Deep Learning Using Python

Samir Kundu
samirkundu@soa.ac.in
9475094548

Book: Fundamentals of Deep Learning
By Nitin Buduma
Nikhil Buduma
Joe Papa

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-1
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 Lelum, L Bottou, Y Bengio, P Haffner (1998)]

Column Matrix: $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_{n\times 1}$ $\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & & & \\ \vdots & & & \\ x_{m1} & & & \end{bmatrix}_{n\times m}$ when x_{nm} is 1 it will glow otherwise not

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: $\boxed{Ax = 0}$

$$\begin{bmatrix} 4 & 6 \\ 2 & 7 \\ 5 & 8 \end{bmatrix}_{3\times 2} * \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_{2\times 1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}_{3\times 1}$$

$\Rightarrow 4x_1 + 6x_2 = 0$ -(i)
$2x_1 + 7x_2 = 0$ -(ii)
$5x_1 + 8x_2 = 0$ -(iii)

By solving the eqn", Null Space $= \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

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

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

$x_1 = 2x_2$

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

Eigen Values: $\boxed{A - \lambda I_n = 0}$ $\quad \det(A - \lambda I) = 0$

Eigen Vectors: $\boxed{(A - \lambda I_n)x = 0}$

03-09-25

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

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

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

$\Rightarrow \det(A - \lambda I) = 0$
$\Rightarrow (1-\lambda)(4-\lambda) - 10 = 0$
$\Rightarrow \lambda^2 - 5\lambda - 6 = 0$
$\Rightarrow \lambda^2 + \lambda - 6\lambda - 6 = 0$

$\lambda(\lambda+1) - 6(\lambda+1) = 0$
$(\lambda+1)(\lambda-6) = 0$
$\lambda = 6, -1$ Eigen Value

eigen value ↑ eigen vector ↑

Eigen vector corresponding to $\lambda = 6 \Rightarrow (A - \lambda I)x = 0$

Putting $\lambda = 6$ in $A - \lambda I \Rightarrow \begin{bmatrix} -5 & 2 \\ 5 & -2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$\Rightarrow -5x_1 + 2x_2 = 0$
$5x_1 - 2x_2 = 0$
$x_2 = \frac{5}{2}x_1$

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

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

where c is non-zero real number

Eigen vector corresponding to $\lambda = -1$

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

$5x_1 + 5x_2 = 0$

$x_2 = -x_1$

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

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

$\Rightarrow \det(A - \lambda I) = 0$

$\Rightarrow (1-\lambda)(\lambda^2 - 1) + 1(1-\lambda) = 0$

$\Rightarrow \lambda^2 - 1 - \lambda^3 + \lambda + 1 - \lambda = 0$

$\Rightarrow \lambda^2 - \lambda^3 = 0$

$\Rightarrow \lambda^2(1-\lambda) = 0$

$\Rightarrow \lambda = 0, 0, 1$

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

$\Rightarrow x_1 + x_2 + x_3 = 0$ - (i)

$-x_1 - x_2 - x_3 = 0$ - (ii)

$x_3 = 0$ - (iii)

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

$x_2 + x_3 = 0$ - (i)

$-x_1 - 2x_2 - x_3 = 0$ - (ii)

$0 = 0$ - (iii)

$\boxed{x_3 = -x_2}$ or $\boxed{x_2 = -x_3}$

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

$-x_1 - x_2 = 0$

$\boxed{-x_2 = x_1}$ (or) $\boxed{x_2 = -x_1}$

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., $E = \{2,4,6\}$.

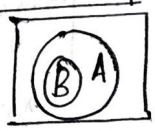
→ **Probability of an event**: $P(E) = \frac{n(E)}{n(S)} = \frac{3}{6} = \frac{1}{2}$

→ **Properties of Probability**:

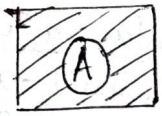
(i) Outcomes - $O_1, O_2 \ldots O_n$ → Probability of outcomes, $P(O_1) + P(O_2) + \ldots P(O_n) = 1$

$$\boxed{\Sigma P(O) = 1}$$

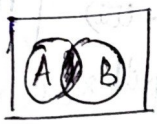
(ii) If A & B are events & $A \subseteq B$ then $\boxed{P(A) \leq P(B)}$



(iii) $\boxed{P(A^c) = 1 - P(A)}$



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



→ **Conditional Probability**:

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

(probability of A given B)
[B occurs earlier, A occurs late]

Eg: 3 coin is tossed

HHH
HHT
HTH
HTT
THT
TTT
TTH
THT
THH

B: H in 1st trial
A: Even no. of H

$P(A) = \frac{3}{8}$ $P(B) = \frac{4}{8} \frac{1}{2}$

$P(A \cap B) = 2/8$

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

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

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

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

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: $P(\text{Girl}) = \frac{30}{50} = \frac{3}{5}$

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

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

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

$$= \frac{2/5}{35/50} = 4/7.$$

⇒ **Baye's Theorem:** $P(A/B) = \frac{P(A \cap B)}{P(B)}$ $\quad P(B/A) = \frac{P(A \cap B)}{P(A)}$

$\Rightarrow P(A \cap B) = P(A/B) \cdot P(B) \quad \Rightarrow P(A \cap B) = P(B/A) \cdot P(A)$

$\Rightarrow P(A|B) \cdot P(B) = P(B|A) \cdot P(A)$

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

Eg: Spam filter
Given: 1% email 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".

