DLP

**Artificial Intelligence (AI)** - AI is the capability of a machine to imitate intelligent human behaviour.

AI is the tech that enables computers and machines to simulate human intelligence and problem solving capabilities.

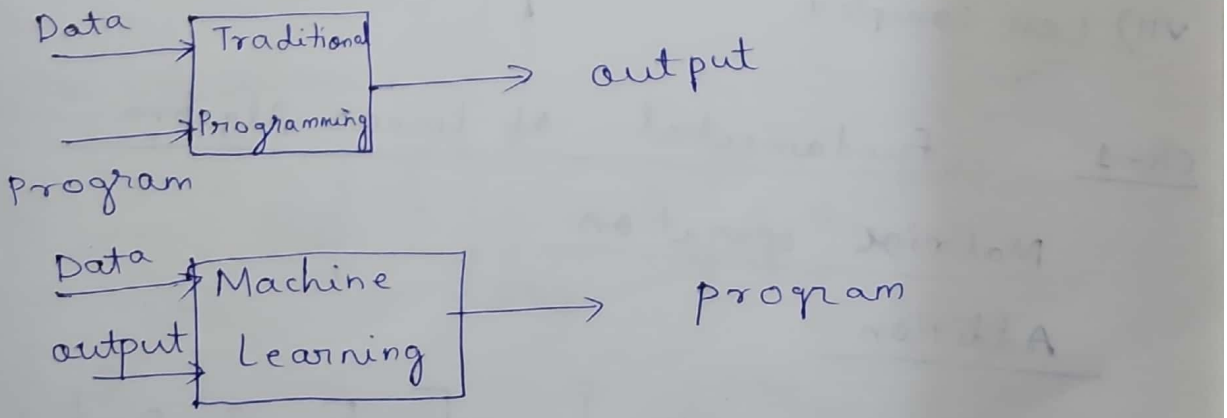
**Eg of Application**

→ search engines (Google)
→ content Recommendations (youtube, Netflix)
→ chatGPT
→ self driving cars
→ facial recognition
→ computer games


**Machine Learning**

It is the study of computer algo that improve automatically through experience.

ML refers to the study of computer systems that learn & adapt automatically from experience. without being explicitly programmed.



**Deep Learning** - is the subset of ML.

DL uses algo called ANN, (Artificial Neural N/W) which are inspired by the structure & function of the brain and are capable of self learning.

## Difference b/w ML & DL

### ML

I) Subset of AI

II) It can train on smaller datasets.

III) Require more human intervention to correct & learn.

IV) shorter training

V) Lower accuracy

VI) It can train on a CPU.

VII) Less complex

### DL

I) Subset of ML

II) It requires train large datasets.

III) It learns of its own from environment & past mistakes

IV) Larger training

V) higher accuracy

VI) It needs specialized GPU.

VII) more complex

## Ch-1 Fundamental of Linear Algebra

### Matrix operation

### Addition

$$\begin{bmatrix} 1 & 0 & 5 \\ 2 & 4 & 7 \end{bmatrix}_{2\times3} + \begin{bmatrix} 5 & 3 & 2 \\ 6 & 1 & 4 \end{bmatrix}_{2\times3} =$$

$$\begin{bmatrix} 6 & 3 & 7 \\ 8 & 5 & 11 \end{bmatrix}_{2\times 3}$$

## Scalar Multiplication

$$2 * \begin{bmatrix} 1 & 0 & 5 \\ 2 & 4 & 7 \end{bmatrix} = \begin{bmatrix} 2*1 & 2*0 & 2*5 \\ 2*2 & 2*4 & 2*7 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 0 & 10 \\ 4 & 8 & 14 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 6 \\ 2 & 7 \end{bmatrix} * \begin{bmatrix} 1 & 9 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 4+30 & 36+12 \\ 2+35 & 18+14 \end{bmatrix}$$

$$= \begin{bmatrix} 34 & 48 \\ 37 & 32 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 6 \\ 2 & 7 \end{bmatrix}\begin{bmatrix} 1 & 9 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 4 & 6 \\ 2 & 7 \end{bmatrix}\begin{bmatrix} 1 \\ 5 \end{bmatrix} & \begin{bmatrix} 4 & 6 \\ 2 & 7 \end{bmatrix}\begin{bmatrix} 9 \\ 2 \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} 34 & 48 \\ 37 & 32 \end{bmatrix}$$

## Vector Operations

$$\begin{bmatrix} 4 & 3 & 6 \end{bmatrix}_{1\times 3}\begin{bmatrix} 2 \\ 7 \\ 1 \end{bmatrix}_{3\times 1} = \begin{bmatrix} 8+21+6 \end{bmatrix}_{1\times 1}$$

$$= [35]_{1\times 1}$$

$$\begin{bmatrix} 2 \\ 7 \\ 1 \end{bmatrix}_{3\times 1}\begin{bmatrix} 4 & 3 & 6 \end{bmatrix}_{1\times 3} = \begin{bmatrix} 8 & 6 & 12 \\ 28 & 21 & 42 \\ 4 & 3 & 6 \end{bmatrix}_{3\times 3}$$

## Matrix Vector Multiplication

$$\begin{bmatrix} 4 & 6 \\ 2 & 7 \\ 5 & 8 \end{bmatrix}_{3\times 2} \begin{bmatrix} 1 \\ 5 \end{bmatrix}_{2\times 1} = \begin{bmatrix} 4+30 \\ \\ \\ \end{bmatrix}_{3\times 1}$$

$$= 1\begin{bmatrix} 4 \\ 2 \\ 5 \end{bmatrix} + 5\begin{bmatrix} 6 \\ 7 \\ 8 \end{bmatrix}$$

$$= \begin{bmatrix} 4 \\ 2 \\ 5 \end{bmatrix} + \begin{bmatrix} 30 \\ 35 \\ 40 \end{bmatrix} = \begin{bmatrix} 34 \\ 37 \\ 45 \end{bmatrix}_{3\times 1}$$

## The fundamental Spaces

① column space

② Null space

### Column Space

If A is a matrix, column space is represented by C(A).

