Null space

→ Kernel of a matrix

→ When a matrix A is multiplied with a vector x it gives

$$A\vec{x} = 0$$

→ Null space is the subspace of a matrix.

① $A = \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}$

Solve for null space using $Ax = 0$.

$x_1 + x_2 = 0$ — ①

$2x_1 + 2x_2 = 0$ — ②

from this $x_1 = -x_2$

Let $x_1 = c$, $x_2 = -c$

$$\vec{x} = \begin{bmatrix} c \\ -c \end{bmatrix}$$

② Find out null space of A.

$$A = \begin{bmatrix} -1 & 3 & 2 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

Ans.

$$Ax = 0$$

$$\begin{bmatrix} -1 & 3 & 2 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0$$

$R_3 \to R_3 - R_2$, $R_2 = R_2 + R_1$

$$\begin{bmatrix} -1 & 3 & 2 \\ 0 & 4 & 2 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$R_1 = -R$

$R_2 = R_2/4$

$$\begin{bmatrix} 1 & -3 & -2 \\ 0 & 1 & 0.5 \\ 0 & 0 & 0 \end{bmatrix}$$

$R_1 = R_1 + 3R_2$

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

$x_1 - 0.5x_3 = 0$

$x_2 + 0.5x_3 = 0$

Eigen value & Eigen vector.

↓

feature selection

↓

PCA (Principal component analysis)

Eg:

$$A = \begin{bmatrix} 0 & 5 & -10 \\ 0 & 22 & 16 \\ 0 & -9 & -2 \end{bmatrix}$$

$$x_A = \begin{bmatrix} -5 \\ -4 \\ 3 \end{bmatrix}, \quad x_B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad x_C = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Sol":

$$x_A = \begin{bmatrix} -50 \\ -40 \\ 30 \end{bmatrix}, \quad x_B = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad x_C = \begin{bmatrix} -5 \\ 31 \\ 11 \end{bmatrix}$$

$AX_A = 10X_A$; $AX_B = 0X_B$.

$AX = \lambda X$

↳ eigen value

$A * \vec{x} \quad (\vec{x} \neq 0)$

* For the first two cases we can say write $AX = \lambda x$, where λ is a scalar, when this eqn holds for x & λ, the scalar is called as the eigen value of A.

* For case 1 $\lambda = 10$, and
  case 2 $\lambda = 0$

* when $A\vec{x} = \lambda\vec{x}$, $\vec{x} \neq 0$, we can say $\vec{x}$ is an eigen vector of matrix A.

Q) Find out the eigen vector for A

$$A = \begin{bmatrix} 4 & 1 \\ 2 & 3 \end{bmatrix}$$

Sol":

$$Ax = \lambda x$$

$$|A - \lambda I|x = 0$$

$$\Rightarrow \left| \begin{bmatrix} 4 & 1 \\ 2 & 3 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right| \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$

$$\Rightarrow (4-\lambda)(3-\lambda) - 2 = 0$$

$$\Rightarrow 12 - 4\lambda - 3\lambda + \lambda^2 = 0$$

$$\Rightarrow \lambda^2 - 7\lambda + 10 = 0$$

$$\Rightarrow (\lambda - 5)(\lambda - 2) = 0$$

$$\Rightarrow \lambda = 5, 2$$

$$\left(\begin{bmatrix} 4 & 1 \\ 2 & 3 \end{bmatrix} - \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} \right) \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$

$$\begin{bmatrix} -1 & 1 \\ 2 & -2 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 0$$

$$+x_1 = +x_2$$

let $x_1 = 1$, $x_2 = 1$ $\quad \vec{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$\lambda = 2$:

$$\left\{ \begin{bmatrix} 4 & 1 \\ 2 & 3 \end{bmatrix} - \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \right\} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 0$$

$$= \begin{bmatrix} 2 & 1 \\ 2 & 1 \end{bmatrix}$$

$$2x_1 + x_2 = 0$$

$$\Rightarrow \quad x_1 = -x_2/2 .$$

Let $x_2 = 1$

$$X = \begin{bmatrix} -1/2 \\ 1 \end{bmatrix}$$

Ch-2 Fundamentals of probability

* Probability : field of mathematics that quanti-fies uncertainity of wrt event.

→ App^n : image processing, NLP, medical data handling.

→ Benefits : building robust model, handling noisy i/p, decision making,

* Random variable : mathematical rep^n of a variable, where value is generated by randomness.

→ Role of random no. in NN : • weight initializ^n
• stochastic gradient descent
• dropout.

→ It improves convergence of model.

Expectation

$E(x)$ - let x be input distrib^n

$E(x)$ = input distribution multiplied by its prob.

$= \{x_1 \cdot P_1\} + \{x_2 P_2\} + \ldots \{x_n p_n\}$.

Q). A pair of 6 side dice be rolled. let x be the random variable representing the no. of on dice.

Ans. $x = 1, 2, 3, 4, 5, 6$

$P(x) = 1/6$

$E(x) = 1 * 1/6 + \ldots$

$= \frac{21}{6} = \frac{7}{6}$.

Q) A game involves rolling a pair 6 sided die. If the no. is even you win \$10 else loose \$5. Let x be random variable representing amt. won or loss. Find out $E(x)$.

Ans. $x = \{10, 5\}$

$E(x) = \frac{1}{2} \times 10 + \frac{1}{2} \times 5 = \frac{15}{2}$

$E(\text{win}) = 5$; $E(\text{loss}) = 5/2$

$E(x) = 5 - 2.5 = 2.5$

Variable & standard deviation

$\text{var}(\sigma^2) = E(x-\mu)^2 = (x_1-\mu)^2 + \ldots \frac{*}{n}$

where, x = input distr^n

, μ = mean

Q)

Ans.

Q)

Ans

Standard (σ) = $\sqrt{\text{var}}$

Q) Find out pop^n variance of data 5, 7, 9, 10, 14, 15. Find var & std.

Ans. $\mu = 10$

$\sigma^2 = 12.66$; $\sigma = 3.55$

* Entropy ($H(x)$)

$= -P(x) \log_2 P(x)$

$P(x)$ = prob of event x.

↓ uncertainity of event x.

→ It measures an avg. level of uncertainity present in random variable placed outcome.

→ The more unpredictable an event is the higher will be entropy.

coin toss $= -[P(H) \log_2 P(H) + P(T) \log_2 P(H)]$

Q). $P(H) = 0.6$, $P(T) = 0.4$

Find $H(x)$.

Ans $H(x) = 0.9709$

Cross Entropy Loss

It measures the diff b/w true label & predicted class label probability during data classification.

$L(y, y') = -\Sigma y * \log(y)$

y = actual class

y' = predicted class.

2 marks

Q.) Consider a 2 class classific^n problem. The true label & predicted label of classific^n are given as:

	True label (Y)	Predicted label (Y')
Sample 1	[1, 0]	[0.7, 0.3]
Sample 2	[0, 1]	[0.4, 0.6]

Find out the cross entropy loss.

Ans. Cross entropy = $-\Sigma Y * \log(Y')$

Sample 1 = $-[1 * \log(0.7) + 0 \cdot \log(0.3)]$ = 0.5145

Sample 2 = 0.7369

$L = L_1 + L_2$ = 0.5145 + 0.7369

= 1.2514

Q) There are 3 containers containing shapes such as triangle & circle. The prob. of picking a shape from these 3 containers are given as:

Container No.	Picking Δ	Picking O
C1	26/30	4/30
C2	14/30	16/30
C3	1/30	29/30

Apply KL divergence & find out from which container the picking of shapes are container & why?

Ans. Container C1 = 2.34

C2 = −0.089

C3 = −0.161

C3 will be preferred.

Q). Consider a 3 class classifi" problem, considering cat, dog & bird. There are 2 cases for which the labels are: True label [1, 0, 0]

case 1: Predicted label [0.7, 0.2, 0.1]

case 2: Predicted label [0.9, 0.05, 0.05].

Evaluate which will be predicted.

Ans. $L_1(Y, Y') = 0.5145$

$L_2(Y, Y') = 0.1520$

Loss L_2 < Loss L_1. Thus, M_2 is better.

KL divergence

It is the difference b/w two prob. distri-bution. Represent as:

$$D(P||Q) = \sum P(x) \log_2 \left(\frac{P(x)}{Q(x)}\right)$$

where P & Q are prob. distributions, of random variable.

Q) Let P & Q are given as

$P(x) = [0.4, 0.3, 0.3]$

$Q(x) = [0.2, 0.5, 0.3]$

Find out KL divergence D.

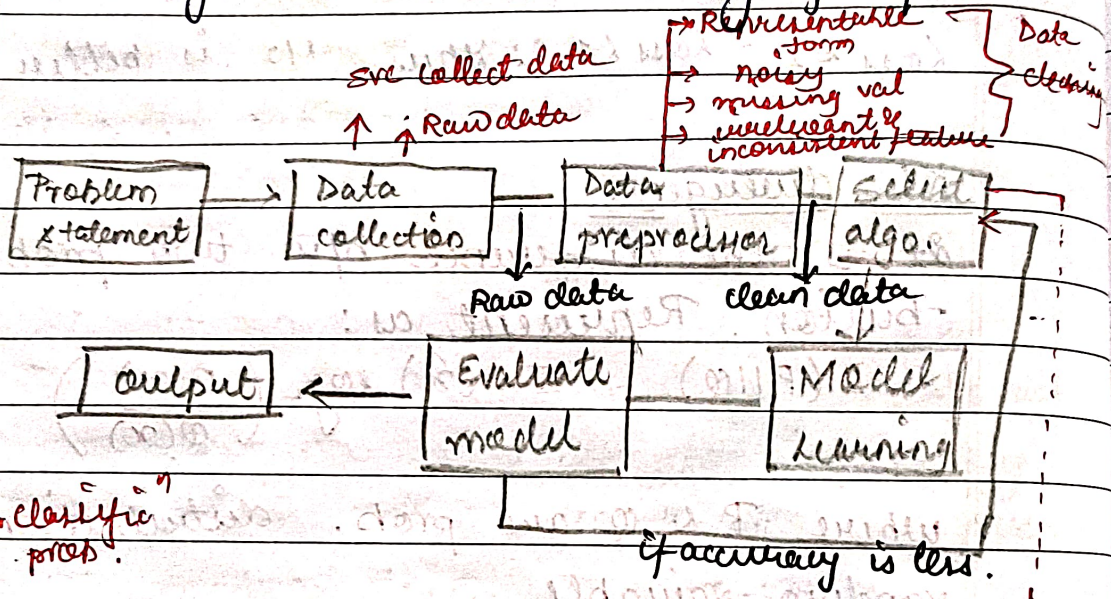
Ans. D = 0.179

8. The Neural Networks (NN)

Def^n of intelligence — learn, understand, adapt

Limitation of traditional system — set of defined instruc^n efficient in performing op^n (arithmetic)

Eg: understand handwriting of a person.



Eg:
- Support vector machine
- Decision tree
- Random forest
- Linear Regression
- ANN
- Naive Bayes classifier

* A machine learning technique is an algo. that is able to learn from data.
* ML enables us to tackle tasks that are too diff. to solve with the fixed program written by human.

classmate
Date
Page

* ML tasks are usually described in terms of how a model should process an example.

Types of Machine Learning

① Supervised learning - labelled data.
② Unsupervised learning - unlabelled data.
③ Semi-supervised learning - few of both.
④ Reinforcement learning - completely diff. from ① & ②.

① Supervised Learning

→ It is the study of algo. that use labelled data in which every sample is associate with class label.
→ Used to discover hidden patterns.

Eg: Classific", Regression

② Unsupervised learning

→ Involves difficult tasks as pre defined labels are not available.

Eg: Clustering, Anomaly Detec", Dimensionality Reduction.

③ Semi-Supervised learning

→ few samples of labelled & few of unlabell-ed.
→ It fills the gap b/w ① & ②. It reveals the strength of both approaches by training using labelled data with unlabelled.

④ Reinforcement Learning

→ It is completely different from first two.

→ It adapts an agent as it interacts with surroundings & learns.

→ The agent learning is through experiments. Thus maybe rewarded when performs desired task.

Eg: Recommend" model, Gaming AI.

Confusion matrix

Actual outcome	Predicted outcome
1	1 (TP)
0	0 (TN)
1	0 (FP)
0	1 (FN)

⇒ True positive (TP) - +ve sample correctly identified

⇒ True negative (TN) - -ve sample correctly identified

⇒ False positive (FP) - -ve sample incorrectly identified

⇒ False negative (FN) - +ve sample incorrectly identified

Actual class	Predicted class: 1	Predicted class: 0
1	TP	FN
0	FP	TN

① High value of TP & TN = Good performance

② High value of FP & = Over +ve prediction (unnecessary action)

③ High value of FN = under predi" of positive samples.

Q.) A confusion matrix wrt. COVID dataset is as follows: TP = 30, TN = 950, FP = 15 and FN = 5. How to interpret this outcome.

Ans. Total no. of samples = 1000

Total no. of positive prediction = TP + FP
= 30 + 15 = 45.

Total no. of negative prediction = TN + FN
= 950 + 5 = 955.

(b) Find out precision, recall, accuracy, sensitivity, F1 score.