$P("free"|spam) = 0.8$
$P("free"|\neg spam) = 0.1$
$P(spam|"free") = ?$

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

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

$= \frac{0.8 \times 0.01}{0.8 \times 0.01 + 0.1 \times 0.99} = \frac{0.0148}{}$

⇒ **Random Variable:**

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

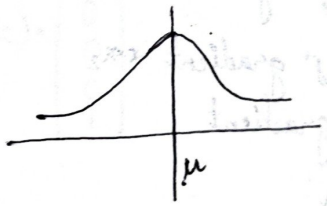
Eg: X : # heads $\quad X(HHH) = 3$
Y : # tails $\quad Y(HHH) = 0$
$Y(HTH) = 1$

⇒ **Probability Density function/Mass function:**

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

⇒ **Normal Distribution:**

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



⇒ **Probability Distribution function:**

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

$F(x) = \int_{-\infty}^{x} f(x)\,dx \qquad f(x) = \frac{d}{dx} F(x)$

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

⇒ **Expectation:**

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

Eg: X = # Head (coin tossed)
$X = 0, X = 1$
$E(X) = 0 \times P(X=0) + 1 \times P(X=1)$
$= 0 + 1 \times 1/2$
$= 1/2$.

Eg: X = # 6 (dice rolled)
$X = 0, X = 1$
$E(X) = 0 \times P(X=0) + 1 \times P(X=1)$
$= 0 \times 5/6 + 1 \times 1/6$
$= 1/6$.

⇒ Properties of Expectation :

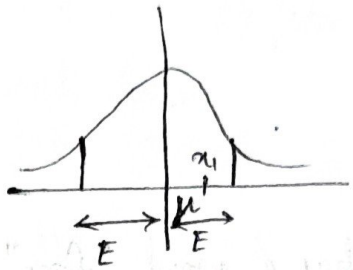
i) $E(c) = c$ [c = constant]

ii) $E(cX) = cE(X)$

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

$E(X) = \mu$
$= \text{mean}$ $\left[\text{Mean} = \frac{x_1 + x_2 + \cdots x_n}{n}\right]$

⇒ Variance :



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

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

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

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

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

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

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

⇒ Co-Variance :

$$\boxed{cov(X,Y) = E(XY) - E(X).E(Y)}$$

$$var(X+Y) = var(X) + var(Y) + 2cov(X,Y)$$

⇒ Activation

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

Error $= y - \hat{y}$

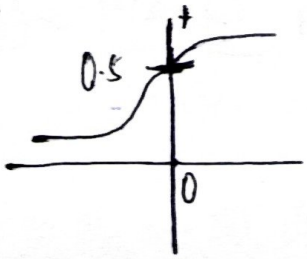
Gradient = Error × learning rate (positive constant)

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

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

$b = b + \text{gradient}$

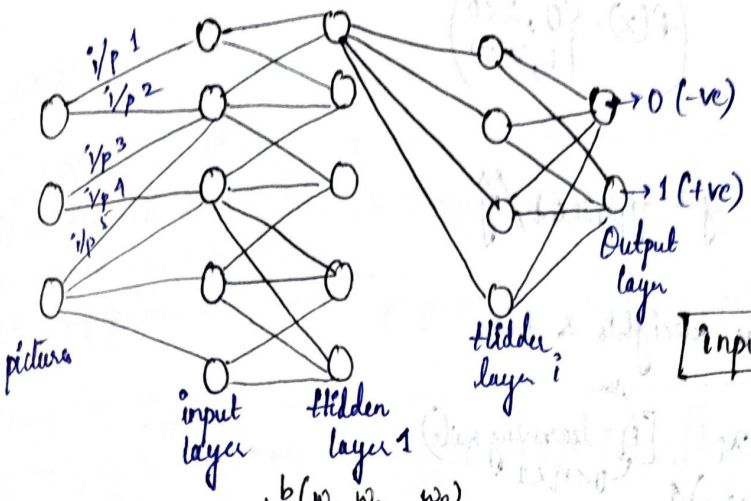
⇒ Sigmoid Function :



08-09-25

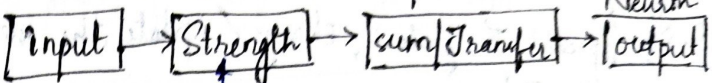
Chapter-3

Building Block of DL → ANN

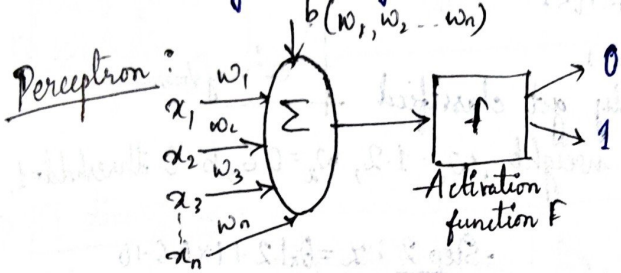


Feed-Forward Neural Network

Activation Function: Activates the nodes of hidden layer as per requirement of the selection of the function.
The nodes are called perceptrons / Artificial Neuron

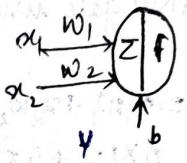


Perceptron:



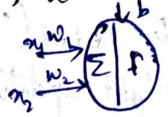
$$x = b + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

$$w_1x_1 + w_2x_2 + b = 0$$



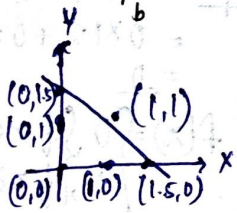
Eg: $w_1 = 1, w_2 = 1, b = -1.5$, AND Gate

x	y	x ∧ y
0	0	0
0	1	0
1	0	0
1	1	1



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

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



$\Rightarrow x_1 + x_2 - 1.5 = 0$

$$f(z) = x \begin{cases} -1.5 + 0 + 0 = -1.5 < 0 \\ -1.5 + 1x0 + 1x1 = -0.5 < 0 \\ -1.5 + 1x1 + 0x1 = -0.5 < 0 \end{cases}$$

$w = [b, w_0, w_1, \ldots w_n]$

$w = [-1.5, 1, 1]$

$$x = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad (b \;\; x_1 \;\; x_2)$$