C(A) = all possible combo of columns of A.

$$A = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}, \quad c_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad c_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Feature

→ used for dimension reduction

→ used to find out relationship among features.

## Null Space

The null space of matrix A is represented by N(A).

$Ax = 0$.

$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

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

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

$\Rightarrow \begin{bmatrix} x_1 + 2x_2 \\ 2x_1 + 3x_2 \end{bmatrix} = 0$

$\Rightarrow x_1 + 2x_2 = 0$

$\Rightarrow x_1 = -2x_2$

$\begin{bmatrix} -2x_2 \\ x_2 \end{bmatrix} = x_2 \begin{bmatrix} -2 \\ 1 \end{bmatrix}$

## Eigen Value & Eigen Vector

↓

Vector representation of a matrix

→ It is a non-zero matrix

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

$|A - \lambda I| = 0$

$\Rightarrow \begin{vmatrix} 4-\lambda & 1 \\ 2 & 3-\lambda \end{vmatrix} = 0$

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

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

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

$\Rightarrow \lambda^2 - 2\lambda - 5\lambda + 10 = 0$

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

$(\lambda - 2)(\lambda - 5)$

$\lambda_1 = 2, \lambda_2 = 5$

considering $\lambda_1 = 5$,

$$(A - 5I)x = 0$$

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

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

$$\Rightarrow -x_1 + x_2 = 0$$

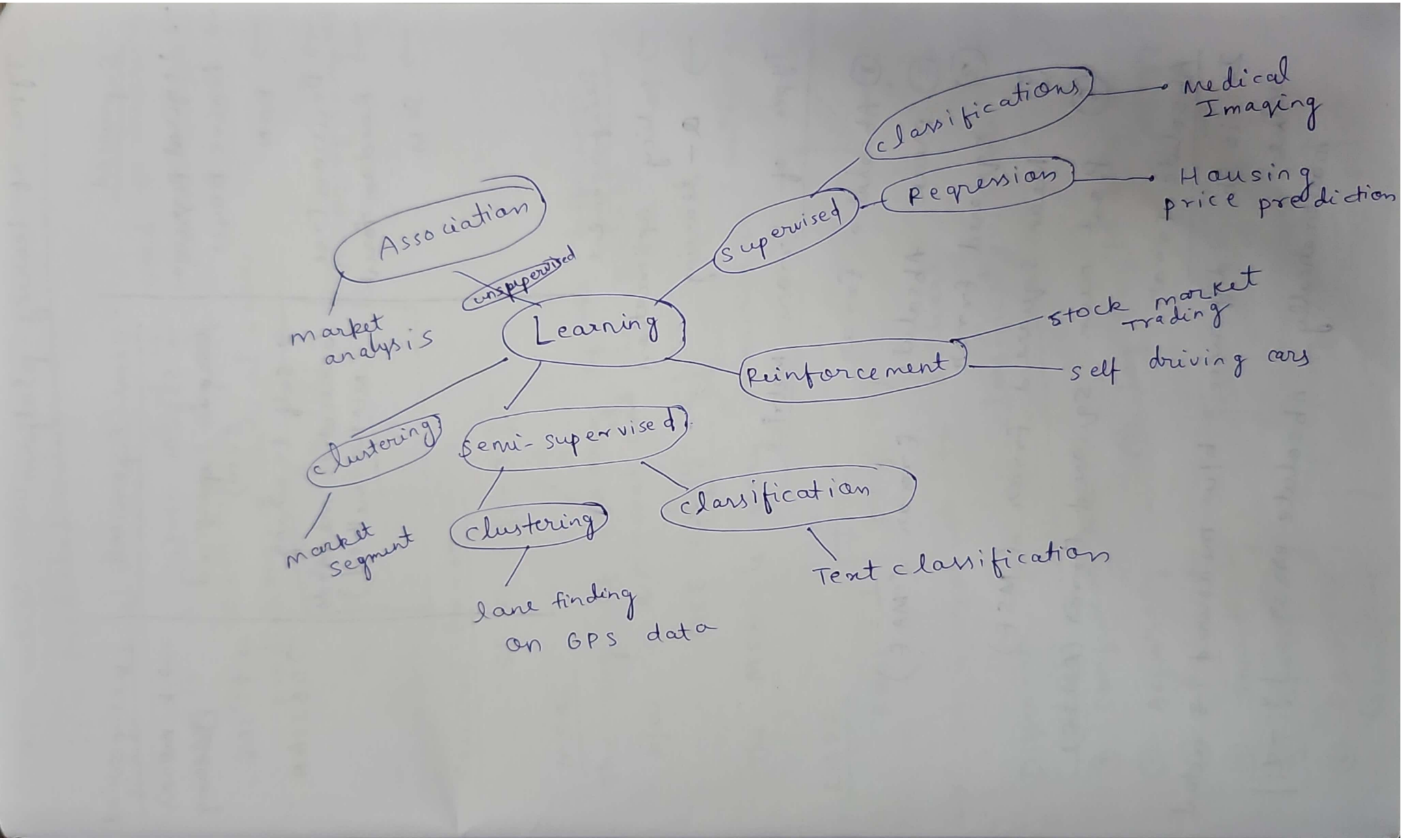
$$\Rightarrow x_1 = x_2$$

let $x_1 = 1$, $x_2 = 1$

eigen value $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$

## Types of Learning

1. supervised Learning - All data are labelled, training data includes desired output. eg - classification, regression
2. unsupervised " - no data is labelled, training data doesn't include desired o/p eg - clustering, generative models
3. semi - supervised " → datum - singular small amt of data is labelled (training data includes desired o/p)
4. Reinforcement " → No labelled data. It tells which step is correct & which step is wrong



# Types of learning platform

## Supervised

- → Logical Regression
- → Naive Bayes
- → KNN
- → Decision Tree
- → Random forest
- → SVM

## semi-supervised

- → SSSVM (semi-supervised SVM)
- → self training
- Generative methods (Mixture method)