Ans. Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$ = $\frac{980}{1000}$ = 0.98

Precision = $\frac{TP}{TP+FP}$ = $\frac{30}{45}$ = 0.667

Recall = $\frac{TP}{TP+FN}$ = $\frac{30}{35}$ = 0.857

F1 score = 0.75 . = $\frac{2 \times Pre * recall}{Pre + Recall}$

Error = 1 - accuracy = 0.02.

Deep Learning

* It is a sub-field of machine learning.
* It is a type of ANN in which automatic feature selection takes place.
* Explicit feature selection is not reqd.

Date
Page

Advantages

* DL can learn complex pattern, especially high dimensional prob.
* Has the ability to learn from raw data such as image or audio.

Requirements

* Huge amt. of data for model training.
* Requires GPU for modelling.
* Requires more training & processing time.

Types of DL model (sort the type of data it investigates)

① Convolutional Neural Network (CNN)
→ image & video data

② Recurrent Neural Network (RNN)
→ used for sequential data learning
Eg: time series data (temporal data)

Data
- temporal (time comp)
- long term (No time comp).

③ Long short term memory (LSTM).
→ Advance version of RNN → temporal data.
→ used for predict" → NLP

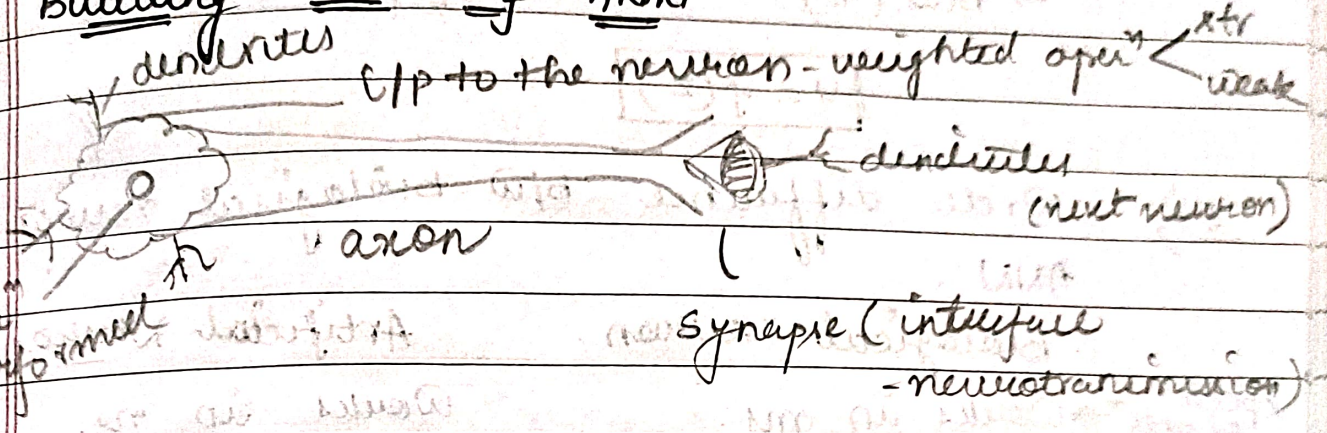
④ Generative Adversial Network (GAN)
→ generate synthetic data (new data)

⑤ Auto encoder
→ noise reduc" → produce new dimension
→ compress data.

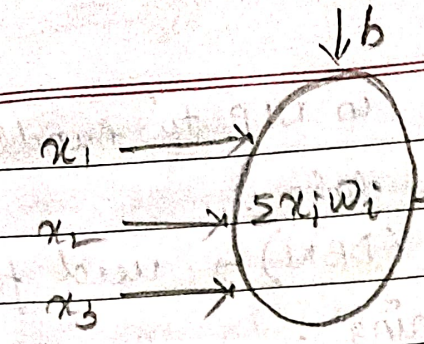
⑥ Transformer: → used in NLP to handle sequence data.

⑦ Deep Belief Network (DBN) → used for unsupervised learning.
Eg: clustering, anomaly detection.

Building Blocks of ANN



* Neuron recieves i/p through mini dendrites.
* Input is either strengthened or weakened depending on how & which dendrite will participate in info. sharing.
* After being weighted by the strength of respective connⁿ the i/p are summed in the cell body.
* The summed value is transformed into a new signal & propagates to axon to other neurons.
* Here, synapses are present, which act as an interface b/w two neurons & helps in neurotransmission.



→ input (x)
→ weight (w)
→ bias (b)
→ activation func^n (z)

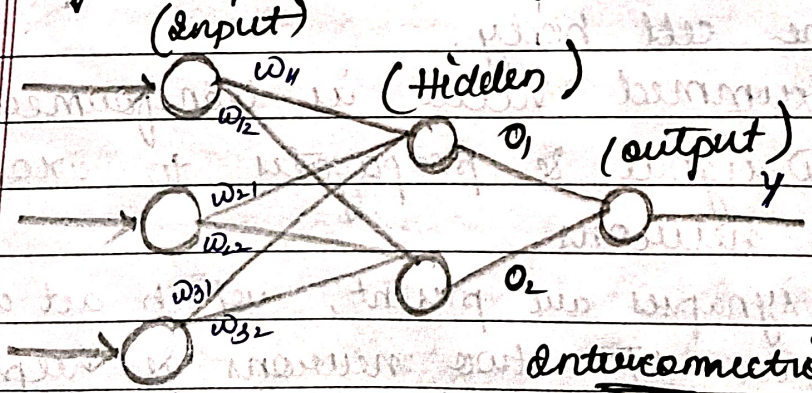
$$z = \sum_{x=1}^{n} x_i w_i + b$$

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

Q.) Write difference b/w biological neuron & ANN.

	Biological Neuron.	Artificial Neuron
speed of oper^n	→ Works in ms.	Works in ns.
weighted i/p proces.	→ Parallel	Parallel.
storage cap.	→ unlimited	limited.
size	→ Huge	comparatively less.
complexity	→ More	comparatively less.
fault tolerance	→ hardly any loss	data loss, w/o failure

Layered Architecture of ANN



Interconnection

(takes i/p from dataset) ① Input layer
(weights are defined) ② Hidden layer (optional)
(③ Output layer

① **Input Layer** : Top layer in ANN. Here, the input data in the form of image pixels, numbers, alphabets is received.

② **Hidden Layer** : Is located in centre of ANN. Runs the variety of mathematical oper" over input data to identify patterns.

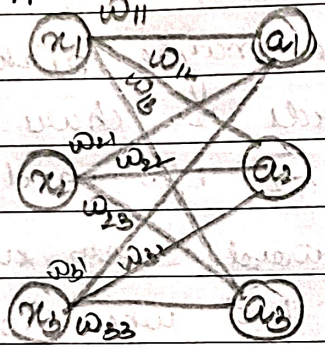
③ **Output Layer** : The computa" in middle layer enable us to generate desired output in the output layer.

* ANN employs a variety of mathematical process-ing stages. It ~~its~~ has a no. of units arranged in no. of layers. A single unit is known as neuron.
* The input layers input units takes a variety of input from the outside env.
* The data then travels to the hidden units, which transforms it into a format that output units may use it.
* Interconnec" maybe defined as the way of processing elements in ANN, are connected to each other.
* These arrangements have always two layers common in every n/w that are: input layer (buffers the I/P signal) & output layer (generates O/P of n/w).

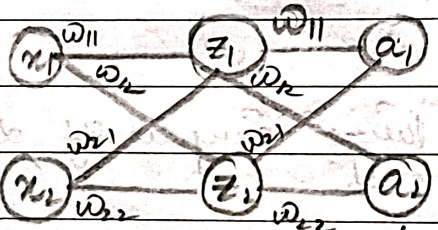
DLUP

* The third layer is ~~out~~ hidden layer, in which neurons are neither kept in input layer nor in output layer.
* These layers are hidden from people who are interfacing from the system. Thus, it is completely a blackbox.
* No. of hidden layers & no. of neurons are used in ANN to introduce non linearity.

(I/P layer) (O/P layer)

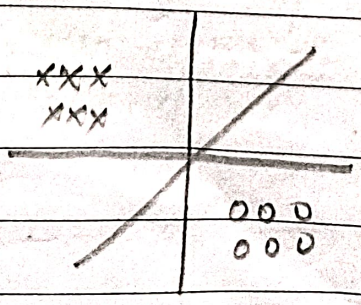
① 

Single layer feed fod neural n/w.
(1 layer FFNN)
- Linear neuron

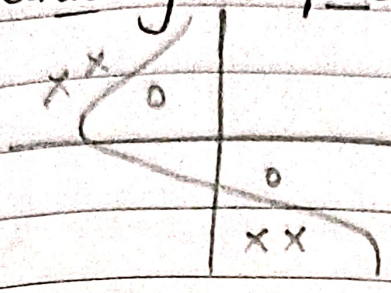
② 

Multilayer feed fod neural n/w
m-layer FFNN

* Types of problem
① linearly separable problem

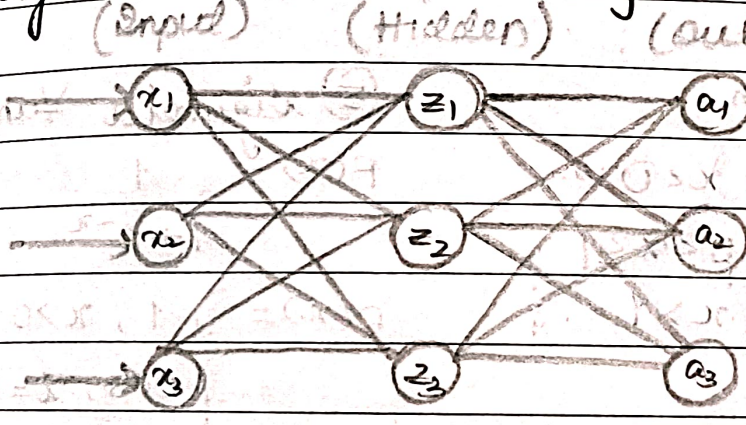


Non linearly separable problem

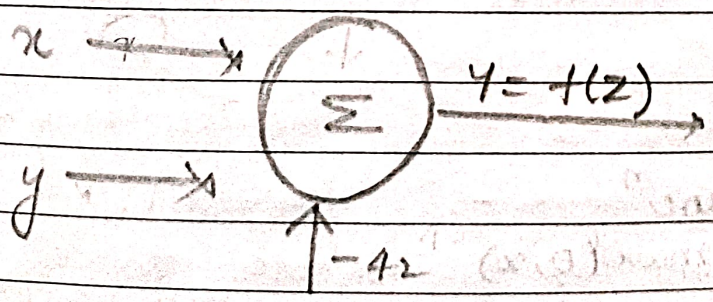


* Linear neurons never use hidden layer in NN.
* In realtime situation problems are non-linear.
* Hidden layers are helpful to extract imp. features from data.
* It is used to learn complex relationship & direct the output towards the desired o/p.

1) Design a feed fwd. NN with 3-layers. Each layer contains 3 no. of neurons.



2) Let $z = 3x + 4y - 42$. Design a NN for this.



$z = w_1 x + w_2 y + b$

$y = f(z)$

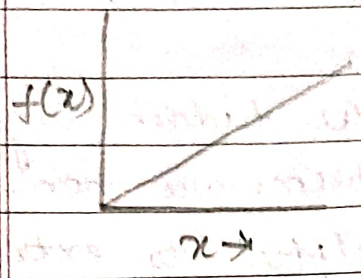
$= f(w_1 x + w_2 y + b)$

$= f(3x + 4y - 42)$

Activation function

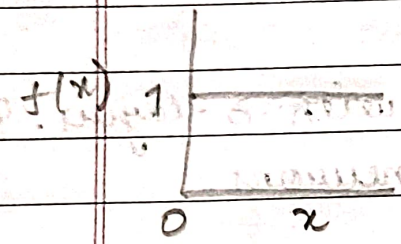
① Identity func^n

$f(x) = x$



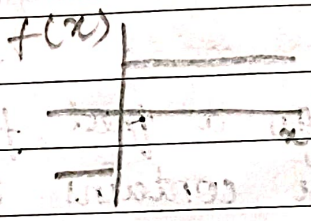
② Binary func^n

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



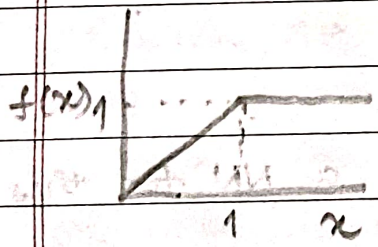
③ Bipolar step func^n

$$f(x) = \begin{cases} 1, & x \geq \theta \\ -1, & x < \theta \end{cases}$$ threshold



④ Ramp func^n

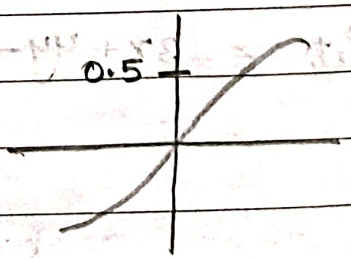
$$f(x) = \begin{cases} 0, & x<0 \\ x, & 0 \leq x \leq 1 \\ 1, & x>1 \end{cases}$$



⑤ Sigmoid func^n

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

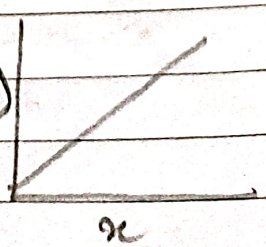
$$f(x) = \begin{cases} 1, & x>0 \\ 0.5, & x=0 \\ 0, & x<0 \end{cases}$$



⑥ ReLU func^n

$f(x) = \max(0, x)$

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



⑦ Tanh function

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

9) Consider a label dataset of size 5x4. The data descrip^n is as follows:

Data =

1	0	0	1
1	0	1	1
0	1	1	0
0	1	0	0
1	1	1	1

The last row represents class label. Given weights of NN are $0.5 (w_1), 0.6 (w_2), 0.4 (w_3)$. Given activation func^n $P(z) = \begin{cases} 0, & z \le 0.5 \\ 1, & z > 0.5 \end{cases}$

Find out classific^n performance of given n/w.