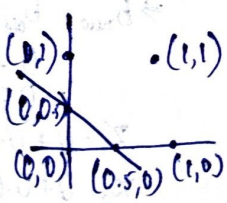
$$x = xw^T = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}_{4\times3} \begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix}_{3\times1} = \begin{bmatrix} -1.5 \\ -0.5 \\ -0.5 \\ 0.5 \end{bmatrix} \Rightarrow f(z) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Eg: Above eg in OR Gate

$x_1 + x_2 - 0.5 = 0$

$w_1x_1 + w_2x_2 + b = 0$

$x = -0.5 + 1.x_1 + 1.x_2$



$$w = [-0.5 \;\; 1 \;\; 1]$$

$$x = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$xw^T = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 0.5 \\ 0.5 \\ 1.5 \end{bmatrix}$$

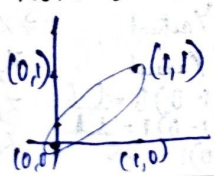
$$\Rightarrow f(z) = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

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

$\boxed{f(z) = xw^T}$ 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.

10-09-25

Perceptron training Rule

1. Randomly choose initial weights & bias. $\left(f(z)=\begin{cases}0, z<0\\1, z\geq 0\end{cases}\right)$
2. Find the z. $\left(z = b + \sum_{i=1}^{n} x_i w_i = \sum_{i=1}^{n} x_i w_i\right)$
3. Apply the activation function on z to get $y_{predicted}$ / $\hat{y}$
4. Compare $y_{predicted}$ with y_{actual}
5. If $y_{predict} \neq y_{actual}$, change / update the weights & bias.

Updation Rule:

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

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

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

⊙ very slow, actual value

Q) Design a perceptron for AND gate using initial weight, $w_1 = 1.2$, $w_2 = 0.6$, $b = 0$. threshold = 1, & $\eta = 0.5$.

x_1	x_2	$y = x_1 \wedge x_2$ (y_{actual})
0	0	0
0	1	0
1	0	0
1	1	1

Truth table for AND gate.

Step 1: $z = x_1w_1 + x_2w_2 + b$
$= 0\times1.2 + 0\times0.6 + 0$
$= 0$

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

$y_{predict} = 0 = y_{actual}$

⇒ No weight updation

Step 2: $z = 0\times1.2 + 1\times0.6 + 0$
$= 0.6$

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

⇒ No weight updation.

Step 3: $z = 1\times1.2 + 0\times0.6 + 0$
$= 1.2$

$y_{pred} = 1 \neq y_{actual}^{(0)}$ ($z \geq 1.2$)

$\Rightarrow w_{1new} = w_{1old} + \eta(y_{actual} - y_{pred})x_1$
$= 1.2 + 0.5(0-1)1$
$= 0.7$

$\Rightarrow w_{2new} = w_{2old} + \eta(y_{actual} - y_{pred})x_2$
$= 0.6$

$\Rightarrow b_{new} = b_{old} + \eta(y_{actual} - y_{pred})$
$= 0 + 0.5(0-1)$
$= -0.5$

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

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

$z_3 = 0.7\times1 + 0\times0.6 - 0.5 = 0.2 < 1$, $\hat{y} = 0 = y_{pred}$

⇒ No updation

Step 4: $z = 0.7\times1 + 0.6\times1 - 0.5$
$= 0.8 < 1$

$y_{pred} = 0 \neq y_{actual}^{(1)}$

$\Rightarrow w_{1new} = 0.7 + 0.5(1-0)1$
$= 1.2$

$\Rightarrow w_{2new} = 0.6 + 0.5(1-0)1$
$= 1.1$

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

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

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

⇒ Updation needed

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

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	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

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

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

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

$\Rightarrow w_{2new} = 0.6 + (1-0)1 = 1.1$

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

$\Rightarrow z_1 = 0.5 \Rightarrow y_{1pred} = y_{1actual}$

$z_2 = 1.1 + 0.5 = 1.6 \Rightarrow y_{2pred} = y_{2actual}$

$z_3 = 0.6 + 0.5 = 1.1 \Rightarrow y_{3pred} = y_{3actual}$

$z_4 = 1.1 + 0.6 + 0.5 = 2.2 \Rightarrow y_{4pred} = y_{4actual}$

$w_1 x_1 + w_2 x_2 + 0.6 = 0$

11-09-25

⇒ **Entropy:** $p(x_i)$

It is denoted by $H(x)$.
It is used for calculating back propagation or forward propagation.

$$\text{Entropy}: H(x) = E_{p(x)}\left(\log_2 \frac{1}{p(x)}\right) = \sum_i p(x_i) \log \frac{1}{p(x_i)} = -\sum \left(p(x_i) \log_2 p(x_i)\right) \quad \left[E(x) = \sum x_i p(x_i)\right]$$

$$= -E(\log_2 p(x))$$

⇒ **Cross Entropy:**

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

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

⇒ **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.

$$\text{Entropy} = -\sum p(x_i) \log_2 p(x_i)$$
$$= -\left[p(X) \log_2 p(X) + p(Y) \log_2 p(Y) + p(Z) \log_2 p(Z)\right]$$
$$= -\left[0.7 \log_2 (0.7) + 0.2 \log_2 (0.2) + 0.1 \log_2 (0.1)\right]$$

Q) $p(X) = 0.7$ | $q(X) = 0.6$
$p(Y) = 0.2$ | $q(Y) = 0.3$
$p(Z) = 0.1$ | $q(Z) = 0.1$

Compute Cross Entropy & KL divergence.

$$CE(p\|q) = -\sum p(x_i) \log q(x_i)$$
$$= -\left[0.7 \times \log_2 0.6 + 0.2 \times \log_2 0.3 + 0.1 \times \log_2 0.1\right]$$

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

$$= 0.7 \times \log \frac{p(x_i)}{q(x_i)}_{i=1} + 0.2 \log \frac{p(x_i)}{q(x_i)}_{i=2} + 0.1 \log \frac{p(x_i)}{q(x_i)}_{i=3}$$

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

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.