## unsupervised

- → k-means clustering
- → T-SNE
- → DBSCAN

## Reinforcement

- → policy optimization
- → Q-learning

## Types of Errors Metrics

1. Absolute Error
2. Mean Absolute Error (MAE)
3. squared Error
4. Mean squared Error (MSE)
5. Root mean squared Error (RMSE)

## Absolute Error

It is the difference b/w measured & actual value.

mathematically, Absolute error = $|y_i - x_i|$

$x_i$ = actual value

$y_i$ = predicted value

## Mean Absolute Error

MAE is a metric that is used to measure the arithmetic avg of deviation b/w predictions & actual value.

$$MAE = \frac{\sum_{i=1}^{n} y_i - x_i}{n}$$

n = no. of samples

## Squared Error

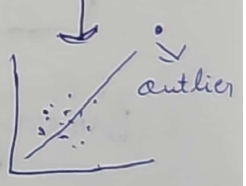
$$SE = (y_i - x_i)^2$$

## MSE

The avg of squares of the difference b/w estimated ~~and~~ at the actual values.

$$MSE = \frac{\sum_{i=1}^{n} [y_i - x_i]^2}{n}$$

The MSE is very sensitive ~~to~~ outliers.




## RMSE

$$RMSE = \frac{1}{n}\sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

→ It is robust to outliers

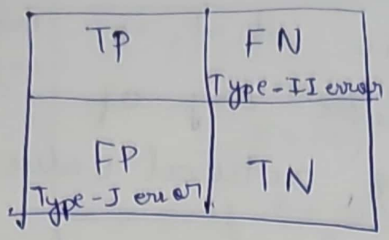
## Different types of classification Evaluation metrics

1. Confusion Matrix
2. Accuracy
3. Precision
4. Sensitivity
5. Specificity
6. F1-score
   ↳ combo of precision & recall
7. Log loss / Binary cross entropy

$$\text{Sensitivity} = TPR = \frac{TP}{TP + FN}$$

$$\text{Specificity} = PPR = \frac{TN}{TN + FP}$$

## Confusion Matrix




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

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

→ useful for skewed dataset / unbalanced

$$\text{F1-score} = \frac{2PR}{P+R}$$

**Binary entropy** – It compares each of the predicted probability to actual class o/p. It is the negative avg of log of corrected predicted probabilities

$$= -\frac{1}{N} \sum_{i=1}^{N} y_i \log\left(p(y_i)\right) + (1-y_i) \cdot \log\left(1-p(y_i)\right)$$

Ch-2 Fundamentals of probability

Entropy, cross-Entropy, KL-divergence

Entropy measure uncertainity and randomness within a single probabilty distribution

## Entropy

$$E(p) = -\sum P(x) \cdot \log P(x)$$

$\Sigma \rightarrow$ represents the sum over all possible events.

$P(x) \rightarrow$ prob. of event $x$ in the distr. prob.

## cross-Entropy

It measures how well a predicted probability distribution ($Q$) approximates a true probability distribution ($P$)

$$CE(P, Q) = -\sum P(x) \cdot \log (Q(x))$$

$\Sigma$ - sum over all posible events

$P(x)$ - true probability of event $x$

$q(x)$ - predicted probability of event $x$

## KL Divergence

① for discrete Distribution

② For continuous "

KL divergence measures the info lost when approximating the distribution $p$ with distribution $q$.

The KL
- It quantifies the difference b/w 2 probability dist.

It results 0, iff the dist are identify.

$$KL(P||Q) = \sum P(i) \cdot \log\left(\frac{P(i)}{Q(i)}\right)$$

$P_i$ of $P$ is the probability of event $i$ in the true dist $P$.

$Q_i$ in the $Q$ " in the approximating dist $Q$.

$\sum$ denote $\Sigma$ of all possible outcomes

$P(i)$ prob. of event $i$ in the predicted distr $Q$.

$$KL(P||Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx$$

## Properties

### Non-negative

$KL(P||Q) \geq 0$

### zero divergence

$KL(P||Q) = 0$ if ~~the~~ two dist are ~~undefined~~ identical

i.e $P = Q$

## Ch-3 The Neural Network

The neuron- It is fundamental unit of human brain.

The avg human brain has approx 100 billions of neurons

$= 10^{11}$

It is a small cell that receive electrochemical signal from various sources & response by transmitting electricall impulses to other neurons.

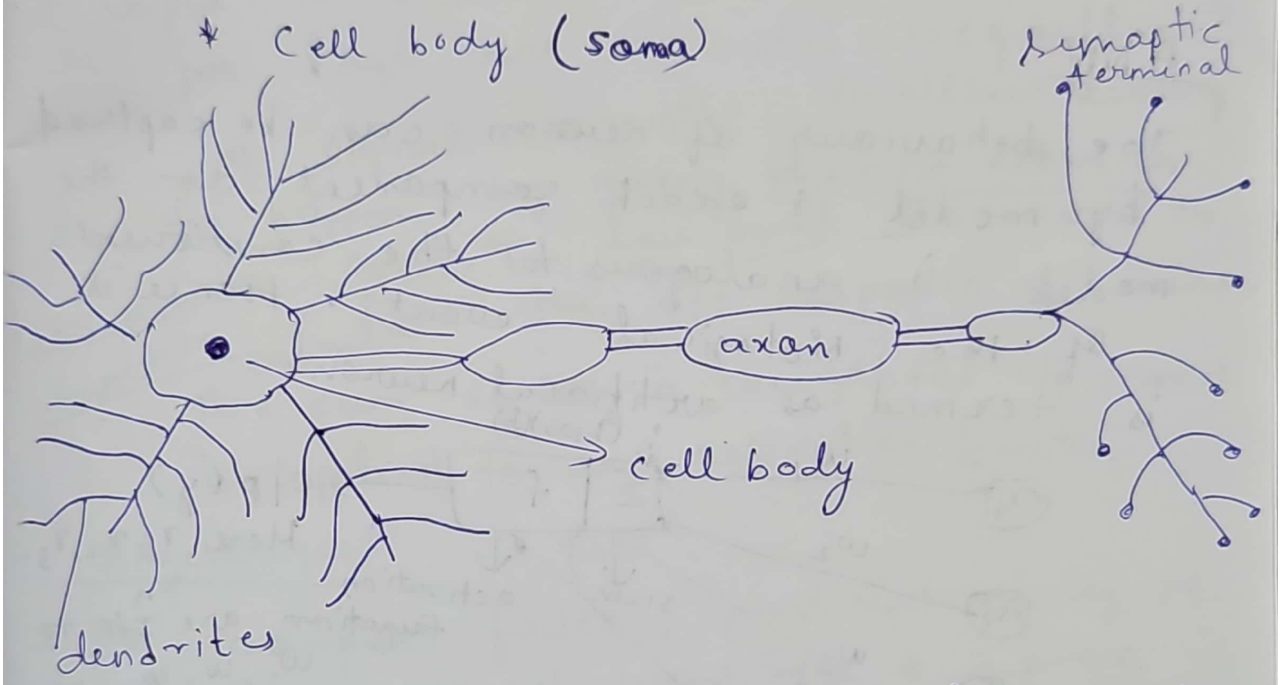
Each neuron is densely connected to $10^4$ no. of neurons.