Ans. $z_1 = 0.5 \times 1 = 0.5 = 0 = 0$

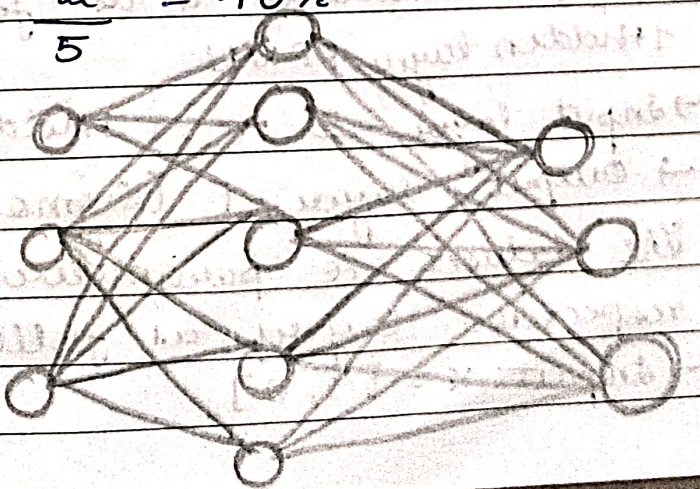
$z_2 = 0.5 + 0.6 = 1.1 = 1$

$z_3 = 0.6 + 0.4 = 1 = 1$

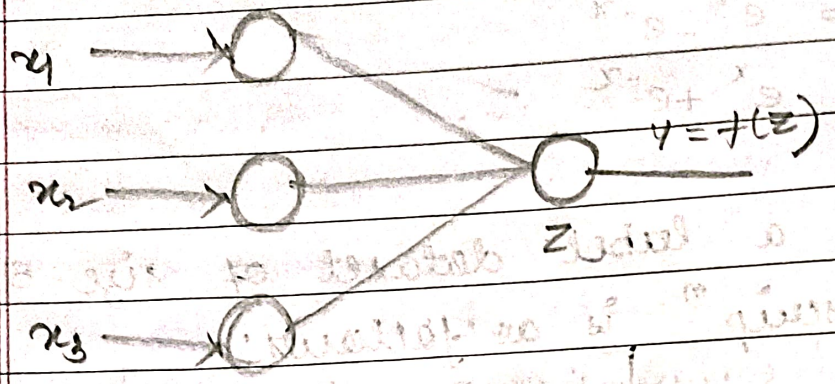
$z_4 = 0.6 = 1$

$z_5 = 1.0 = 1$

Accuracy $= \frac{2}{5} = 40\%$



0.5, -0.6, 0.4



$w_1 = 0.5$

$w_2 = -0.6$

$w_3 = 0.4$

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

Q) Solve the given classific^{n} problem & find out precision & recall. $f(z) = \begin{cases} 1, & z \ge 0 \\ 0, & z < 0 \end{cases}$

Input (x)	z	Actual class	PC
0.1	2x-1 (-0.8)	0	0
0.4	-0.2	0	0
0.5	1	1	1
0.6	0.2	1	1
0.9	0.8	1	1

Ans Precision = 1 ; Recall = 1.

Q) Evaluate the performance of given n/w. The n/w structure is as follows:

1 Hidden layer FFNN

→ Input layer - 2 → Hidden layer - 2 (sigmoid)

→ Output layer - 1 (sigmoid)

The dataset & parameter details for respective layers as follows:

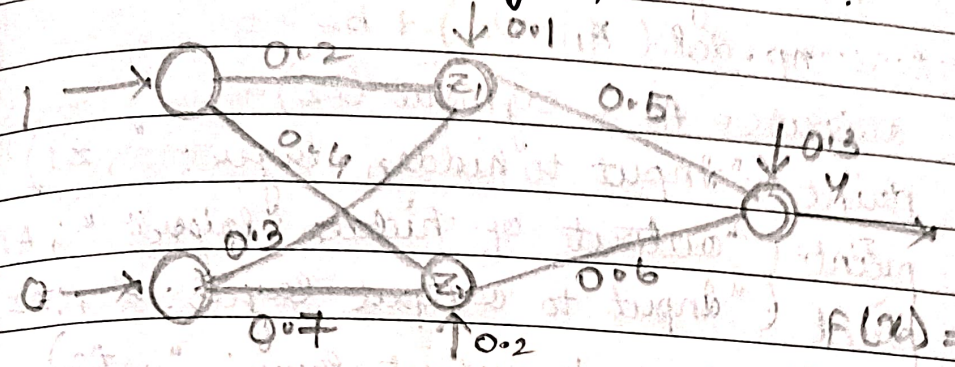
→ Input x = [1, 0]

→ weights [input → hidden layer] : $W^1 = \begin{bmatrix} 0.2 & 0.4 \\ 0.3 & 0.7 \end{bmatrix}$

→ Bias for hidden layer, $b^1 = [0.1, 0.2]$.

→ Weights [hidden → output] = $W^2 = [0.5, 0.6]$

→ Bias for output layer, $b^2 = 0.3$.



$$f(x) = \begin{cases} 1, & x > 0 \\ 0.5, & x = 0 \\ 0, & x < 0 \end{cases}$$

$z_1 = f(1 \times 0.2 + 0 \times 0.3 + 0.1)$

$= f(0.3) = 0.5744$

$z_2 = f(1 \times 0.4 + 0 \times 0.7 + 0.2) = f(0.6) = 0.6456$

$y = (0.5744 \times 0.5) + (0.6456 \times 0.6) + 0.3$

$= f(0.97456) = 0.7260$

Python code

```
import numpy as np
# define sigmoid func"
def sigmoid(x):
    return 1/(1+np.exp(-x))
# initialize data
x = np.array([1,0])
# define weights & bias
w1 = np.array([[0.2,0.4],[0.3,0.7]])
# weights [input → hidden]
b1 = np.array([0.1,0.2])
w2 = np.array([0.5,0.6])
b2 = 0.3
```

Initialization ann

Step 1 : $z_1 = xw_1 + b_1$

$z_1 = np.dot(x, w_1) + b1$

step 2 : $A1 = P(z_1)$

$A1 = sigmoid(z_1)$

$Z_2 = np.dot(A_1, w_2) + b2$

$A_2 = sigmoid(z_2)$

print ("Input to hidden layer: ", z1)

print ("output of hidden layer: ", A1)

print ("Input to output layer: ", z2)

print ("output to output layer: ", A2)

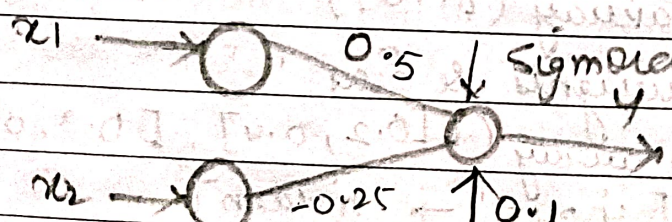
Q) given data & class label of a dataset

$$Data = \begin{bmatrix} 2 & 3 \\ 1 & -1 \\ 0 & 2 \\ 3 & 0 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

a) Design the FFNN with the following n/w details. $w = \begin{bmatrix} 0.5 \\ -0.25 \end{bmatrix}$, $b = 0.1$, AF in O/P layer = sigmoid

b) Evaluate the performance of binary classifi" prob. in form of eval" criteria such as accuracy, error, precision & recall.

Ans.



$$z_1 = \begin{bmatrix} 2 & 3 \\ 1 & -1 \\ 0 & 2 \\ 3 & 0 \end{bmatrix} \begin{bmatrix} 0.5 \\ -0.25 \end{bmatrix}$$

$$= \begin{bmatrix} 0.25 \\ 0.75 \\ -0.5 \\ 1.5 \end{bmatrix} + 0.1 = \begin{bmatrix} 0.35 \\ 0.85 \\ -0.4 \\ 1.6 \end{bmatrix}$$

$$A(z) = \begin{bmatrix} 0.5866 \\ 0.7005 \\ 0.4013 \\ 0.8320 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

TP, FP = 2
FP, TN = 0
FN, FN = 1
FP, TP = 1

$$\text{Accuracy} = \frac{TP}{TP+FP+TN+FN} = \frac{1}{4}$$

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{1}{3}$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{1}{2}$$

$$\text{Error} = 1 - 0.25 = 0.75$$

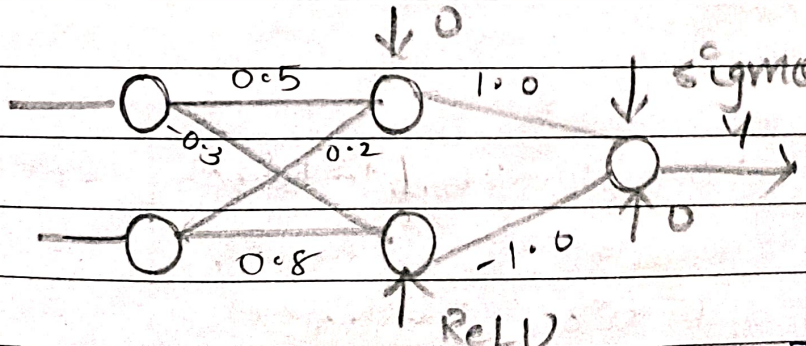
Q) Consider 2 layer FFNN. Input, $x = [1, 2]$

Input layer details as follows:

$$W_1 = \begin{bmatrix} 0.5 & -0.3 \\ 0.2 & 0.8 \end{bmatrix}, \quad b_1 = 0, \quad AF = \text{ReLU}$$

$$W_2 = \begin{bmatrix} 1.0 \\ -1.0 \end{bmatrix}, \quad b_2 = 0, \quad AF = \text{Sigmoid}$$

Ans.



$$z_1 = \begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 0.5 & -0.3 \\ 0.2 & 0.8 \end{bmatrix} = \begin{bmatrix} 0.9 & 1.3 \end{bmatrix}$$

$$z_2 = \begin{bmatrix} 0.9 & 1.3 \end{bmatrix} \begin{bmatrix} 1.0 \\ -1.0 \end{bmatrix} = [-0.4]$$

$$= 0.4013$$

Q Binary classification problem

* Q over single layer FFNN

a) Design a FFNN with the following n/w details

b) Evaluate the performance of binary classif problem in terms of evaluation criteria like accuracy, error, precision, Flscore.

$$X = \begin{bmatrix} 2 & 1 \\ 1 & -1 \\ 0 & 2 \\ 3 & 0 \end{bmatrix}, \quad AC = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$$W_1 = \begin{bmatrix} 0.2 & -0.5 \\ 0.3 & 0.8 \end{bmatrix}, \quad b_1 = [0.1, -0.2]$$

$$W_2 = \begin{bmatrix} 0.4 \\ -0.6 \end{bmatrix}, \quad b_2 = 0.05.$$

AF_1 = ReLU, AF_2 = sigmoid

Ans.

Ans.

Q.) $x = \begin{bmatrix} 1 & 2 \\ 2 & 0 \\ 0 & 1 \end{bmatrix}$ $y = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$

Hidden layer wt. & bias.

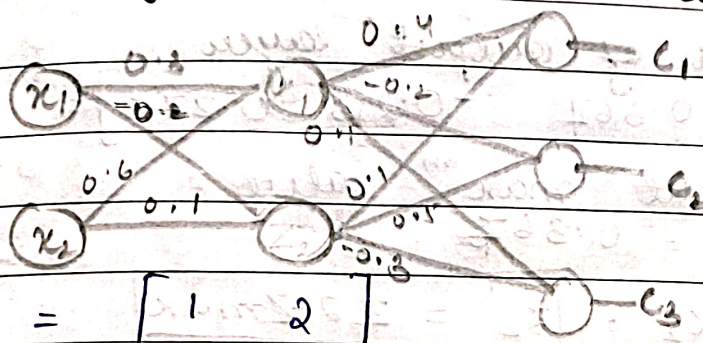
$W_1 = \begin{bmatrix} 0.3 & -0.2 \\ 0.6 & 0.1 \end{bmatrix}$ $b_1 = \begin{bmatrix} 0 \\ 0.5 \end{bmatrix}$ $AF_1 = \tanh$.

output layer wt. & bias.

$W_2 = \begin{bmatrix} 0.4 & -0.2 & 0.1 \\ 0.1 & 0.5 & -0.3 \end{bmatrix}$ $b_2 = \begin{bmatrix} 0 \\ -0.1 \\ 0.2 \end{bmatrix}$

AF_2 = Softmax. Evaluate output of the FFNN.

Ans.



