

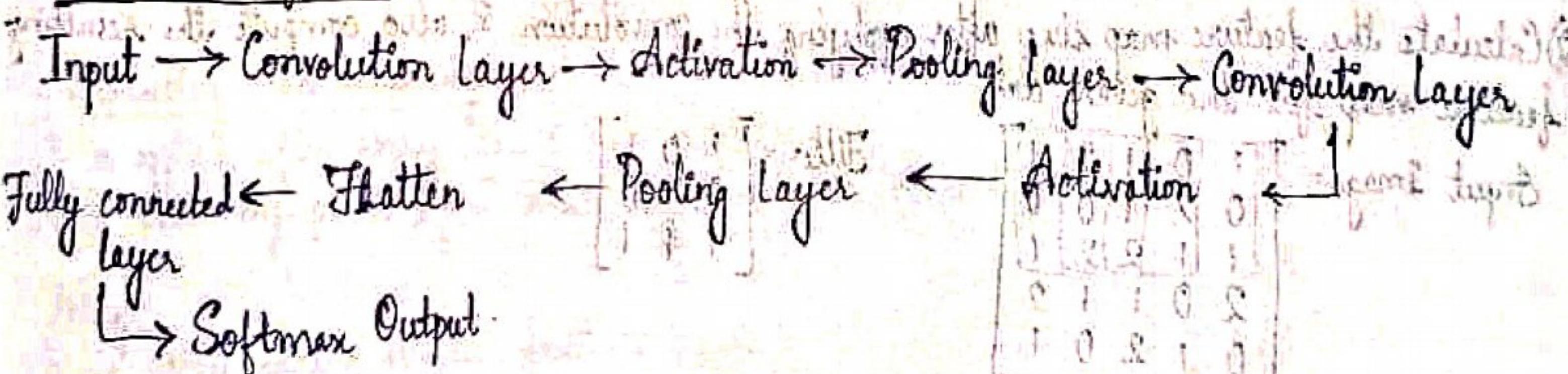
- Local receptive fields: looks at only small parts of the image (like using a magnifying glass).
- Weight Sharing: A single filter detects the same pattern everywhere in the image.
- Hierarchical Features: lower layers detect edges → Higher layer detect objects.

Together, these make CNNs extremely powerful for tasks like:

1. Image detection
2. Object detection
3. Face recognition, etc.

01-12-25

→ Architecture of CNN :



→ Dim/Size of convolutional : $\left\lfloor \frac{n-f+1}{s} \right\rfloor$

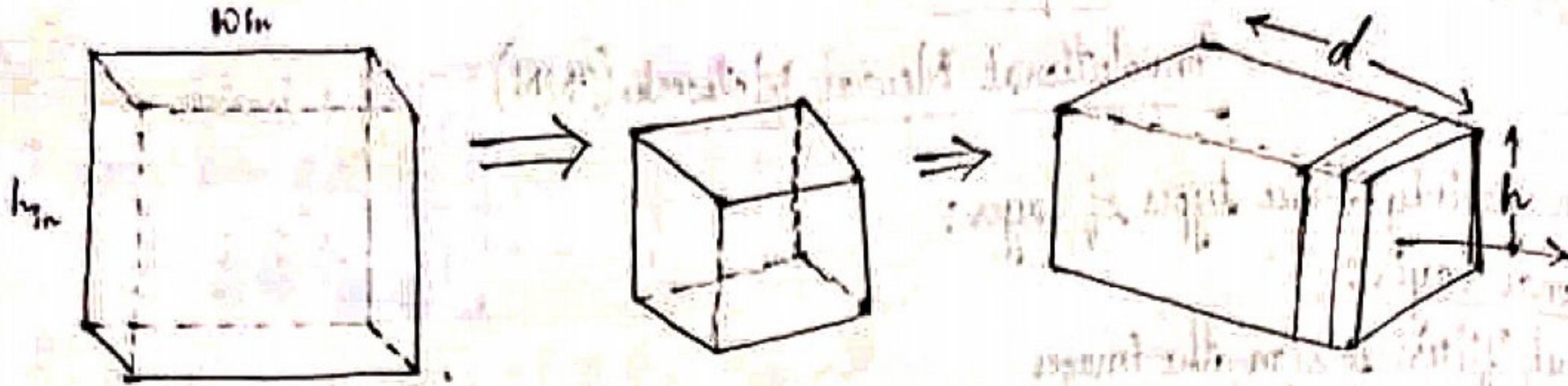
n: Size of input

f: Size of filter

s: no. of stride

p: padding size

If padding, then: $\left\lfloor \frac{n-f+2p}{s} \right\rfloor + 1$



$k = \text{no. of features}$

$$\Rightarrow W_{\text{out}} = \left\lceil \frac{W_{\text{in}} - f + 2p}{s} \right\rceil + 1$$

$$\Rightarrow h_{\text{out}} = \left\lceil \frac{h_{\text{in}} - f + 2p}{s} \right\rceil + 1$$

03-12-25

Eg: Image size = $32 \times 32 \times 3$ ^{height width depth}

Fully connected layer has 10 neurons

Feature size = 3×3

Total no. of learnable parameters = ?