$b=0$, $z = x_1w_1 + x_2w_2 + x_3w_3$

$= 2-4+0 = -2$

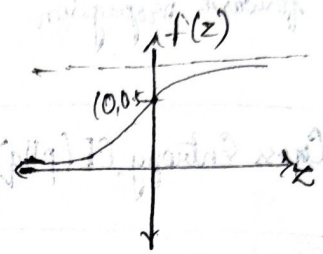
$f(z) = \begin{cases} 0 & \text{if } z<0 \\ 1 & \text{if } z \geq 0 \end{cases}$

$y = f(z) = 0$

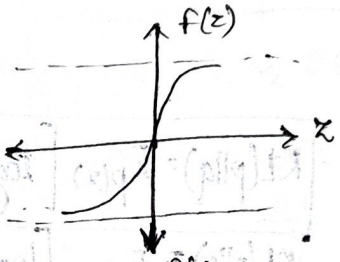
⇒ Activation function

i) Step function
ii) Sigmoid function
iii) Tanh function
iv) ReLU function

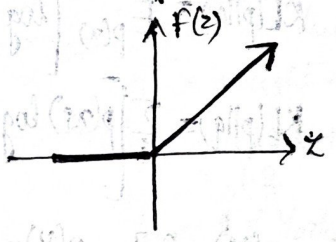
→ Sigmoid function: $f(z) = \frac{1}{1+e^{-z}}$; $0 < f(z) < 1$



→ 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 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.

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

(expectation)

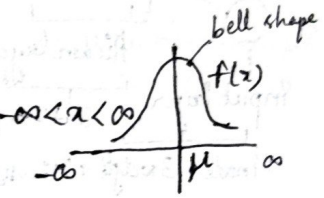
$$= E_{p(x)}\left[\log_2 \frac{1}{q(x)}\right] - E_{p(x)}\left[\log_2 \frac{1}{p(x)}\right] \quad \because E(X \pm Y) = E(X) \pm E(Y)$$

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

⇒ Continuous distribution function:

1. Uniform distribution function: $f(x) = \begin{cases} \frac{1}{b-a}, & a \le x \le b \\ 0, & \text{otherwise} \end{cases}$

2. Normal distribution function (Gaussian df): $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma}}, \quad -\infty < x < \infty$



μ = mean
σ^2 = variance

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

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

$$E(x) = \int_x x f(x)\,dx \quad \text{[Expectation]} \qquad H(f(x)) = -\int_x f(x) \log_2 f(x)\,dx \quad \text{[Entropy]}$$

$f(x) = f(x = x)$

$$Var(x) = E\left[(x-\mu)^2\right] \quad \text{[Variance]} \qquad CE(f(x)\|g(x)) = -\int_x f(x) \log_2 g(x)\,dx \quad \text{[Cross Entropy]}$$

$$= \int_x (x-\mu)^2 f(x)\,dx \qquad KL(f(x)\|g(x)) = \int_x f(x) \log_2 \left(\frac{f(x)}{g(x)}\right) dx \quad \text{[KL-divergence]}$$

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

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

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

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

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

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

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

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

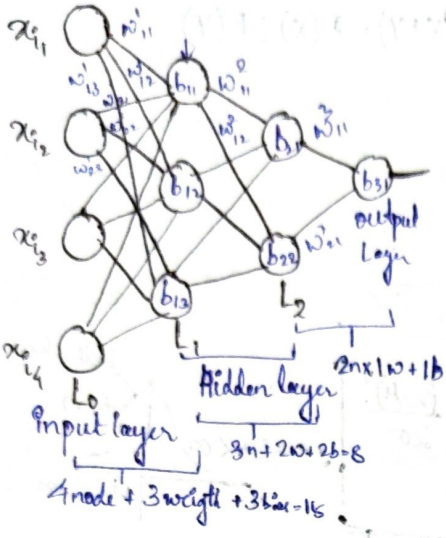
$$\Rightarrow H(g(x)) = -\int_x \tfrac{1}{2} \log_2\left(\tfrac{1}{2}\right) dx = -\frac{1}{2}(-1)\left[x\right]_0^2 = \frac{1}{2}[2-0] = 1$$

$$CE(f(x)\|g(x)) = -\int_x 1 \cdot \log_2\left(\tfrac{1}{2}\right) dx = -\left[x\right]_x (-1).$$

15-09-25

Chapter-4

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



cgpa(x_1)	iq(x_2)	attendance(x_3)	Hours(x_4)
8 x_{11}	7 x_{12}	5 x_{13}	6
9 x_{21}	8 x_{22}	3 x_{23}	8
5 x_{31}	5 x_{32}	3 x_{33}	3

Total learnable parameter = 15+8+3 = 26

* If there are 'n' no. of nodes & 'm' no. of neurons, total learnable parameter - $\boxed{n \times m + m}$

Layer 0

$$\begin{bmatrix} w^1_{11} & w^1_{12} & w^1_{13} \\ w^1_{21} & w^1_{22} & w^1_{23} \\ w^1_{31} & w^1_{32} & w^1_{33} \\ w^1_{41} & w^1_{42} & w^1_{43} \end{bmatrix}^T \begin{bmatrix} x_{i_1} \\ x_{i_2} \\ x_{i_3} \\ x_{i_4} \end{bmatrix}_{4\times 1} + \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix}_{3\times 1} = \begin{bmatrix} O_{11} \\ O_{12} \\ O_{13} \end{bmatrix} \Rightarrow \begin{bmatrix} w^1_{11}x_{i_1} + w^1_{21}x_{i_2} + w^1_{31}x_{i_3} + w^1_{41}x_{i_4} \\ \vdots \\ \vdots \end{bmatrix} + \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix} = \begin{bmatrix} O_{11} \\ O_{12} \\ O_{13} \end{bmatrix}$$