① Construct NN

② $z_1 = xw_1 + b_1$

③ $A_1 = \tanh(z_1)$

④ $z_2 = A_1 w_2 + b_2$

⑤ Softmax (z_2)

② $z_1 = \begin{bmatrix} 1 & 2 \\ 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.3 & -0.2 \\ 0.6 & 0.1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0.5 \end{bmatrix}$

$= \begin{bmatrix} 1.5 & 0 \\ 0.6 & -0.4 \\ 0.6 & 0.1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0.5 \end{bmatrix} \begin{bmatrix} 0. \\ 0.05 \end{bmatrix}$

$= \begin{bmatrix} 1.5 & 0.05 \\ 0.6 & -0.35 \\ 0.6 & 0.15 \end{bmatrix}$

$\tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$.

③ $A_1 = \begin{bmatrix} \tanh(1.5) & \tanh(0.05) \\ \tanh(0.6) & \tanh(-0.35) \\ \tanh(0.6) & \tanh(0.15) \end{bmatrix}$

$= \begin{bmatrix} 0.905 & 0.049 \\ 0.537 & -0.336 \\ 0.537 & 0.148 \end{bmatrix}$ output of hidden layer.

④ $z_2 = \begin{bmatrix} 0.905 & -0.049 \\ 0.537 & -0.336 \\ 0.537 & 0.148 \end{bmatrix} \begin{bmatrix} 0.4 & -0.2 & 0.1 \\ 0.1 & 0.4 & 0.3 \end{bmatrix} + \begin{bmatrix} 0 \\ -0.1 \\ 0.2 \end{bmatrix}$

$= \begin{bmatrix} 0.3669 & -0.176 & 0.075 \\ 0.1812 & -0.275 & 0.154 \\ 0.229 & -0.033 & 0.009 \end{bmatrix} + \begin{bmatrix} 0 \\ -0.1 \\ 0.2 \end{bmatrix}$

$= \begin{bmatrix} 0.366 & -0.256 & 0.275 \\ 0.181 & -0.375 & 0.354 \\ 0.229 & -0.133 & 0.209 \end{bmatrix}$ output

output of output layer

$z_2 = [0.367 \quad -0.256 \quad 0.271]$

Step 1: Find out max^m value

$z_{max} = 0.367$.

Step 2: Shift z_2, $\boxed{z_2' = z2 - z_{max}}$

$z_2' = [0, -0.623, -0.09]$

Step 3: Exponentiate z_2'

$z_2' = e^{z_2'} = [1, 0.536, 0.912]$.

Step 4: Normalize z_2',

Sum = 2.44

$P_1 = \left[\frac{1}{2.44}, \frac{0.536}{2.44}, \frac{0.912}{2.44}\right] = [0.408, 0.218, 0.372]$.

For Row 2:

$z_2 = [0.181 \quad -0.375 \quad 0.354]$

$z_{max} = 0.354$

$z_2' = [-0.173 \quad -0.729 \quad 0]$.

$e^{z_2'} = [0.841, 0.482 \quad 1]$, Sum = 2.323

$P_2 = [0.362 \quad 0.207 \quad 0.4304]$

For Row 3: $[0.229 \quad -0.133 \quad 0.204]$

$z_2' = [0, \quad -0.362, \quad -0.02]$

$e^{z_2'} = [1 \quad 0.696 \quad 0.980]$, Sum = 2.676

$P_3 = [0.373 \quad 0.260 \quad 0.366]$

$$P = \begin{bmatrix} \text{Class 1} & \text{Class 2} & \text{Class 3} \\ 0.408 \checkmark & 0.219 & 0.372 \\ 0.362 & 0.207 & 0.430 \checkmark \\ 0.373 \checkmark & 0.260 & 0.366 \end{bmatrix}$$

* Evaluate which col. has highest contribu^n towards class label.

$$y = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}, \quad y' = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

Solving Regression Problem.

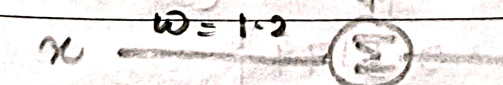
1.) Find out the performance of given n/w in terms of MSE.

① $x = 2.0$, $y = 3.5$, $w = 1.2$, $b = 0.1$

$$MSE = \frac{1}{2} \sum_{i=1}^{n} (y_i - y_i')^2$$

$z = xw + b = 2 \times 1.2 + 0.1 = 2.5$

$MSE = \frac{1}{2} (3.5 - 2.5)^2 = 0.5$



② $x = [1, 2]$, $y = 4$, $w_1 = \begin{bmatrix} 0.5 & -0.3 \\ 0.2 & 0.4 \end{bmatrix}$

$b_1 = [0 \quad 0.1]$, AF = ReLU

$w_2 = \begin{bmatrix} 1 \\ -0.5 \end{bmatrix}$ $\quad b_2 = 0.2$.

Ans.

$$z_1 = \begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 0.5 & -0.3 \\ 0.2 & 0.4 \end{bmatrix} + \begin{bmatrix} 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.9 & 0.5 \end{bmatrix} + \begin{bmatrix} 0 & 0.1 \end{bmatrix} = \begin{bmatrix} 0.9 & 0.6 \end{bmatrix}$$

$$z_2 = \begin{bmatrix} 0.9 & 0.6 \end{bmatrix} \begin{bmatrix} 1 \\ -0.5 \end{bmatrix} + 0.2$$

$$= 0.8$$

MSE = 0.12

Multi-output based Regression

$x = [1, 2]$ $\quad y = [3, 1]$

$$w_1 = \begin{bmatrix} 0.3 & 0.7 \\ -0.2 & 0.4 \end{bmatrix} \quad b_1 = \begin{bmatrix} 0 & -0.1 \end{bmatrix}$$

$$w_2 = \begin{bmatrix} 0.5 & -0.1 \\ 0.2 & 0.8 \end{bmatrix} \quad b_2 = \begin{bmatrix} 0.1 & -0.1 \\ -0.2 & \end{bmatrix}$$

Ans. $z_1 = x w_1 + b_1$

$$= \begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 0.3 & 0.7 \\ -0.2 & 0.4 \end{bmatrix} + \begin{bmatrix} 0 & -0.1 \end{bmatrix}$$

$$= \begin{bmatrix} -0.1 & 1.04 \end{bmatrix}$$

$AF_1 = \begin{bmatrix} -0.099 & 0.885 \end{bmatrix}$

$z_2 = A_1 w_2 + b_2$

$$= \begin{bmatrix} -0.099 & 0.885 \end{bmatrix} \begin{bmatrix} 0.5 & -0.1 \\ 0.2 & 0.8 \end{bmatrix} + \begin{bmatrix} 0.1 & -0.1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.1275 & 0.7179 \end{bmatrix} + \begin{bmatrix} 0.1 & -0.2 \end{bmatrix}$$

$$= \begin{bmatrix} 0.2275 & 0.5179 \end{bmatrix}$$

MSE = $\frac{1}{2}(3.959)$.

Example of regression

How to analyse time series data using FFNN?

Price forecasting, stock market

Eg: Gold Prediction

Mon →	y_{t-2}	1
Tue →	y_{t-1}	0.5
Wed →	y_t	0.15
Thu →	y_{t+1}	0.148
Fri →		0.1511
Sat →		

$x = [y_{t-1}, y_{t-2}] = [0.5, 1]$

$w_1 = [0.2, -0.1]$

$b_1 = 0$, $w_2 = 0.3$, $b_2 = 0$

AF = Sigmoid

Wed ⇒ $z_1 = x w_1^T + b_1$

$= [0.5 \quad 1] \begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix} + 0 = 0$

$A_1 = \partial(0) = 0.5$

$z_2 = A_1 w_2 + b = 0.5 \times 0.3 = 0.15$.

Thu ⇒ $z_1 = x w_1 + b$ $\quad x = [y_t, y_{t-1}]$.

$= [0.15 \quad 0.5] \begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix} = 0.03 - 0.01$

$= 0.02$

$A_1 = \partial(0.02) = 0.495$.

$z_2 = 0.495 \times 0.3 = 0.1485$

Fri ⇒ $z_1 = [0.148 \quad 0.15] \begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix} = 0.0296 - 0.015$

$= 0.0147$

$A_1 = \partial(0.0147) = 0.503$

$z_2 = 0.503 \times 0.3 = 0.1511$

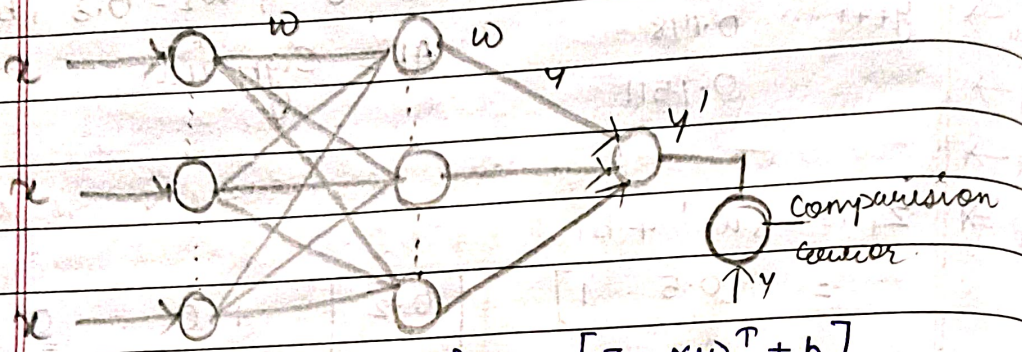
Outcome: For the next 3 days the output (gold price) varies is mostly fixed & a minor fluctuation is observed wrt 0.15.

4. Training FFNN

Types of learning in FFNN

i) Forward propagation (no training is possible).

ii) Backward propagation



Forward pass
- linear transformation $[z = xw^T + b]$.
- non linear tranformation [Activ^n Func^n].

⇒ Classification problem - Cross Entropy loss.

⇒ In regression problem - MSE/RMSE.

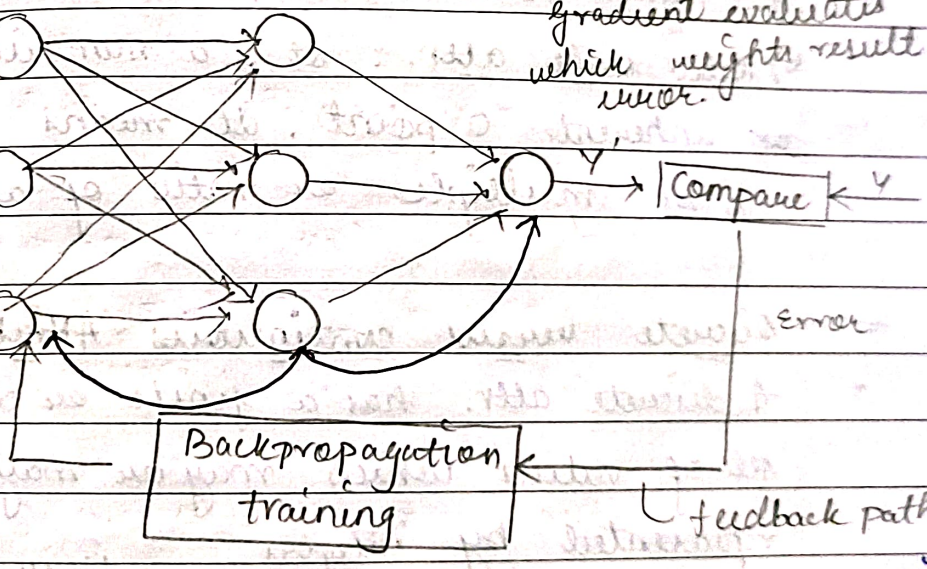
Loss = compare (true o/p, predicted o/p)

→ Loss occurs due to improper selec^n of weight & bias. To reduce it take derivatives wrt weights $(\partial L/\partial w)$

* Neural n/w is a n/w of interconnected neurons having weight, bias & activ^n func^n.
* Learning starts from the linear transform^n of i/p to non linear transform^n using Af.
* There are 2 types of learning
* Forward Propag^n: I/p are passed to hidden layer with some randomly initialized weights alongwith bias as linear transfor^n i.e. $z = xw + b$.

* To solve complex prob. non linear transfor^n is used through AF. The output of linear transfor^m is supplied to AF i.e. $A = f(Z)$
* Every layer applies these linear followed by non-linear transfor^n. Hence, o/p of applying AF in o/p layer is our final outcome.
* In forward prop^n, as i/p travels from i/p through hidden to o/p layer, no training is possible.

Backward Propagation Neural Network (BPNN)



* BP means propag^n of error in the backward direc^n in NN.
* Its objective is to reduce the diff b/w actual & predicted o/p by adjusting weight & bias.
* It operates iteratively to adjust weight & bias.
* In each EPOCH the models adapts these parameters by reducing loss following error gradient.

* BPNN in two phases :

① Forward Pass - I/p travels from input through hidden to o/p layer following linear & non linear transformation.

② Backward pass - error wrt. predicted & actual output is evaluated & it propagates back through the n/w to adjust weight & bias.
→ It continues layer by layer ensuring that the n/w learns & improves performance.

When to stop learning ?