No. of filters = 16

A: Number of learnable parameters before fully connected layer = $3 \times 3 \times 3$

For 16 filters = $3 \times 3 \times 3 \times 16$.

With no. of bias = $(3 \times 3 \times 3) \times (16) + 16 = 448$

Formula = (filter height) \times (filter width) \times (#input channels) \times (no. of filters) + bias of each filter

Number of parameters in fully connected layer = (#input) \times (#neurons) + bias of each neuron in pby

$$= (32 \times 32 \times 16) \times 10 + 10 = 163850$$

Total no. of learnable parameters = $448 + 163850 = 164298$.

Q) Calculate the feature map size after applying the convolution & also compute the resulting feature map for the given data.

Input image =

1	0	1	2	1
0	2	1	0	0
1	1	2	2	1
2	0	1	1	2
0	1	2	0	1

$$\text{Filter} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

A: Feature map size = $\left\lceil \frac{n-f+1}{s} \right\rceil = \left\lceil \frac{5-3+1}{1} \right\rceil = 3$

Feature map = $\begin{bmatrix} 8 & 8 & 7 \\ 5 & 6 & 7 \\ 6 & 7 & 7 \end{bmatrix}$, padding = 1

$$1 + \left\lceil \frac{(n+2p-f) \times (m+2p-f)}{s^2} \right\rceil$$

$$\text{Q) Input} = \begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix} \quad \text{Filter} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

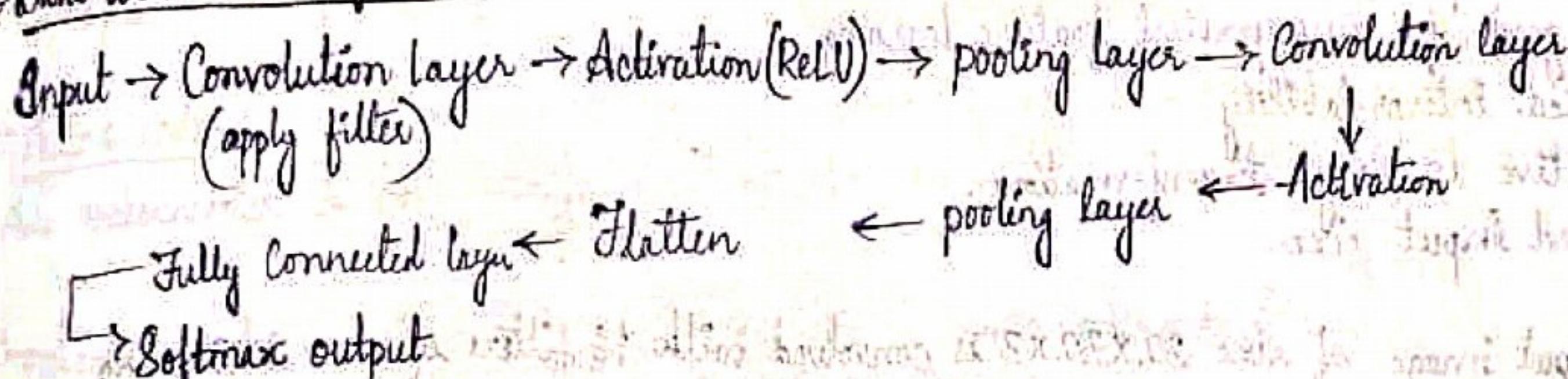
$$1: \text{After padding} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{1 \times 4}$$

Size of feature map = $\left\lfloor \frac{n-f+2p}{s} \right\rfloor + 1$
 $= \left\lfloor \frac{2-3+(2 \times 1)}{1} \right\rfloor + 1$
 $= 2$

$$\text{Feature map} = \begin{bmatrix} 3 & 3 \\ 3 & 2 \end{bmatrix}$$

06.12.25

⇒ Basic architecture of CNN:



⇒ CNNs for Classification:

- i) Convolution
- ii) Non-linearity
- iii) Pooling

⇒ CNN: Spatial Arrangement of Output Volume:

Layer Dimension - $h \times w \times d$; h, w - spatial dimension; d - no. of filters

Stride - Filter step size

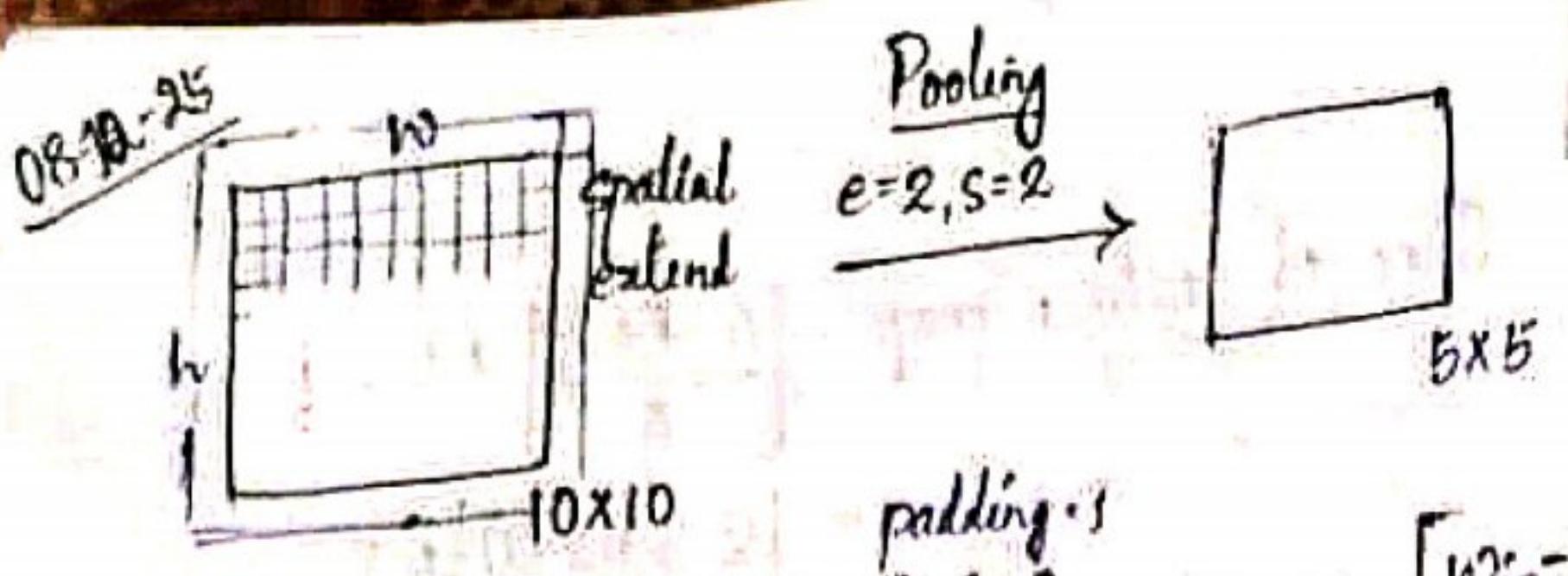
Receptive Field - Location in input that a node path connected to.

⇒ Introducing Non-Linearity:

- It is applied after every convolution operation
- ReLU: pixel-by-pixel oper" that replaces all -ve values by zero

⇒ Pooling:

- It is downgrading of the feature map.
- Max Pooling: By considering the ^(highest) maximum element.
- Avg Pooling: By considering the average



padding = 1

s=3, S=2

$$W_{\text{out}} = \left\lceil \frac{W_{\text{in}} - e}{s} \right\rceil + 1 = \left\lceil \frac{10 - 3}{2} \right\rceil + 1 = 4 + 1 = 5$$

$$h_{\text{out}} = \left\lceil \frac{h_{\text{in}} - e}{s} \right\rceil + 1$$

→ Limitation of CNN :

- i) Requires large labeled dataset.
- ii) High computational cost.
- iii) Not good for unsupervised feature learning.
- iv) Limited interpretability.
- v) Sensitive to large transformations.
- vi) Fixed input size.

Q) An input image of size $32 \times 32 \times 3$ is convolved with 16 filters of size 3×3 using stride=1, padding=0. Find the size of output feature map & total no. of learnable parameters.

Size of feature map : $\frac{32 - 3 + 0}{1} + 1 = 30$

$$\Rightarrow 30 \times 30 \times 16$$

Total no. of learnable parameters = $(3 \times 3 \times 3 + 1) 16$