The weight of brain is 1.5kg

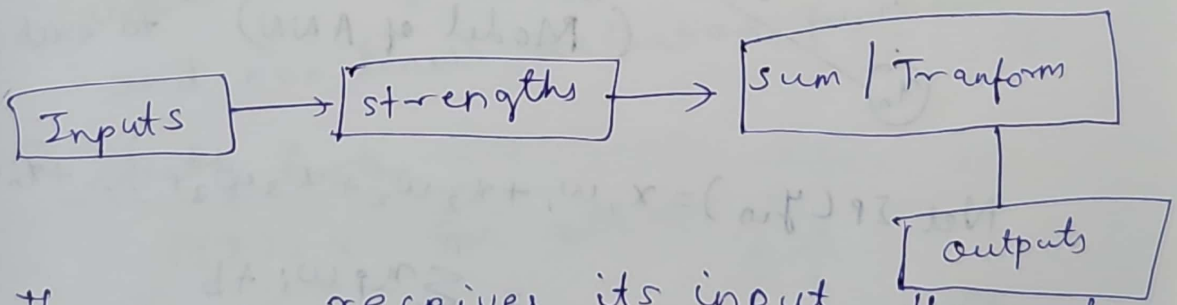
The " " avg neuron is $1.5 \times 10^{-9}$ gm

A neuron consist of 3 main parts.

* dendrites
* Axon
* Cell body (soma)




(Structure of biological neuron)



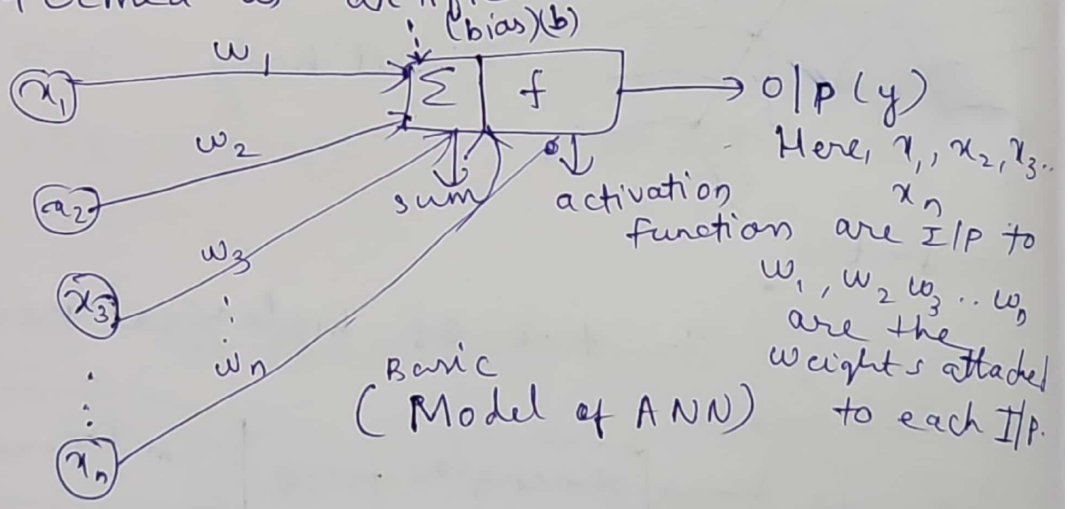

- The neuron receives its input through antena like structure called dendrites
- The dendrites behave as i/p chain
  Each of these incoming connections is dynamically strengthen or weaken based on how often it is used.
- The strength of each connection determines the contribution of i/p to the neurons o/p.

After being waited by the strength of their respective connections, the i/p are summed together in the cell body

The sum is then transformed into a new signal that is propagated along the cells axon & sent off to the other neurons.

## ANN

The behaviour of neuron can be captured by model & each component of the model is analogous to the constituents of the biological events. Hence it is termed as artificial neuron.




Here, $x_1, x_2, x_3 \ldots x_n$ are I/p to $w_1, w_2 w_3 \ldots w_n$ are the weights attached to each I/p.

Basic (Model of ANN)

$$\text{Net Ip}(y_{in}) = x_1 w_1 + x_2 w_2 + x_3 w_3 + \ldots + x_n w_n$$

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

$$y = f\left(\Sigma w_i x_i + b\right)$$

Let's reformulate the i/ps as a vector

$$x = [x_1, x_2, x_3 \ldots x_n]$$

$w = [w_1, w_2, w_3 \ldots w_n]$

Net $I/p(y_{in}) = X \cdot W$

The O/p of the neuron is given by

$$y = f(x \cdot w + b)$$

biases are the scalar values added to the input to ensure that at least a few nodes per layer are activated regardless of signal strength. It allows learning to happen by giving the n/w action in the event of low signal.