① Define threshold

② Set no. of epochs

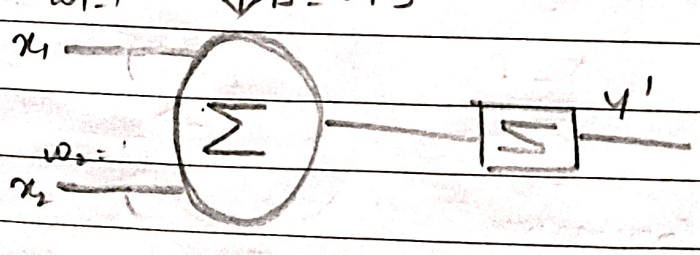
Q) Construct a perceptron which act as an AND gate for the given straight line $x_1 + x_2 - 1.5 = 0$.

Ans

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad W = [1, 1] \quad b = -1.5$$

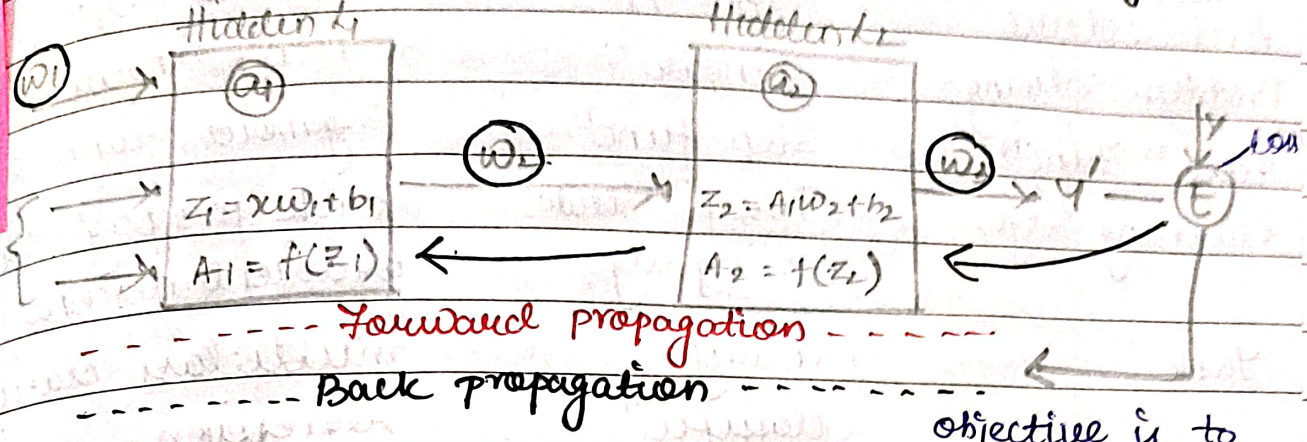
$$Y = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$



$$Af = \text{Step func}^n = f(z) = \begin{cases} 1, & z > 0 \\ 0, & z < 1 \end{cases}$$

$$A_1 = f(z) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Working principle of backpropagation of algo.



$\frac{\partial E}{\partial w}$ → gradient func^n.

objective is to reduce error & select optimal weights.

E = actual o/p − predicted o/p.

$$\frac{\partial a_1}{\partial w_1} \times \frac{\partial a_2}{\partial a_1} \times \frac{\partial Y'}{a_2} \times \frac{\partial E}{\partial Y'} = \frac{\partial E}{\partial w_1}.$$

* A set of i/p neurons with input x connected to neurons of next layer having certain weights are multiplied & passed to the activ^n function giving a certain o/p.
* E is calculated keeping in mind that the actual o/p is calculated using derivative of cost func^n.
* Basically back propag^n is to update the weights for reducing loss in the next iter^n.
* We need to have derivative of cost func^n i.e. partial derivative of cost func^n wrt weight. follows a chain rule i.e.
* Aggregating all derivatives the change of 'w' is evaluated.

Criteria	SLP	MLP
Architecture	I/P + O/P	I/P + HL + O/P
Problem Solving	Linear	Non - linear
Activ^{n} Func^{n}	Step func^{n}	sigmoid, tanh, Relu
Learning algo.	Percep^{n} learn - ning algo.	Back propag^{n} + gradient descent.
Task	Binary classific^{n}	multiclass classific^{n}, regression
Types of app^{n}	Simple	Complex
N/w complexity	simple	complex.

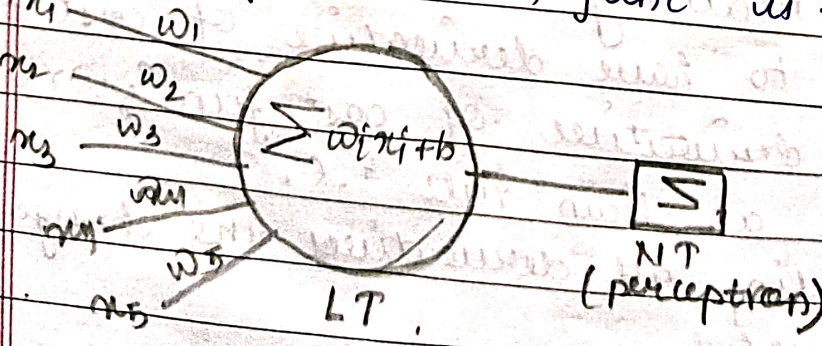
Perceptron

→ When neurons are connected through a network it called perceptron.

→ They have diff n/w parameters - weight, bias, activa^{n} func^{n}.

→

→ Output of activation func^{n} is the predicted o/p.



Types of perceptron

① Single layer perceptron (No Hidden layer)
② Multilayer perceptron (1 or more hidden layer)

Perceptron Training Rule

→ It is an algo. used to train perceptron.
→ It adjust the weights assigned to the i/p features based on errors made in predic^n.
→ This allow the perceptron to improve its ability to distinguish b/w two classes.

① Initialization: Start initializing (s) w(b) to a very small random value.

② Set learning rate: Choose a learning rate that controls the magnitude of weight updates. It is set to very small value like 0.01, 0.001...

③ Training process:
i) present each training example at a time.
ii) For each i/p vector x calculate the weighted sum. $(z = xw + b)$ — ①
iii) Apply AF, typically a step func^n to determine the predicted o/p.

$$y_{pred} = \begin{cases} 0, & \text{if } z < 0 \\ 1, & \text{if } z \geq 0 \end{cases} \quad \text{— ②}$$

iv) Calculate error by comparing predicted o/p with actual o/p.

$$\text{Error} = y - y_{pred} \quad \text{— ③}$$

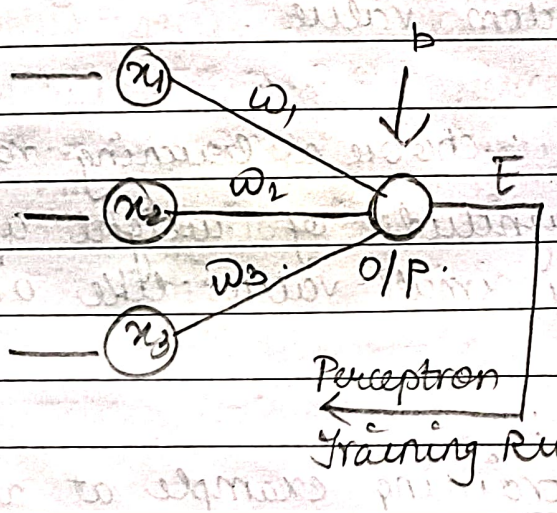
v) If there is error, then update weight
i.e. $y_{pred} \neq y$

$$w \leftarrow w + \eta * Error * x \quad — (4)$$

$$b \leftarrow b + \eta * Error \quad — (5)$$

This adjustment moves the decision boundary in a direction that reduces the classification error for the current sample.

(4) Repeat the training process EPOCHS until (1) the perceptron correctly classifies all training example or (2) until a max. EPOCHS is reached.



Q) $x = \begin{bmatrix} 2 & 1 \\ 1 & -1 \\ -1 & -2 \end{bmatrix}$ $t = \begin{bmatrix} +1 \\ -1 \\ -1 \end{bmatrix}$ $w = [0, 0]$, $b = 0$, $\eta = 1$

$$y = \text{sign}(z) = \begin{cases} 1, & z > 0 \\ 0, & z = 0 \\ -1, & z < 0 \end{cases}$$

Ans. $z = xw + b = \begin{bmatrix} 2 & 1 \\ 1 & -1 \\ -1 & -2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} + [0] = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

Consider 1st sample

$x = [2, 1]$, $w = [0, 0]$, $b = [0]$, $\eta = 1$, $t = +1$

$z = xw^T + b = 2 \times 0 + 0 \times 1 + 0 = 0$.

$y = \text{sign}(z) = 0$.

Error $= 1 - 0 = 1$.

$w_1' = 0 + 1 \cdot 1 \times 2 = 2$; $w_1' = 0 + 1 \cdot 1 \times 1 = 1$.

$w_1' = [2\ 1]$, $b' = b + \eta * E = 1$.

Consider 2nd sample.

$x = [1\ -1]$, $w_1' = [2\ 1]$, $b_1 = 1$, $t = -1$

$z = xw^T + b = [1\ -1] \begin{bmatrix} 2 \\ 1 \end{bmatrix} + [1]$.

$= [1] + 1 = 2$.

$y = \text{sign}(z) = +1$

$e = t - y = -1 - 1 = -2$.

$w_2' = 2 + 1 \times -2 \times 1 = [0\ 3]$.

$= 1 + 1 * -2 \times -1$

$b = 1 + 1 \times (-2) = -1$.

Consider 3rd sample

$x = [-1, -2]$, $w = [0, 3]$, $b = 1$, $t = -1$, $\eta = 1$.

$z = -7$; $y = -1$.

$E = -1 + 1 = 0$ (no update)

$w = [0, 3]$; $b = -1$

Now verify for all samples

Perception learning for AND/OR Gate

Q) Apply the perceptron training rule to find optimal weights. Given:
$w_1 = 1.2$, $w_2 = 0.6$, $\eta = 0.5$, threshold = 1.

AND

x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1

$z = wx$

$y = 0$

$E = t - y$ < 0 - no update, >0 - update wt.

$w \leftarrow w + \eta E x$.

Ans. For sample 1:

$x = [0, 0]$; $w = [1.2, 0.6]$; $\eta = 0.5$

$\theta = 1$; $t = 0$;

Iter^n 1: $z = wx = 0$.

$y = z < 1 = 0$; $E = 0 - 0 = 0$

No weight update.

For sample 2:

$x = [0\ 1]$; $w = [1.2, 0.6]$

$z = [0\ 1] * \begin{bmatrix} 1.2 \\ 0.6 \end{bmatrix} = 0.6$

$y = z < 1 = 0$; $E = 0 - 0 = 0$.

No weight update.

For sample 3:

$x = [1\ 0] * \begin{bmatrix} 1.2 \\ 0.6 \end{bmatrix} = 1.2$.

$y = 1$; $E = 0 - 1 = -1$

$w_1' = 1.2 + (0.5 \times (-1) \times 1) = 0.7$

~~$= 0.6 + 0.5 = 1.1$~~ 0.6

classmate
Date
Page

Iter 2:

For sample 1:

$x = [0, 0]$, $w = [0.7, 0.6]$.

$z = wx = [0.7 \ 0.6] \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0$

$E = 0$

No update.

For sample 2:

$x = [0, 1]$

$z = [0 \ 1] \begin{bmatrix} 0.7 \\ 0.6 \end{bmatrix} = 0.6$

$\Rightarrow y = 0$; $E = 0$

No update.

For sample 3

$x = [1 \ 0]$; $w = [0.7 \ 0.6]$

$z = xw = [1 \ 0] \begin{bmatrix} 0.7 \\ 0.6 \end{bmatrix} = 0.7$

$y = z < 1 = 0$. ; $E = 0$

No update.

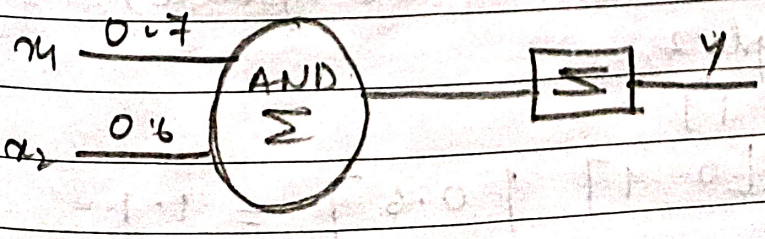
For sample 4

$x = [1 \ 1]$

$z = [1 \ 1] \begin{bmatrix} 0.7 \\ 0.6 \end{bmatrix} = 0.7 + 0.6$
$= 1.3$.

$y = z > 1 = 1$. $E = 0$

No update.



classmate
Date
Page

Q) Apply perceptron training rule over OR gate to find optimal weights.

$w_1 = 0.6$; $w_2 = 0.6$, thresh = 1 ; $\eta = 0.5$

OR

x_1	x_2	t
0	0	0
0	1	1
1	0	1
1	1	1

For sample 1:

$x = [0, 0]$; $w = [0.6 \quad 0.6]$

$z = xw = [0 \quad 0] \begin{bmatrix} 0.6 \\ 0.6 \end{bmatrix} = 0$

$y = z < 0 = 0$. (No update)

For sample 2:

$x = [0 \quad 1]$

$z = [0 \quad 1] \begin{bmatrix} 0.6 \\ 0.6 \end{bmatrix} = 0.6$

$y = z < 0 = 0$; $E = 1 - 0 = 1$.