4×3: no. of input × no. of neurons in hidden layer

Layer 1

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

Layer 2

$$\begin{bmatrix} w^3_{11} \\ w^3_{21} \end{bmatrix}^T \begin{bmatrix} O_{21} \\ O_{22} \end{bmatrix} + b_{31} = O_{31} \rightarrow \text{way of forward propagation}$$

$\Rightarrow O_{31} = w^3_{11}O_{21} + w^3_{21}O_{22} + b_{31}$

$a^{[1]} = a\left[w^{[1]} \cdot a^{[0]} + b^{[1]}\right]$
$a^{[2]} = a\left[w^{[2]} \cdot a^{[1]} + b^{[2]}\right]$
$a^{[3]} = a\left[w^{[3]} \cdot a^{[2]} + b^{[3]}\right]$

$a^{[3]} \Rightarrow O_{31} = a\{w^{[3]} \cdot a(w^{[2]}a^{[1]} + b^{[2]}) + b^{[3]}\}$
$= a\{w^{[3]} \cdot a(w^{[2]} \cdot a(w^{[1]} \cdot a^{[0]} + b^{[1]}) + b^{[2]}) + b^{[3]}\}$

So, multilayer perceptron is a cascade of functions

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

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

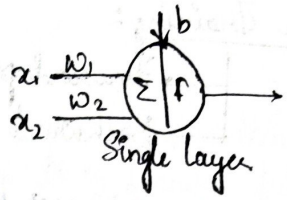
* Without using objective functions in the hidden layers, it become linear. So, we need something non-linear.

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 layer.

Root mean sq error: $\sqrt{\frac{\Sigma - |Y_{pred} - Y_{act}|^2}{n}}$

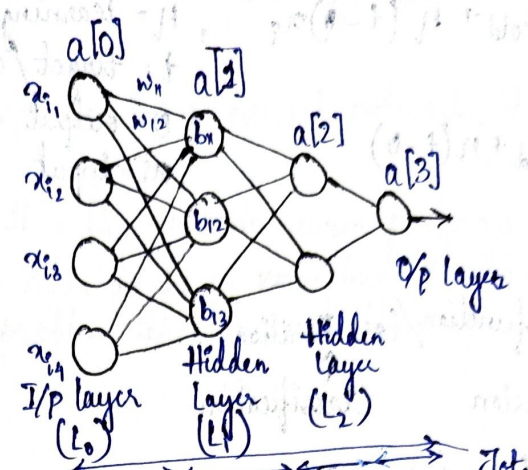
17-09-25

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



Eg:

x_1	x_2	x_3	x_4	Y
x_{11} 8	7 x_{21}	5 x_{31}	9	1
x_{12} 9	8 x_{22}	6 x_{32}	8	1
x_{13} 5	5 x_{23}	8 x_{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

→ Total no. of weight = 4×3 = 12 → Total parameter = 6+2 = 8 → Total parameter = 2+1 = 3

→ bias = 3

→ Total parameter = 15

Total learnable parameters = 15+8+3 = 26

For layer 1:
$$\underset{3\times4}{\begin{bmatrix} w_{11}^1 & w_{21}^1 & w_{31}^1 & w_{41}^1 \\ w_{12}^1 & w_{22}^1 & w_{32}^1 & w_{42}^1 \\ w_{13}^1 & w_{23}^1 & w_{33}^1 & w_{43}^1 \end{bmatrix}} \underset{4\times1}{\begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \\ x_{i4} \end{bmatrix}} + \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix} = \begin{bmatrix} O_{11} \\ O_{12} \\ O_{13} \end{bmatrix}$$

For layer 2:
$$\begin{bmatrix} w_{11}^2 & w_{21}^2 & w_{31}^2 \\ w_{12}^2 & w_{22}^1 & w_{32}^2 \end{bmatrix} \begin{bmatrix} O_{11} \\ O_{12} \\ O_{13} \end{bmatrix} + \begin{bmatrix} b_{21} \\ b_{22} \end{bmatrix} = \begin{bmatrix} O_{21} \\ O_{22} \end{bmatrix}$$

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

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

$a_{31} = f(w^{[3]} + f(w^{[2]} f(w^{[1]} . a^{[0]} + b^{[1]}) + b^{[2]})$ → So, multi-layer perceptron can be written as cascade of function.

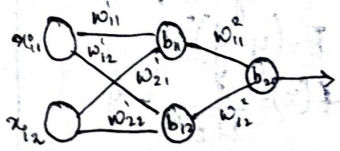
$a^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$

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

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

$= wx + b$ → If activation is not present then we cannot find non-linearity.

Eg:



Total learnable parameter = 6 + 3 = 9
(weight, bias)

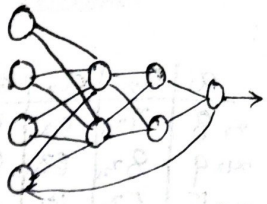
⇒ **Perceptron Training:**

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

→ Backward propagation If $y_{pred} \neq y_{actual}$ Updates weight & bias, when loss is high.

$w_{new} = w_{old} + \eta (t - o) x_i$, η = learning rate, t = target/actual value, o = output, x_i = input

$b_{new} = b_{old} + \eta (t - o)$



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.

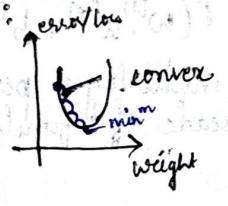
→ Categorical Cross-Entropy Loss: $L = -\frac{1}{N} \cdot \sum_{i=1}^{N} \left(\sum_{j=1}^{C} y_{ic} \log \hat{y}_{ic} \right)$, C = no. of classes

Activation function is SoftMax.