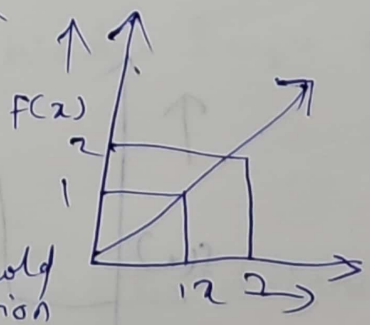
Activation function: It is used to calculate the o/p response of a neuron. The sum of a weighted i/p signal is applied to an activation function

Advantages

a) It computes faster, speed up training, It helps n/w to use useful info & supress irrelevant info. There are some linear and non linear activation functions.

① Identity function

$f(x) = x$ for all $x$

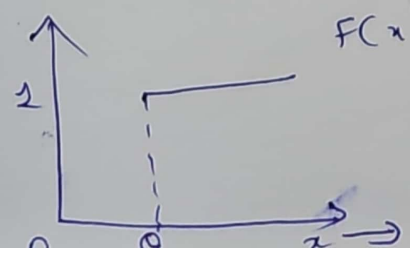



② Binary step function

Step function

→ Also called threshold function

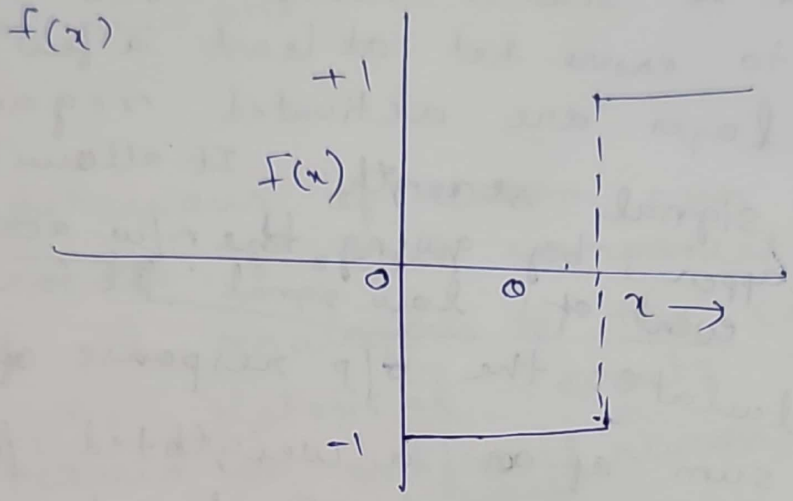
$$F(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{o/w} \end{cases}$$



## Bipolar step function

### signum function

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




### ③ Sigmoid function

there are 2

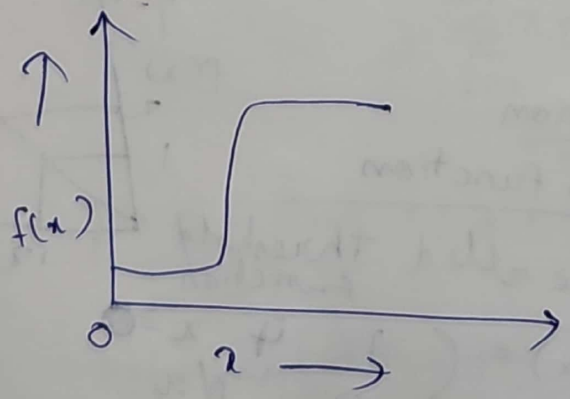
① Binary sigmoidal function - ranges b/w 0 & 1

② Bipolar " "

IA ranges b/w 0 & 1

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

x = [-0.5, 0, 0.5, 0.7



Adv
- easy to train on small dataset
- reduce extreme values or outlets in data without removing

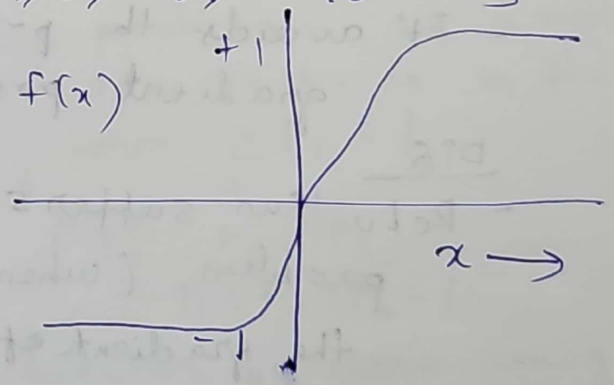
Dis
→ Vanishing gradient problem
→ Not zero centred
→ It is used in o/p (not an hidden layer)

Bipolar sigmoidal Fun
tangent hyperbolic function (tanh)

→ ranges b/w +1 & -1

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

$x = [-1, -0.75, -0.5, 0, 0.25, 0.5, 0.75, 1]$
$f(x) = (\text{---})$




Adv
- It is 0-centred
- It can be used both in hidden layer & o/p layer

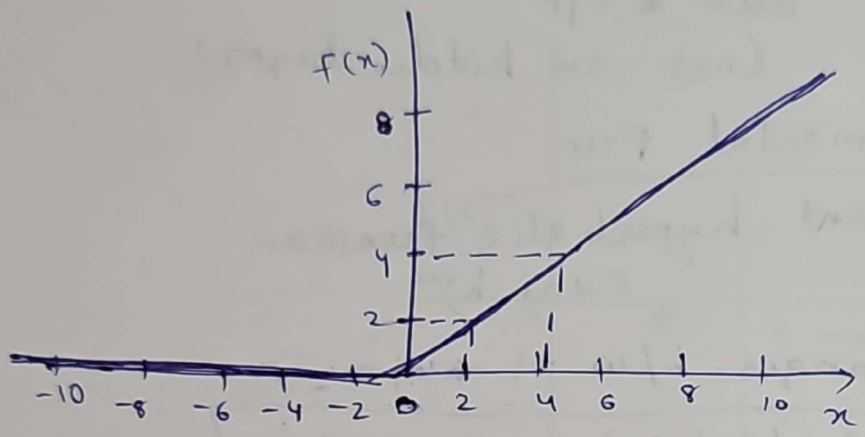
dis
- Vanishing gradient problem
- It is hard to train on smaller datasets

④ ReLU function

Rectified linear Unit

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

Relu fun always gives 0 for -ve values




- It is computationally efficient (It allows the n/w to converge very quickly)
- It avoids the problem of vanishing gradient problem.