$w = 0.6 + 0 = 0.6$

$0.6 + 0.5 \times 1 \times 1 = 1.1$.

New weight $= [0.6 \quad 1.1]$.

Iter 2:

$x = [0 \quad 0]$

$z = 0$ $E = 0$ (No update)

For sample 2

$x = [0 \quad 1]$

$z = [0 \quad 1] \begin{bmatrix} 0.6 \\ 1.1 \end{bmatrix} = 1.1$

$y = 1$; $E = 0$ (No update)

For sample 3:

$x = [1 \quad 0]$

$z = [1 \quad 0] \begin{bmatrix} 0.6 \\ 1.1 \end{bmatrix} = 0.6$

$y = z < 1 = 0$; $E = 1 - 0 = 1$.

$w = 0.6 + 1 \times 0.5 \times 1 = 1.1$

$= 1.1 + 0 = 1.1$

New weight $[1.1 \quad 1.1]$

Iteration 3: $x = [0 \quad 0]$

$z = 0$, $y = 0$, $E = 0$ (No update)

For sample 2

$x = [0 \quad 1]$

$z = 1.1$; $y = 1$; $E = 0$

For sample 3

$x = [1 \quad 0]$

$z = 1.1$; $y = 1$; $E = 0$

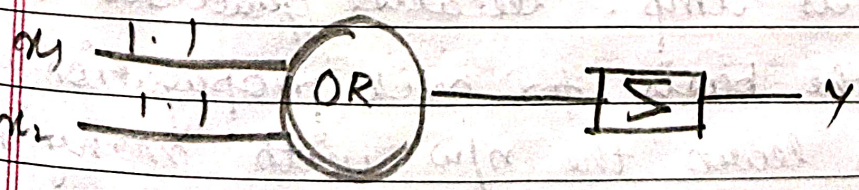
For sample 4

$x = [1 \quad 1]$

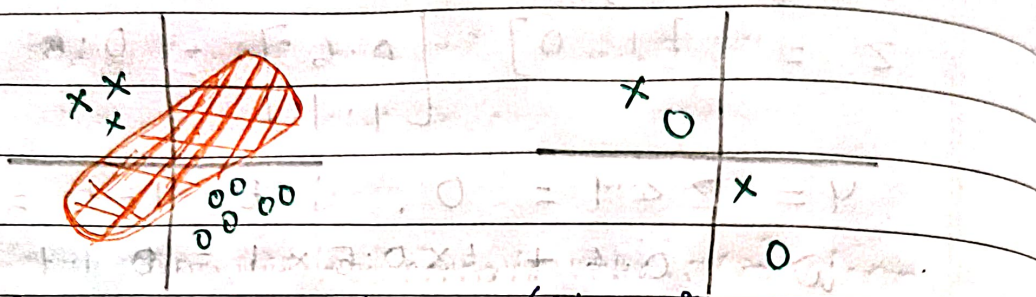
$z = 1.1 + 1.1 = 2.2$

$y = z > 1 = 1$.

$E = 0$ (No update).

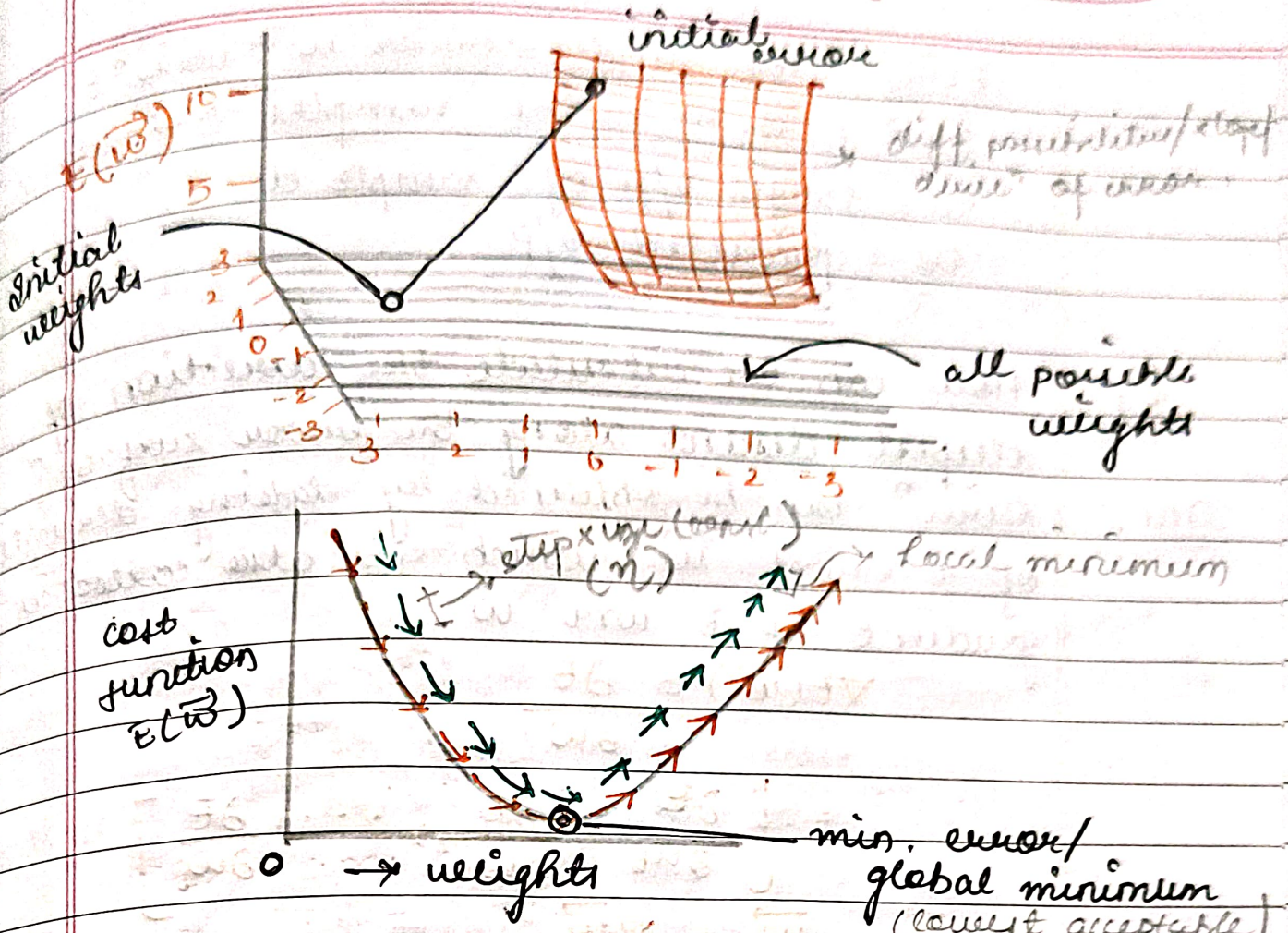


gradient descent & delta rule



(linearly separable problem) (Non linearly separable problem)

* Perceptron training rule finds a successful weight vector when training examples are linearly separable, but it fails to converge if examples are non linearly separable.
* A 2nd rule called as delta rule is designed to overcome this issue.
* If the training examples are not linearly separable, delta rule converts towards a best fit approx to a target o/p.
* The key idea behind delta rule is to use gradient descent to search the hypothetical space of all possible weight vector, to find the weights that best fit the training example.
* This rule is imp. because gradient descent provides the basis for back propagation algo, which can learn the n/w with many inter-connected units.



How delta rule works?

* The delta training rule is best understood for considering the task of training an unthresholded perceptron which is a linear unit.

Here, o/p 'o' is defined as:

$$o = w_0x_0 + w_1x_1 + w_2x_2 + \dots\dots$$

$$\Rightarrow \boxed{o(\vec{w}) = \vec{x} \cdot \vec{w}}$$

* In order to derive a wt. learning rule for linear learning unit, let us begin by specifying a measure for training error.
* There are many ways to identify this error, one such way is considering MSE.

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$E(\vec{w})$ = cost func^n = error wrt weight.
$d \in D$ = $d \Rightarrow$ no. of samples $D \rightarrow$ dataset
t_d = target o/p of sample d
o_d = predicted o/p.

Q) How can we calculate the direction of steepest descent along the error surface?

Ans. Direc^n can be obtained by taking derivation of 'E' wrt $\vec{w}$, which is also called a gradient of E wrt w.

$$\nabla E(\vec{w}) = \frac{\partial E}{\partial w}$$

$$= \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_i} \right]$$

$$\vec{w}_n = \vec{w}_0 + \nabla \vec{w}$$

$$\nabla w = -\eta \frac{\partial E}{\partial w}$$

$\eta \rightarrow$ learning rate a +ve constraint which determines step size in gradient descent search.

* The -ve sign is present because we want to move the $\vec{w}$ in direc^n that decreases E.

$$\frac{\partial E}{\partial w_i} = \frac{1}{2} \frac{\partial}{\partial w_i} \sum_{d \in D} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$= \frac{1}{2} \times \sum (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= (t_d - o_d) \frac{\partial}{\partial w_i} (t_d) - \frac{\partial}{\partial w_i} (o_d)$$

Step-1:
Step-2:
Step-3:
Step-4:

Q).

Ans.

$= (td - od) - \frac{\partial}{\partial w_i} (od) \rightarrow \frac{\partial}{\partial w_i}(w_1 x_1 + w_2 x_2 + w_i x_i)$

$= -(td - od)\ x_i$

$\therefore \nabla \vec{w} = +\eta [+ (td - od) x_i]$

$= \eta (td - od) x_i$

$w = w + \nabla w$

$= w + \eta (td - od) x_1$

Delta rule over linear perceptron

Step-1: $y = wx$ (Apply forward pass).

Step-2: Find out error

$E = t - y$, t = actual class.

Step-3: If $E \neq 0$, then apply delta rule

$\nabla w = \eta (t - y) x$.

Step-4: Update weights.

$w_n = w_0 + \nabla w$.

Continue till all samples have $E = 0$.

Q). $x = 0.5$, $t = 0.8$, $w = 0.4$, $\eta = 0.1$. Apply delta rule & find out E for 2 iterat^n, update weights. & show the trend.

Ans. $y = wx = 0.5 \times 0.4 = 0.2$

$E = t - y = 0.8 - 0.2 = 0.6$

$\nabla w = 0.1 \times 0.6 \times 0.5$

$= 0.03$

$w_n = 0.4 + 0.03 = 0.43$.

Step^n 2: $y = 0.43 \times 0.5 = 0.215$

$E = 0.8 - 0.215 = 0.585$

$\nabla w = 0.1 \times 0.585 \times 0.5 = 0.029$

$w = 0.43 + 0.029 = 0.459$

Iterd It	Error	Weight
1.	0.6	0.43
2.	0.585	0.459

Q). $x_1 = 0.6$; $x_2 = 0.8$; $w_1 = 0.2$; $w_2 = 0.4$; $t = 0.75$; $b = 0.1$; $\eta = 0.1$. apply delta rule over the given linear perceptron, find weight & bias update for 2 iterations

Ans. $y = xw + b$

$$= [0.6 \quad 0.3] \begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix} + 0.1$$

$$= 0.24 + 0.1 = 0.34$$

$E = t - y = 0.41$

$\nabla w = 0.1 \times 0.41 \times 0.6 = 0.024$

$0.1 \times 0.41 \times 0.3 = 0.012$

$w_n = [0.224 \quad 0.412]$ $\nabla b = \eta E = 0.041$; $b = 0.141$

Iter^n 2: $y = [0.6 \quad 0.3] \begin{bmatrix} 0.224 \\ 0.412 \end{bmatrix} + 0.1$

$= 0.258 + 0.141 = 0.399$

$E = 0.75 - 0.399 = 0.351$

$\nabla w = 0.1 \times 0.351 \times 0.6 = 0.021$

$0.1 \times 0.351 \times 0.3 = 0.010$

$w_n = [0.245 \quad 0.422]$

$\nabla b = \eta E = 0.1 \times 0.1 = 0.0351$

$b_n = 0.141 + 0.0351 = 0.176$

If non-linear definitely Hidden Layer

Deviation of learning rule for a perceptron with sigmoid activation function

Step 1: Initialize weights & bias.

Step 2: Apply forward pass & calculate 'z'.

$$z = \sum_{i=1}^{n} x_i w_i \quad \text{OR} \quad \sum_{i=1}^{n} x_i w_i + b$$

Step 3: Apply activation func^n sigmoid

$$y' = \frac{1}{1+e^{-z}}$$

Step 4: If $y' \neq y$, apply delta rule & update wt.

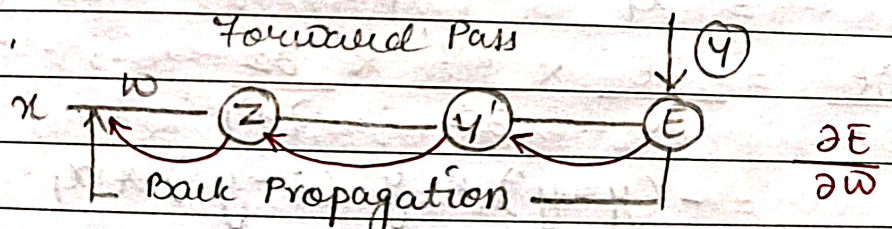
$$w_i = w_i - \eta \frac{\partial E}{\partial w_i}$$

$y' \rightarrow$ predicted o/p ; $y \Rightarrow$ actual o/p

$\eta \rightarrow$ learning rate ; $\frac{\partial E}{\partial w_i} \rightarrow$ gradient of error wrt weight.