→ **Objective of Cost function:**

- Minimize the error

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



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

→ direction

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

For sigmoid,

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

$$= -2(y - \hat{y}) \times \frac{e^{-z}}{(1 + e^{-z})^2} \times x_1$$

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

$E = (y - \hat{y})^2$

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

$z = w^T x + b$

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

For sigmoid,

$E = (y - \hat{y})^2$

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

$z = w_1 x_1 + w_2 x_2 + b$

22-09-25

⇒ Activation Function :

An ANN, each neuron form a weighted sum of its inputs & passes the resulting scaler 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 :

i) Non-linear
ii) Differentiable
iii) Computationally inexpensive
iv) Normalized
v) Non-saturating function

→ Sigmoid function :

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

Advantage :

(i) It squeezes the data from $(-\infty, \infty)$ to $(0,1)$ & we can treat it as an probability function & we can apply it on the output layer when we are dealing with binary classification function.

(ii) It is a non-linear, so we can capture the non-linear pattern in the data.

Disadvantage :

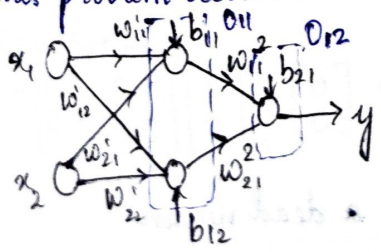
(i) It is a saturating function, i.e, it ranges from $-\infty$ to ∞ & squeezes to 0-1 & due to saturating properties it faces vanishing gradient problem.

(ii) The sigmoid function is not normalized function or it is a non-zero centered & it is expentive.

→ 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

$$\frac{\partial L}{\partial w_{11}^1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_{21}} \cdot \frac{\partial z_{21}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial z_{11}} \cdot \frac{\partial z_{11}}{\partial w_{11}}$$

$$= 0.1 \times 0.1 \times 0.1 \times 0.1 \times 0.1$$

$$= 1 \times 10^{-5}$$

$$z_{11} = x_1 w_{11}^1 + x_2 w_{21}^1 + b_{11} \quad \left[w_{11_{new}}^1 = w_{11_{old}}^1 - \eta \frac{\partial L}{\partial w_{11}} \right]$$

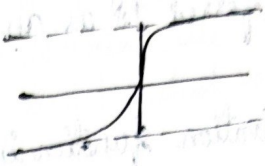
$$z_{21} = o_{11}^1 w_{11}^2 + o_{21}^1 w_{22}^1 + b_{21}$$

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

24-09-25

→ **Tanh**: $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, $x \in (-\infty, \infty) \Rightarrow -1 < f(x) < 1$

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.

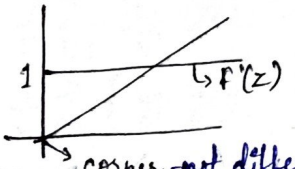
Disadvantage:

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

→ **ReLU**: Used in hidden layer

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

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



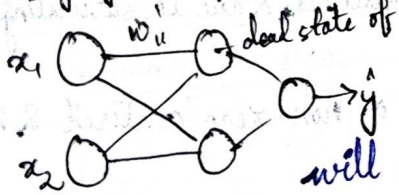
Advantage:

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 $z=0$ → dead
ii) It is not a zero-centered function.
iii) It has Dying-ReLU problem.

→ **Dying ReLU problem.**

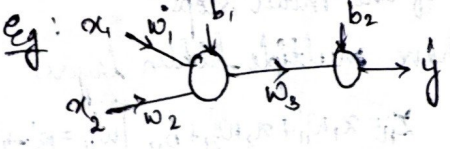


$$w'_{11_{new}} = w'_{11_{old}} - \eta \frac{\partial L}{\partial w}$$

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.

Eg:



$z_1 = w_1 x_1 + w_2 x_2 + b_1$

If $z_1 < 0$, then $f(z_1) = 0$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1} = 0$$

$$w_{1,new} = w_{1,old} - \eta \frac{\partial L}{\partial w_1} = w_{1,old}$$

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

→ Leaky ReLU

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

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

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

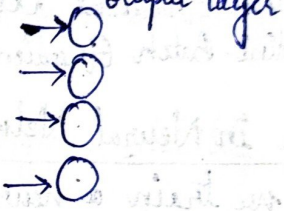
Advantage

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 trainning 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 = 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



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

$x_1 \to 0.5$
$x_2 \to 1.5$
$x_3 \to 0.1$

$f(z_1) = \frac{e^{0.5}}{e^{0.5} + e^{1.5} + e^{0.1}}$

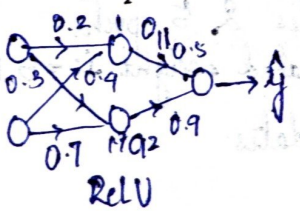
27-09-25

Q1) Consider a small network : Input $x = [1\ 2]^T$, First layer weight, $w_1 = \begin{bmatrix} 0.2 & 0.4 \\ 0.3 & 0.7 \end{bmatrix}$, $b_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$
Second layer weight, $w_2 = \begin{bmatrix} 0.5 \\ 0.9 \end{bmatrix}^T$, $b_2 = 0$. 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 $\frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial w_1}$ (4) Update w_1, w_2, b_1, b_2

$z_1 = w_1 x_1 + b_1$

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



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

$\hat{y} = 1\times0.5 + 1.7\times0.9$

$z_2 = w_2 o_1 + b_2 = \begin{bmatrix}0.5 & 0.9\end{bmatrix}\begin{bmatrix}1\\1.7\end{bmatrix} + 0$