Dis

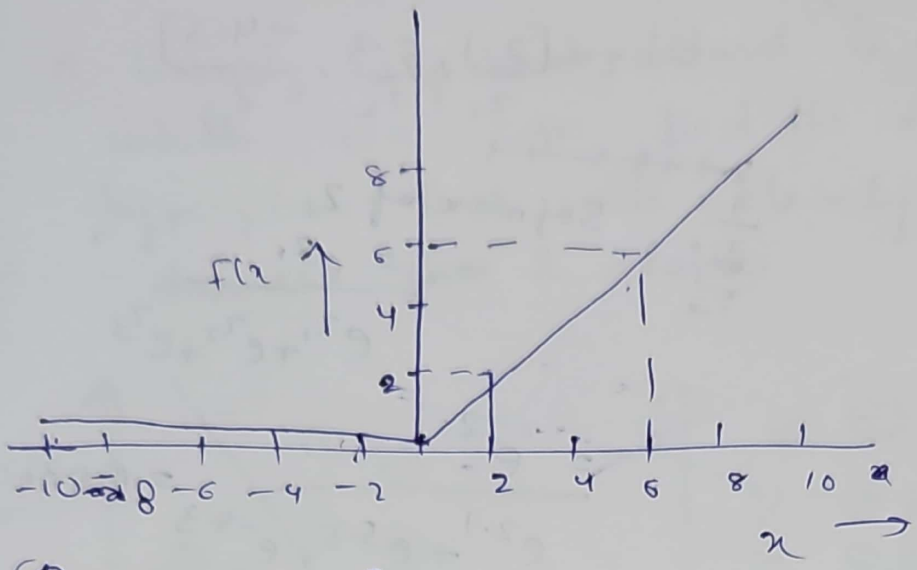
- Relu fun suffers from dead neuron problem (when i/p becomes 0 or -ve, the gradient of the fun becomes 0). So the n/w cannot perform back propagation & can't learn.

⑤ Leaky Relu

Dead neuron problem can be solved by leaky ReLU

- when $x < 0$, the fun will have a small neg slope

$$f(x) \begin{cases} x & \text{if } x<0 \\ 0.01x & \text{if } x<0 \end{cases}$$




(6) Softmax fun

MNIST digit dataset

Sometimes we want our o/p vector to be a probability dist over a set of mutually exclusive levels.

using a probability dist gives us a better idea how confident we are in our prediction. As a result the desired o/p vector is in the following form. $\sum_i P_i = 1$

$$[P_1, P_2, P_3]$$
$$P_1 + P_2 + P_3 = 1$$

This can be achieved by using a special output layer called softmax fun.

The o/p of a neuron in a softmax layer depends on the o/p of all other neurons in its layer. this is because you require sum of all o/ps equal to 1

$$y_i = \frac{e^{z_i}}{\sum e^{z_i}}$$

This fun is mainly used multiclass classification data problems.

$$z = [\underset{z_1}{2.1}, \underset{z_2}{5.5}, \underset{z_3}{-4.3}]$$

Softmax of $z_1$

$$= \frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}$$

$$= \frac{e^{2.1}}{e^{2.1}+e^{5.5}+e^{-4.3}} = 0.0327$$

sm of $z_2$

$$= \frac{e^{5.5}}{e^{2.1}+e^{5.5}+e^{-4.3}} = 0.9676$$

highest 96.1.

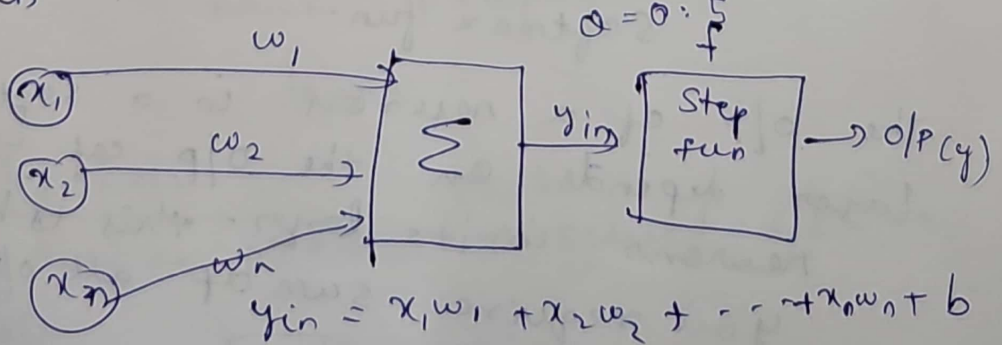
sm of $z_3$

$$= \frac{e^{-4.3}}{e^{2.1}+e^{5.5}+e^{-4.3}} = 5.3657 \times 10^{-5}$$

## The perceptran Model

linear model used for binary classification

This model uses step fun as activation fun



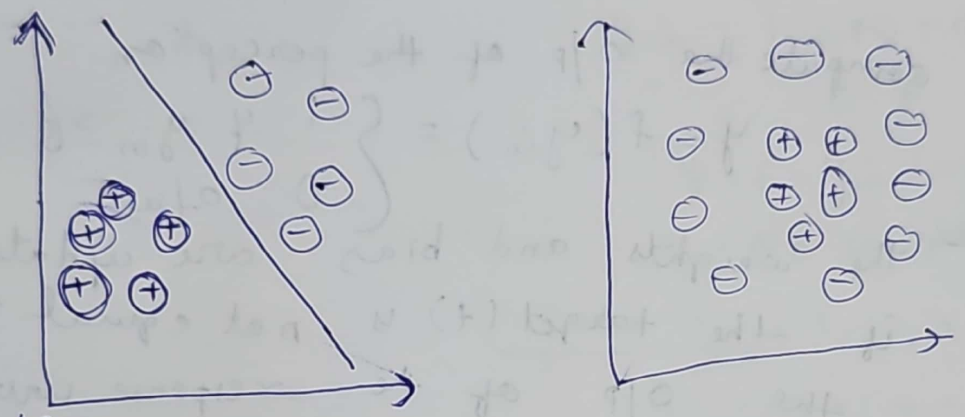

$$y_{in} = x_1w_1 + x_2w_2 + \cdots + x_nw_n + b$$

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

$$y = f(y_{in}) = \begin{cases} 1 & y_{in} >= \theta \\ 0 & y_{in} < \theta \end{cases}$$