Step 5: Repeat step 2 to step 4 until loss func^n converges or stopping criteria reached.

$$E(\vec{w}) = \frac{1}{2}(y-y')^2$$



$$\frac{\partial E}{\partial w}$$

$$\frac{\partial z}{\partial w} \times \frac{\partial y'}{\partial z} \times \frac{\partial E}{\partial y'} = \frac{\partial E}{\partial w}$$

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y'} \times \frac{\partial y'}{\partial z} \times \frac{\partial z}{\partial w}$$

$$\frac{\partial E}{\partial y'} = \frac{\partial}{\partial y}\left[\frac{1}{2}(y-y')^2\right]$$

$$= \frac{1}{2} \times 2(y-y')\frac{\partial}{\partial y'}(y-y')$$

$= (y - y') \times -1$

$= -(y - y')$

$\frac{\partial y'}{\partial z} = \frac{\partial}{\partial z} (1 + e^{-z})^{-1}$

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

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

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

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

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

$y' = (1 + e^{-z})^{-1}$

$1 - y' = 1 - \frac{1}{1 + e^{-z}}$

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

$\therefore \frac{\partial y'}{\partial z} = y(1 - y')$

$\frac{\partial z}{\partial w} = \frac{\partial}{\partial w_1} [w_0 x_0 + w_1 x_1 + w_2 x_2 + \dots]$ ($x_0 = 1$, $w_0 = 0$, $w_2 = 0$)

$= x_1$

$\frac{\partial E}{\partial y} = \frac{\partial E}{\partial y} \times \frac{\partial y'}{\partial z} \times \frac{\partial z}{\partial w_1}$

$= -(y - y') * y'(1 - y') * x_1$

$w_1 = w_1 - \eta \frac{\partial E}{\partial w_1}$

$= w_1 - \eta [-(y - y') y'(1 - y') x_1]$

$= w_1 + \eta (y - y') y'(1 - y') x_1$

Linear	Non-linear
$w = w + \eta (t - o) x$	$w = w + \eta (y - y') y'(1 - y') x_1$
✗	✓
linearly separable	non linearly separable.

AP
Hidden prob

Q). $x_1 = 0.6$, $w_1 = 0.8$, $b = 0.2$, $t = 1$, $\eta = 0.1$
AF = sigmoid. Apply delta rule & update weight for 2 iterations.

Ans. $z = xw + b$

$= (0.6 \times 0.8) + 0.2 = 0.68$

$y' = \text{sigmoid}(z) = \frac{1}{1+e^{-z}} = 0.663$

$E = t - y' = 1 - 0.663 = 0.337$

$\nabla w = \eta (y - y') y' (1 - y') x$

$= 0.008$

$\nabla b = \eta (y - y') y' (1 - y') = 0.007$

$w_n = w_0 + \nabla w = 0.868$

$b_n = b_0 + \nabla b = 0.207$

Iter^n 2: $z = 0.6 \times 0.868 + 0.207 = 0.6902$

$y' = \frac{1}{1+e^{-z}} = 0.6659$

$E = 1 - 0.674 = 0.3341$

$\nabla w = 0.0044$

$\nabla b = 0.0074$

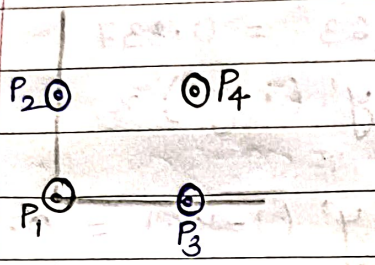
$w_n = 0.8089$

$b_n = 0.2149$

Perceptron training for XOR gate

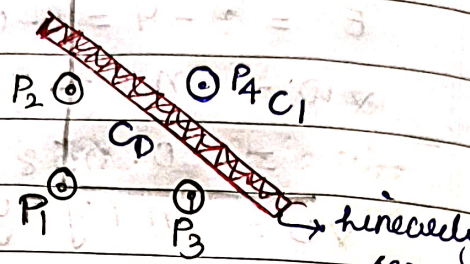
XOR

x_1	x_2	y	
0	0	0	C_0
0	1	1	C_1
1	0	1	C_1
1	1	0	C_0



AND

x_1	x_2	y	
0	0	0	C_0
0	1	0	C_0
1	0	0	C_0
1	1	1	C_1



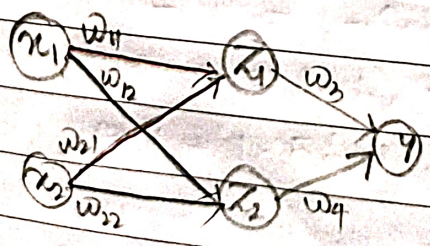
* If we want to draw a straight line w.r.t XOR gate in 2D plane, to differentiate the data objects, we cannot draw.
* This is a different type of logical gate which is nonlinearly separable.
* So, we cannot apply simple perceptron rule to find or implement XOR gate.
* Hence, we will use a different eqⁿ which represent a non-linear solⁿ to XOR gate.

$$y = x_1\bar{x}_2 + \bar{x}_1 x_2$$

$z_1 = x_1\bar{x}_2$ — 1st function

$z_2 = x_2\bar{x}_1$ — 2nd function

$y = z_1$ OR z_2



$z_{11} = xw$

$y' = f(z_{11})$

$E = y - y'$

If $\sum E \neq 0$, update w

$w = w + \Delta\nabla w = w + \eta E^x$

Iter^n 1: S'

Iter^n 2:

$w_1 = [w_{11}, w_{21}] = [1, 1]$

$w_2 = [w_{12}, w_{22}] = [1, 1]$ $\quad \eta = 1.5$

1st function

x_1	x_2	z_1
0	0	0
0	1	0
1	0	1
1	1	0

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

θ = threshold = 1

$z_1 = x_1 \bar{x_2}$

Iter^n 1: S^1: $x_1 = [0, 0]$, $y = 0$

$$z_{11} = [0\ 0] \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0$$

$f(z_{11}) = 0$, no update.

S^2: $x_2 = [0, 1]$, $w = [1, 1]$, $\eta = 1.5$, $y = 0$

$$z_{11} = [0\ 1] \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1.$$

$E = 0 - 1 = -1$; $w = 1 + 1.5 \times (-1) \times 0 = 1$

$= 1 + 1.5 \times (-1) \times 1 = -0.5$

Iter^n 2: S^1: $x = [0, 0]$, $y = 0$

$$z = [0\ 0] \begin{bmatrix} 1 \\ -0.5 \end{bmatrix} = 0$$

$E = 0$, no update.

S^2: $x = [0, 1]$

$$z = [0\ 1] \begin{bmatrix} 1 \\ -0.5 \end{bmatrix} = -0.5$$

$f(z) = 0$; $E = 0$, no update

S^3: $x = [1\ 0]$

$$z_{11} = [1\ 0] \begin{bmatrix} 1 \\ -0.5 \end{bmatrix} = 1.$$

$E = 0$, no update.

s^4: $x = [1\ 1]$

$z = [1\ 1] \begin{bmatrix} 1 \\ -0.5 \end{bmatrix} = 0.5 = 0$

$E = 0 - 0 = 0$,

$w = 1 + 1.5(-1)1 \neq -0.5$

$-0.5 + 1.5(-1)1$

Function 2

$z = \bar{x}_1 x_2$

x_1	x_2	z_2
0	0	0
0	1	1
1	0	0
1	1	0

$w_2 = [w_{12}, w_{22}]$

Iteration 1: $s^1: x = [0\ 0]$

$z = [0\ 0] \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0$

$E = 0$, no update

$s^2: x = [0, 1]$

$z = [0\ 1] \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1$

$E = 0$, no update

$s^3: x = [1, 0]$

$z = [1\ 0] \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1$

$E = 0 - 1 = -1$

$w = 1 + 1.5(-1)1 = -0.5$

$1 + 1.5(-1)0 = 1$

Itern 2: s^1: $x = [0, 0]$

$z = [0\ 0] \begin{bmatrix} -0.5 \\ 1 \end{bmatrix} = 0$

$E = 0 - 0 = 0$, no update.

s^2: $x = [0\ 1]$

$z = [0\ 1] \begin{bmatrix} -0.5 \\ 1 \end{bmatrix} = 1.$

$E = 1 - 1 = 0$, no update

s^3: $x = [1\ 0]$

$z = [1\ 0] \begin{bmatrix} -0.5 \\ 1 \end{bmatrix} = -0.5$

$f(z) = 0.$

$E = 0 - 0 = 0$, no update.

s^4: $x = [1\ 1]$

$z = [1\ 1] \begin{bmatrix} -0.5 \\ 1 \end{bmatrix} = 0.5$

$f(z) = 0.$

$E = 0 - 0 = 0$, no update.

3rd function

x_1	x_2	y
0	0	0
0	1	1
1	0	1
0	0	0

Itern 1: $s^1 = [0\ 0] \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0$

$E = 0$, no update.

S^2: $x = [0\ 1]$

$z = [0\ 1] \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1.$

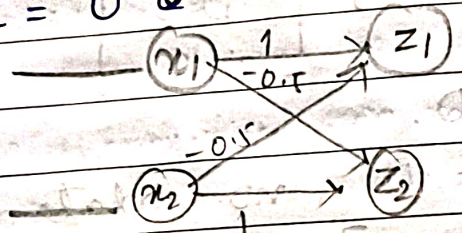
$E = 1 - 1 = 0$, no update.

S^3: $x = [1\ 0] \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1.$

$E = 1 - 1 = 0$, no update.

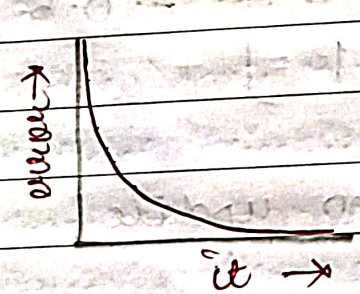
S^4: $[0\ 0] \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0.$

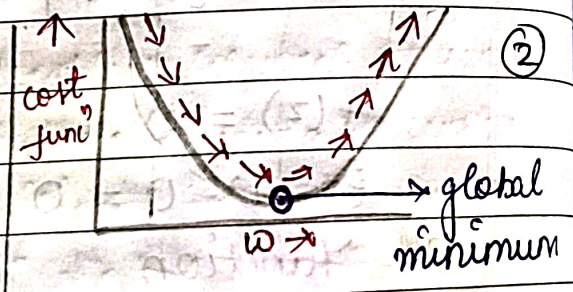
$E = 0 - 0 = +0$, no update



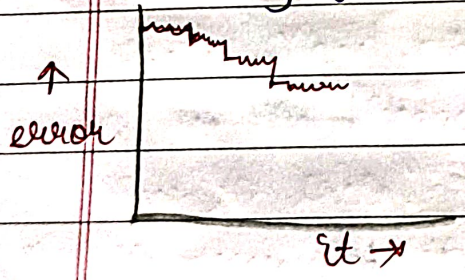
Problem with gradient descent

① Ideal curve

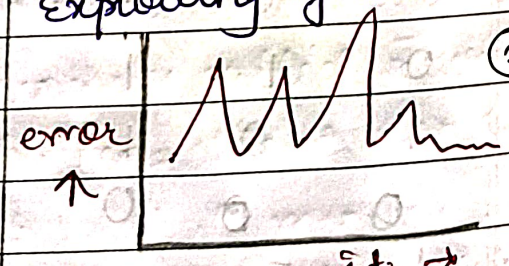


② 

③ Vanishing gradient



Exploding gradient ④



Graph b/w cost func" & weight demonstrate (fig-2) how gradient descent achieves its aim of reaching global cost minima point.

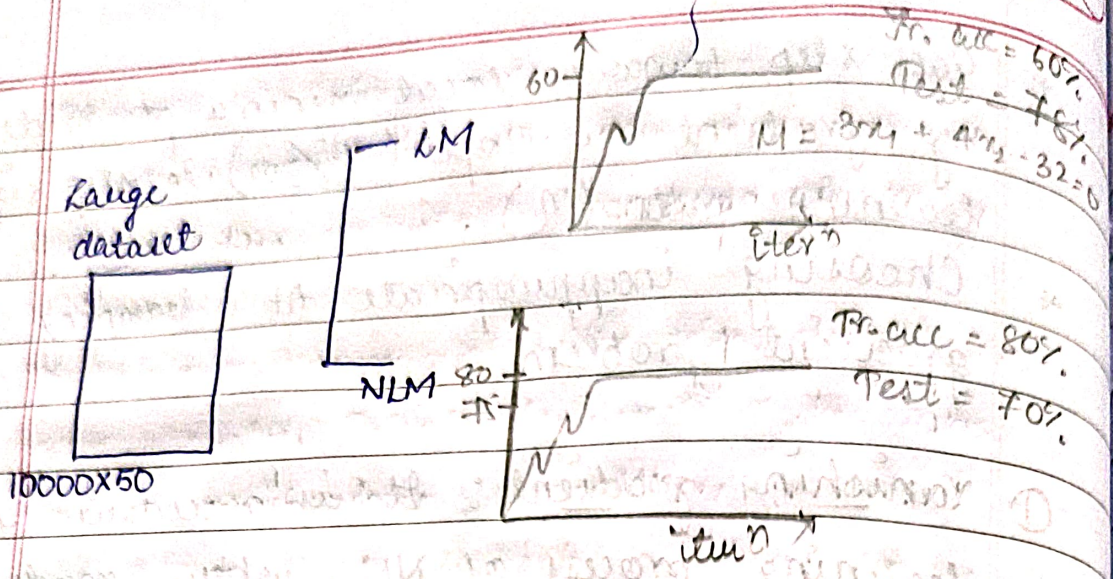
* Each step towards local minima pt is determined by gradient or slope ($\partial E/\partial w$) & step size i.e. learning rate (η).
* Choosing inappropriate AF, learning leads to gradient problem.