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

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

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

$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2} = \left[\frac{\partial z_2}{\partial w_2} = \frac{\partial}{\partial w_2}(w_2 o_1 + b_2) = o_1\right]$

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

$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b_2} = -2(y-\hat{y}) \Rightarrow \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} \cdot \frac{\partial z_2}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$ $\left[\frac{\partial z_1}{\partial w_1} = \frac{\partial}{\partial w_1}(\right.$

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

$= \begin{bmatrix}0.5\\0.1\end{bmatrix}_{2\times1}\begin{bmatrix}1 & 2\end{bmatrix}_{1\times2} = \begin{bmatrix}\quad\end{bmatrix}_{2\times2}$

6-10-25

⇒ Types of Gradient Descent:

i) Statistic 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 ŷ [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 funct^n 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 i/p 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 × 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 (in 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

08-10-25

Q) Suppose we are training a simple linear regression model $y = wx$ using gradient descent
Training data :

	x	y
x_1	1	2
x_2	2	4
x_3	3	6

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

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

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

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

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

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

(b) $w_{old} = 0$

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

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

$w = 0.2 - 0.1(-7.2) = \underline{0.92}$

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

$w = 0.92 - 0.1(-9.72) = \underline{1.892}$

So, after one full pass or epoch using stochastic GD, the weight $w = \underline{1.892}$

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 → $m=2$ (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.

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

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

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

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

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

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

$\hat{y}_1 = wx_1 + b$
$= 0\times 1 + 0 = 0$
$\hat{y}_2 = wx_2 + b = 0$

$\frac{\partial L}{\partial b} = -\frac{1}{2}\left[(y_1 - \hat{y}_1) + (y_2 - \hat{y}_2)\right] = -\frac{1}{2}[2+4] = \underline{-3}$

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

$b_{new} = b_{old} - \eta\frac{\partial L}{\partial b} = 0 - 0.1(-3) = \underline{0.3}$.

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

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

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

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

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

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

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

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

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

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

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

$\hat{y}_1 = wx_1 + b = 2.27\times 1 + 0.796 = \underline{3.065}$
$\hat{y}_2 = 2.27\times 2 + 0.796 = \underline{5.335}$
$\hat{y}_3 = 2.27\times 3 + 0.795 = \underline{7.605}$
$\hat{y}_4 = 2.27\times 4 + 0.795 = \underline{9.875}$

09-10-25

→ Overfitting:

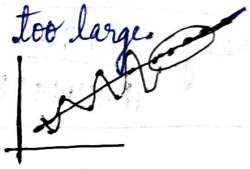
It happens when a model learn the noise & random fluctuations in the training data instead of the true underlying patterns.

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 epoch aspects.

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 $\left[L = (y_i - \hat{y}_i)^2\right]$
L1 regularization: Force some weights to become zero.

ii) Dropout: Randomly "turnoff" a fraction of neurons during training (eg. 20% - 50%). This prevents the network from co-adopting 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:

Predicted output \ Actual output	1	0
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.

Eg:

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	0
1	1	1
0	2	1

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

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

$$\text{Recall} = \frac{TP}{TP+FN}$$ → Total no. of sample which as actually positive

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

Q)

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

Find Accuracy & F1 score.

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

$$\text{Precision} = \frac{40}{40+10} = \frac{40}{50} = 0.8$$

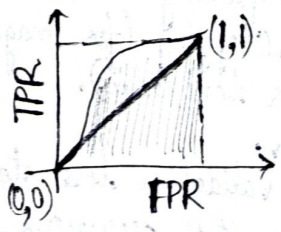
$$\text{Recall} = \frac{40}{40+5} = \frac{40}{45} = 0.889$$

$$\text{F1 score} = \frac{2 \times 0.8 \times 0.88}{0.8+0.88} = \frac{1.408}{1.68} = \frac{176}{165}$$

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

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

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



AUC: Area under curve
ROC: Random Classifier

- Perfect Classifier, AUC = 1
- Random Classifier, AUC = 1/2

15.10.25

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

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

$x = \{[0,0], [0,1], [1,0], [1,1]\}$

$y = \{0,0,0,1\}$.

Step1 : Define activation function

Step2 : Define AND gate truth table

Step3 : Initialize weights & bias

Step4 : Train using perceptron rule

```
for epoch in range (10):
    for i in range (len(x)):
        x = x[i]
        y = y[i]
        z = w x x + b
        y_pred = f(z)
    update w & b
```

Step5 : 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 $\boxed{n!}$ ways to permute neurons without changing the final outputs. If the network has L layers, then there will be $\boxed{(n!)^L}$ many such configurations

3) **Flat regions in the error surface :** Flat region (plateaus) occure when gradients are near zero (stops)
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 base Gradient Descent :** $v_t = \beta v_{t-1} - \eta \frac{\partial L}{\partial \theta_t}$

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

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

Formula :

$$r_i = r_{i-1} + g_i^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{r_i + \varepsilon}} g_i$$

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

$r_i = r_{i-1} + g_i^2 = r_{i-2} + g_{i-1}^2 + g_i^2$
$= r_{i-3} + g_{i-2}^2 + g_{i-1}^2 + g_i^2$
$= g_i^2 + g_{i-1}^2 + \cdots g_i^2$
= Sum of the squares of previous gradients

$\varepsilon = 10^{-8}$
= a small constant, to avoid division by zero.

- 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 avoid the vanishing learning rate.