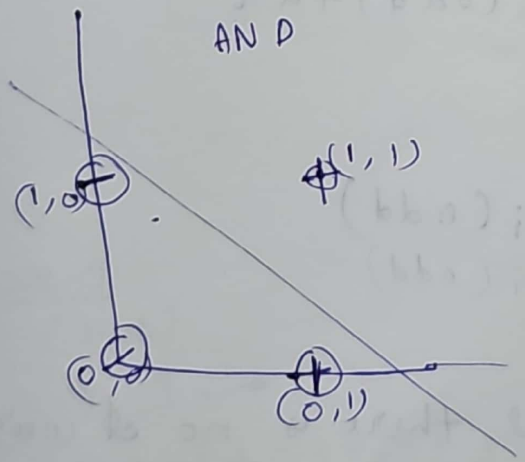
The basic perceptron model is mainly used for linearly separable dataset.

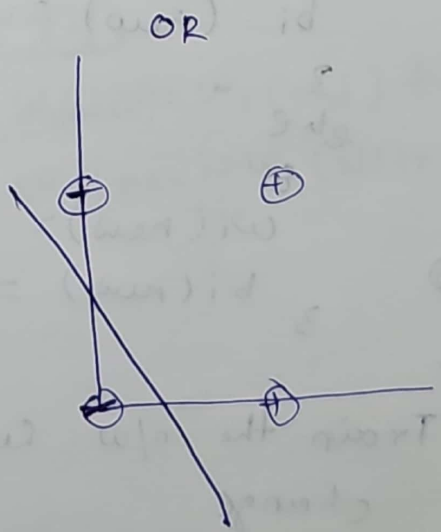
A linearly separable dataset is one for which we can find the values of hyperplain that will clearly divide the dataset into 2 classes.

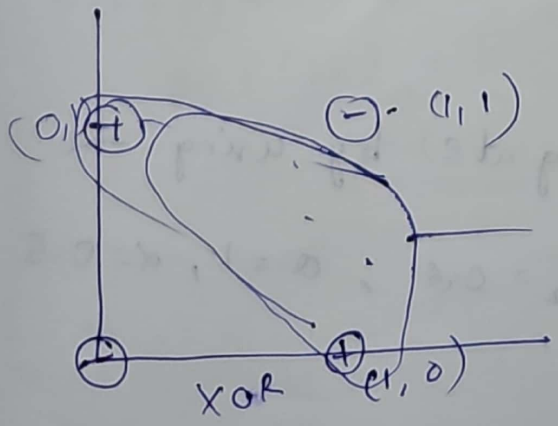


dis

A basic perception cannot solve non-linear problem. (dataset that are not linearly separable).




Inseparable, non-linear

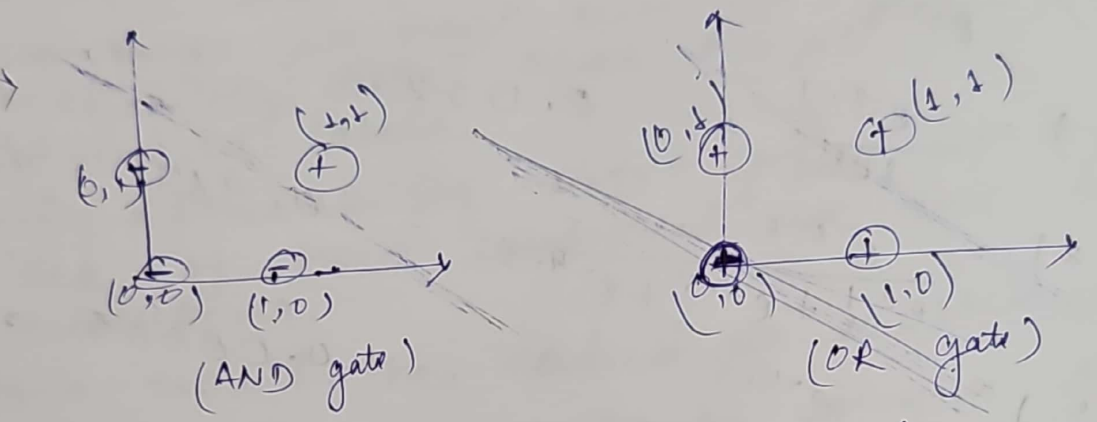
* The basic perception model is mainly used for linearly seperable datasets.

23/09/25

Linearly seperable dataset is one for which we can find values of hyperplane that will clearly divide the dataset into two classes.
- eg:- XOR / AND gate

Limitations

The basic perception model cannot solve non linear problem ( dataset that are not linearly seperable)




But not XOR gate (i.e we cannot draw a line that will seperate the two class)

→ The Training algorithm Perceptron

Step 1
Initialise the weights, bias & learning rate ($\alpha$)

Step 2
Compute the o/p of the response unit.

$$y_{in} = \sum w_i x_i + b$$

Step 3
Compute the o/p of the perceptron

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{otherwise} \end{cases}$$

Step 4
The weights and bias are updated if the target (t) is not equal to the o/p of the response unit (y)

if (t≠y)

{ $w_i(new) = w_i(old) + \alpha(t-y)x_i$

$b_i(new) = b_i(old) + \alpha t$

}

else {

$w_i(new) = w_i(old)$

$b_i(new) = b_i(old)$

}

imp

Step-5

Train the n/w until there is no weight change.