① Vanishing gradient: It is a scenario in the learning process of NN, where model doesn't learn at all.
→ This is due to, when gradient becomes too small, almost vanishes leads weight getting stuck, never reaching the optimal value. This n/w is not able to learn & converge.

② Exploding Gradient: Is exactly opposite to vanishing gradient, where model keeps on learning, weight keeps updating but model never converge.
→ It computes gradient wrt weight which becomes extremely large in the earlier layer of NN in such a way that it explodes or oscillates.

Overfitting

Dataset
- Small dataset. (32×50)
- Large dataset.
  - 50000×85
  - 85×50000



* given with a bunch of data points on a flat plane, our goal is to find a curve, that best describes the data.
* let us consider two models to train these data points:
  ① linear model ② Non linear model
* By building a more complex model it's quite easy to perfectly fit our training dataset because we are giving enough freedom to our model to control itself to fit the observe" in to
  But when we evaluate such complex model over new dataset, model fails. This phenomena is called overfitting & is the biggest challenge that ML & DL model fails.
* To overcome this issue, the entire dataset is not used for training, rather split into training, testing & validation set.

Date
Page

Data Split
25 samples

Training Set
(20 samples)

Test Set
(5 samples)

Training set
16

Validation set
4

Case 1 (✓)

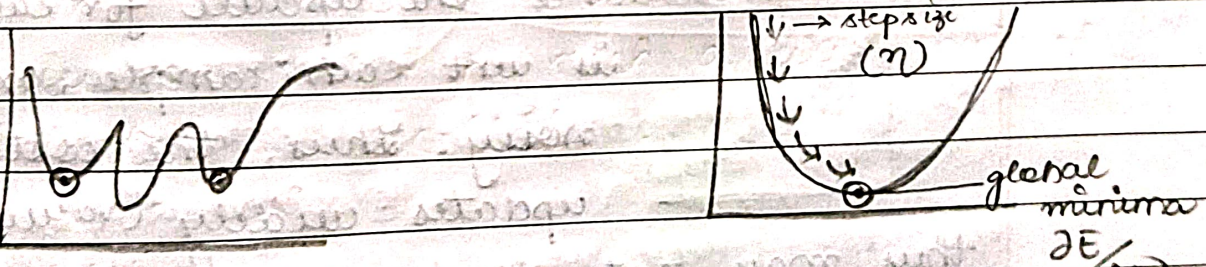
	tr	V
B1	98	96
B2	96	95
B3	93	93
B4	86	95

Case 2 (Overfitting)

	tr	V
B1	98	96
B2	98	95
B3	98	93
B4	98	95.

Challenges with gradient descent

* Local minima & Saddle Point :



→ The point where gradient is 0 & cost is lower than its intermediate neighbour, higher than global minima.

→ Point where gradient is 0, but func" increases in some direc" & decreases in some other. Thus, algo slow down significantly around these points.

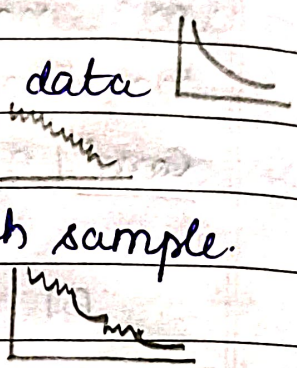
② Choosing learning rate

* To large - overshoot: The algo may overshoot the minima, fail to converge or may diverge.
* To less - slow convergent: The algo. will converge very slowly. Potentially will take long time to reach min. point.

③ Computational time

* Gradient descent (GD)

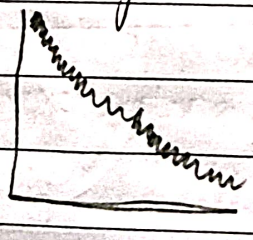
i) Batch gradient descent – entire data
ii) Stochastic GD – single sample
iii) Minibatch GD – a small batch sample.



* Batch GD is computationally very expensive.
* Storing the entire data in the memory & computing avg. gradient is infeasible.

④ Variance stochastic GD & Minibatch GD.

(i) SGD - High variance The update for each step is wrt each sample, maybe noisy. Thus, the cost func" updates widely (variance) & thus may not reach global minima.



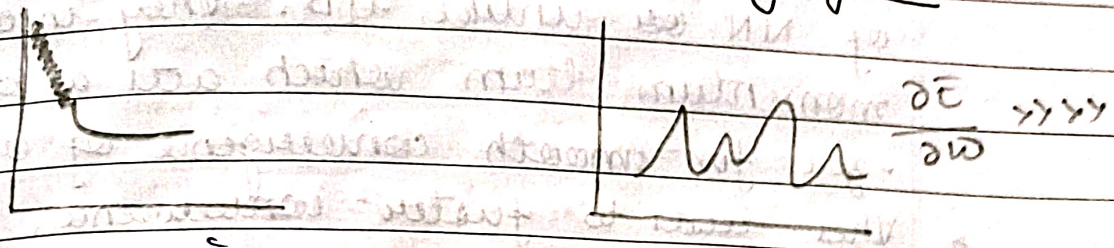
(ii) Selection of batch size in minibatch.

* One extra parameter i.e. batch size is added for efficient learning.

⑤ Scaling issue - feature value - [0-1]

* If the feature value are scaled within a large interval, GD may follow a zig-zag pattern for convergence.

⑥ Vanishing gradient & exploding gradient



* When gradient becomes extremely small i.e. close to 0 in earlier layers, making the weights in those layer learn very slowly or stop learning. In this scenario, model may not reach the global min. point.

* Gradient becomes extremely large, leading to unstable n/w behaviour. Thus, leads to large weight updates & prevents convergence.

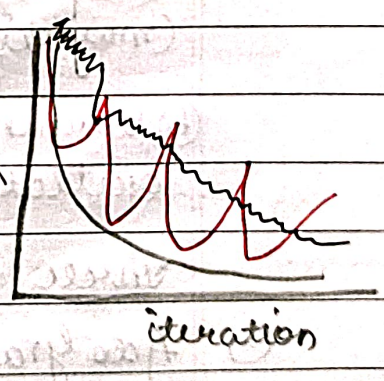
Solution to gradient ascent

* Momentum based optimization

$$v_{t+1} = \beta v_t + (1-\beta)\frac{\partial E}{\partial w_t}$$

$$w_{t+1} = w_t - \eta v_{t+1}$$

$$w_{t+1} = w_t - \eta \frac{\partial E}{\partial w}$$

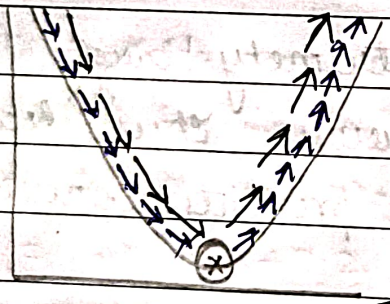


* Here, v_t is the velocity i.e. running avg. of GD.

* β = momentum factor ranges from 0 to 1.
η = learning rate
$\frac{\partial E}{\partial w}$ = Gradient of loss func^n

* Momentum base GD optimizer are advanced techniques used to enhance the training of NN ~~as~~ unlike GD. They incorporate momentum term which acts as an optimizer for smooth convergence of error surface.
* This leads to faster convergence, reduced oscillation & improved performance particularly while training a large dataset.

Impact of learning rate (η) over gradient



① Low value.
② High value.

ⓐ convergence is slow
ⓑ more epoch
ⓒ computat^n cost is high

ⓐ Overshooting (may skip global minima)
ⓑ learning curve may oscillate

three ways to update weight
① Adagrad (Adaptive gradient algo.)
② RMSprop (Root mean square propagation)
③ ADAM (Adaptive moment optimizer).

① Adagrad

* It adapts learning rate per parameter based on squared gradient.

$$lr_t = \frac{\eta}{(G_t + \epsilon)^{1/2}}$$

$\eta \rightarrow$ const

$G_t \rightarrow$ sum of squared grad.

$\epsilon \rightarrow$ small const. to avoid divided by 0 error.

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

$$\theta_{t+1} = \theta_t - lr_t \nabla\theta$$

θ_t: current parameter; $\nabla\theta$: gradient of cost func^n wrt θ.

→ Sparse data (d2)

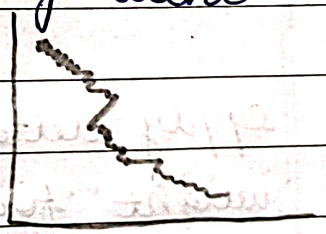
→ Feature have diff. importance

disadv → change of η is too small. May decay [$0 - 10^5$]

② RMSProp

→ learning rate is updated considering running of sum of average squared gradient

$$= \left(\left(\frac{\partial E}{\partial w_1}\right)^2 + \left(\frac{\partial E}{\partial w_2}\right)^2 \cdots\right)^{1/2}$$



→ SGD & Adagrad:

ⓐ gradient at step t

$g_t = \nabla\theta$ (grad. of cost func^n wrt θ)

ⓑ update moving avg of gradient.

$E[g^2]_t = \gamma(E[g^2]_{t-1} + (-\gamma)$

γ = decay rate = considering [0-1], mostly is 0.9

ⓒ update parameter

$$\theta_{t+1} = \theta_t - \frac{\eta}{(E[g^2]_t + \epsilon)^{1/2}}$$

$\eta = 0.0001$

$\gamma = 0.9$

③ Adam

* Computationally efficient
* high performance (in terms of error) & speed up convergence.
* RMSProp + momentum concept (technique).

ⓐ First moment estimate

$$m_t = \beta_1 * m_{t-1} + (1-\beta_1) g_t \quad [1^{st} \text{ moment vector}]$$

ⓑ 2nd moment estimate

$$v_t = \beta_2 + m_{t-1} + (1-\beta_2) g_t^2 \quad [2^{nd} \text{ moment vector}]$$

ⓒ
$$\hat{m}_t = m_t / (1-\beta_1^t)$$
$$\hat{v}_t = m_t / (1-\beta_2 t)$$

ⓓ weight update

$$w_t = w_{t-1} - \alpha * \frac{\hat{m}_t}{(\sqrt{v_t} + \varepsilon)}$$

m_t & v_t = 1st & 2nd moment vector.

$\alpha = 0.0001$; $\beta_1 = 0.9$; $\beta_2 = 0.999$; $\varepsilon = 10^{-8}$

Q) Apply delta rule & gradient descent to update weight for 2nd iteration. Given x/w parameters are

$x = 2$; $y = 0.8$; $w = 0.5$; $b = 0.2$; $\eta = 0.1$

Ans.

$z = 2 \times 0.5 + 0.2$

$= 1.2$

sigmoid = 0.768

$E = 0.0314$

$\nabla w = 0.00114$

$\nabla b = 0.000057$

$w = 0.50114$

$b = 0.200057$

Steps

① $z = xw + b$

② $y' = \text{sigmoid}(z)$

③ $E = y - y'$

④ If $E \neq 0$, update wt. & bias AND restart

$\nabla w = \eta (y - y') y' (1 - y') x$

$\nabla b = \eta (y - y') y' (1 - y')$

⑤ $w = w + \nabla w$

$b = b + \nabla b$.

2ter n 2

Q)

2ter n

2ter n

Itr^n 2: $x = 1.202$

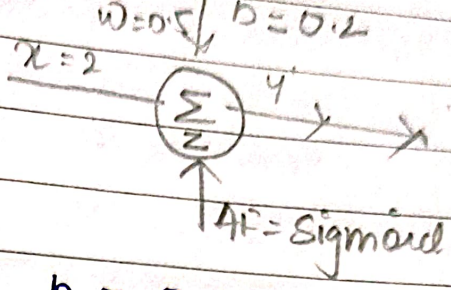
$y' = 0.7688$

$E = 0.0312$

$\nabla w = 0.00109$

$\nabla b = 0.0000554$

$w = 0.50223$; $b = 0.2001124$



Q) $[x_1, x_2] = [2, 3]$; $[w_1, w_2] = [0.4, 0.6]$

$y = 0.8$; $b = 0.2$; $\eta = 0.1$

Itr^n 1: $x = 2.8$; $y' = 0.942$

$E = 0.8 - 0.942 = -0.142$

$\nabla w = [-0.00155, -0.00232]$.

$\nabla b = -0.000775$

$w' = [0.39845, 0.59768]$

$b' = 0.19925$.

Itr^n 2: $x = 2.78919$

$y' = 0.94208$; $E = -0.14208$

$\nabla w = [-0.00155, -0.0$

$\nabla b = -0.0007752$

$w'' = [0.3969, 0.59536]$

$b'' = 0.19847$.