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

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

ρ = decay factor

$$r_i = \rho r_{i-1} + (1-\rho) g_i^2$$
$$= \rho(r_{i-2} + (1-\rho) g_{i-1}^2) + (1-\rho) g_i^2$$
$$= \rho^2 r_{i-2} + \rho(1-\rho) g_{i-1}^2 + (1-\rho) g_i^2$$
$$= \rho^3 r_{i-3} + \rho^2(1-\rho) g_{i-2}^2 + \rho(1-\rho) g_{i-1}^2 + (1-\rho) g_i^2$$
$$\vdots$$
$$= \rho^i r_0 + \rho^{i-1}(1-\rho) g_1 + \dots$$

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

If $\rho = 0 \Rightarrow r_i = g_i^2$
If $\rho = 1 \Rightarrow r_i = r_{i-1}$

→ 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

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

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)

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

Parameter update:

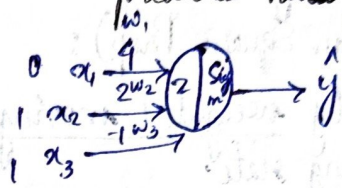
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \varepsilon}} \cdot \hat{m}_t$$

where generally $\beta_1 = 0.1$
$\beta_2 = 0.999$
$\eta = 0.01$
$\varepsilon = 10^{-8}$

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 ⇒ Single Layer Perceptron



$$z = \sum w_i x_i$$

$$= (4\times0) + (2\times1) + (-1\times1)$$

$$= 0 + 2 - 1 = 1$$

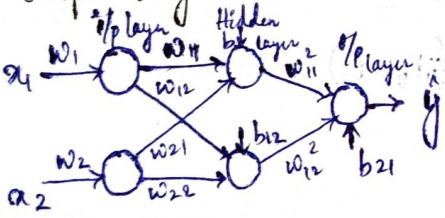
$$\hat{y} = f(z)$$

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

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

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

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



Total no. of learnable parameters = $n \times m + m$ [n = no. of nodes, m = no. of neurons]

$= 6 + 3 = 9$

(weight, bias)

cosine sim(A,B) $= \frac{0.2\times0.3 + 0.5\times0.4 + 0.8\times0.7 + 0.1\times0.2}{\sqrt{0.2^2+0.5^2+0.8^2+0.1^2} * \sqrt{0.3^2+0.4^2+0.7^2+0.2^2}} = \frac{0.84}{0.96 * 0.8839} = \underline{0.98}\ (98\%)$

cosine sim(A,C) $= \frac{0.92}{0.9695} = \underline{0.45}\ (45\%)$ cosine sim(B,C) $= \frac{0.51}{0.88 * 0.97} = \underline{0.59}\ (59\%)$

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): $w_0 = 0$, $\eta = 0.1$, $\beta = 0.9$, $g_1 = 0.5$

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

$w_1 = w_0 - v_1 = 0 - 0.05 = \underline{-0.05}$

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

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

$w_2 = w_1 - v_2 = -0.05 - 0.085 = \underline{0.135}$

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

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

$w_3 = w_2 - v_3 = -0.135 - 0.0565 = \underline{-0.1915}$

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): $\eta_1 = 0.1/\sqrt{1} = 0.1$
$\Delta w_1 = 0.1\ (0.2) = \underline{0.02}$

For 2nd step (t=2): $\eta_2 = 0.1/\sqrt{2} = 0.0707$
$\Delta w_2 = 0.0707\ (0.4) = \underline{0.0283}$

For 3rd step (t=3): $\eta_3 = 0.1/\sqrt{3} = 0.0577$
$\Delta w_3 = 0.0577\ (0.1) = \underline{0.00577}$

So, updated weights: $w_1 = -0.02$
$w_2 = -0.0483$
$w_3 = -0.0541$

③ Given: $\eta = 0.01$
$\beta_1 = 0.9$
$\beta_2 = 0.999$
$\varepsilon = 10^{-8}$
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: $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$

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

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

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

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

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

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

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

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

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	m_t	v_t	Δw_t
1	0.03	0.00009	0.0100
2	0.047	0.00013	0.0097
3	0.0523	0.00014	0.0088

Hessian Matrix:

$L = L(w_1, w_2, \ldots w_n)$ $\quad \frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots \frac{\partial L}{\partial w_n}$

Hessian Matrix (H) $= \begin{bmatrix} \frac{\partial^2 L}{\partial w_1^2} & \frac{\partial^2 L}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 L}{\partial w_1 \partial w_n} \\ \frac{\partial^2 L}{\partial w_2 \partial w_1} & \frac{\partial^2 L}{\partial w_2^2} & \cdots & \frac{\partial^2 L}{\partial w_2 \partial w_n} \\ \frac{\partial^2 L}{\partial w_n \partial w_1} & \frac{\partial^2 L}{\partial w_n \partial w_2} & \cdots & \frac{\partial^2 L}{\partial w_n^2} \end{bmatrix}$

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

- The gradient vector (∇L) tells you the direction of steepest ascent. $\nabla L = \frac{\partial L}{\partial w} = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \frac{\partial L}{\partial w_n} \end{bmatrix}$
- The Hessian matrix tells you how curvature changes, i.e, how fast or how slow the slope changes in each direction.

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

$H = \begin{bmatrix} \frac{\partial^2 L}{\partial w_1^2} & \frac{\partial^2 L}{\partial w_1 \partial w_2} \\ \frac{\partial^2 L}{\partial w_2 \partial w_1} & \frac{\partial^2 L}{\partial w_2^2} \end{bmatrix}$ $\quad \frac{\partial L}{\partial w_1} = 200w_1, \frac{\partial^2 L}{\partial w_1^2} = 200$

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

$= \begin{bmatrix} 200 & 0 \\ 0 & 200 \end{bmatrix}$ $\quad \frac{\partial^2 L}{\partial w_2^2} = 2$, $\frac{\partial^2 L}{\partial w_1 \partial w_2} = 0$

- 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 bee slow
  - Gradients oscillate between steep & flat direction
  - The system is ill-conditioned.

01-11-25

① Given dataset :

x	1	2	3
y	3	5	7

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

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

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

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

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

$\hat{y}_3 = wx_3 + b$

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

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

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

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