Q Realize the AND gate by using the initial weights $w_1 = 1.2$, $w_2 = 0.6$, $\theta = 1$ (threshold), $\alpha = 0.5$.

soln

| $x_1$ | $x_2$ | $t$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

case 1 $(x_1 = 0, x_2 = 0)$

$y_{in} = w_1 x_1 + w_2 x_2$

$= 1.2 * 0 + 0.6 * 0$

$= 0$

$\therefore y = f(y_{in}) = 0$

$y = t$ (no wt. change)

case 2

$(x_1 = 0, x_2 = 1)$

$y_{in} = 1.2 * 0 + 0.6 * 1$

$= 0.6$

$y = f(y_{in}) = 0$

$\therefore y = t$

$\Rightarrow$ no weight change

case 3

$(x_1 = 1 \quad x_2 = 0)$

$y_{in} = 1.2 * 1 + 0.6 * 0$

$= 1.2$

$y = f(y_{in}) = 1$

$\therefore y \neq t$

(as $t = 1$)

so wt. change is required.

$\therefore$ new, $w_1(new) = w_1(old) + \alpha(t-y)x_1$

$w_2(new) = w_2(old) + \alpha(t-y)x_2$

$\therefore$ $w_1(new) = 1.2 + 0.5(0-1)\cdot 1$

$= 1.2 - 0.5 = 0.7$

$w_2(new) = 0.6 + 0.5(0-1)\cdot 0$

$= 0.6$

new,

$y_{in} = w_1 x_1 + w_2 x_2$

$= 0.7 \times 1 + 0.6 \times 0$

$= 0.7$

$y = f(y_{in}) = 0$

$\therefore y = t$

so no wt. change.

case-4 $x_1 = 1$ , $x_2 = 1$

$y_{in} = w_1 x_1 + w_2 x_2$

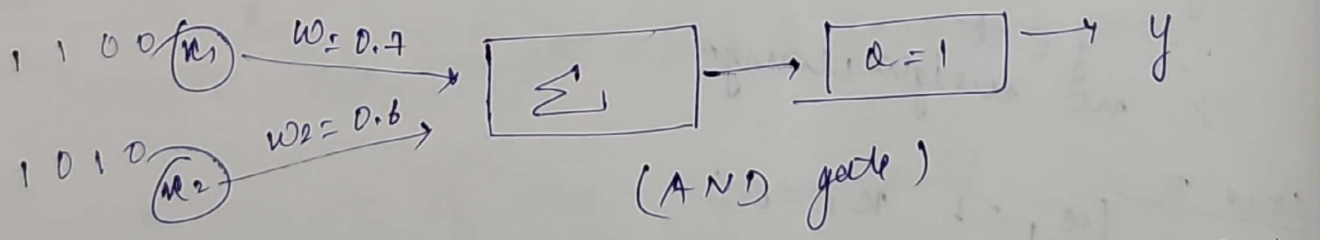
$= 0.7 \times 1 + 0.6 \times 1$

$= 1.3$

$y = f(y_{in}) = 1$

$y = t$

so no wt. change




(AND gate)

② Realize the OR gate by using the initial weight $w_1 = 0.6$ $w_2 = 1.1$ $\theta = 1$ $\alpha = 0.5$

Soln

| $x_1$ | $x_2$ | $t$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Case I → $x_1 = 0, \quad x_2 = 0$

$y_{in} = w_1 x_1 + w_2 x_2$

$= 0.6 * 0 + 0.1 * 0$

$= 0$

$y = f(y_{in}) = 0$

$\therefore y = t$

so no wt. change

Case II

$x_1 = 0, \quad x_2 = 1$

$y_{in} = w_1 x_1 + w_2 x_2$

$= 0.6 * 0 + 1.1 * 1$

$= 1.1$

$y = f(y_{in}) = 1$

$y = t$

so no wt change

Case III

$x_1 = 1 \quad x_2 = 0$

$y_{in} = w_1 x_1 + w_2 x_2$

$= 0.6 * 1 + 1.1 * 0$

$= 0.6$

$y = f(y_{in}) = 0$

but $t = 1$

$\therefore y \neq t$

so wt. change req.

$w_1(new) = w_1(old) + \alpha(t - y) x_1$

$\Rightarrow w_1(new) = 0.6 + 0.5(1 - 0) * 1$

$= 0.6 + 0.5$

$w_1(new) = 1.1$

$w_2(new) = w_2(old) + \alpha(t - y) x_2$

$= 1.1 + 0.5(1 - 0) * 0$

$w_2(new) = 1.1$

New

$y_{in} = 1.1 * 1 + 1.1 * 0$

$= 1.1$

$y = f(y_{in}) = 1$

$\therefore y = t$

so no wt change.

Case IV → $x_1 = 1 \quad x_2 = 1$

$y_{in} = w_1 x_1 + w_2 x_2$

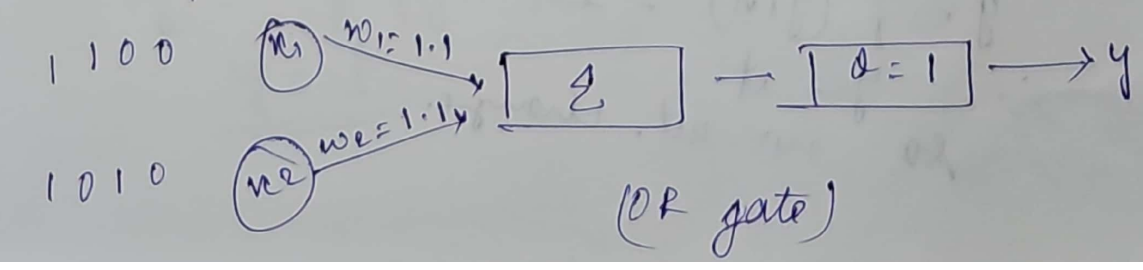
$= 1.1 * 1 + 1.1 * 1$

$= 2.2$

$y = f(y_{in}) = 1$

$y = t$

so no wt. change.




(OR gate)

Q Implement OR gate by using the initial weights
$w_1 = 0 \quad w_2 = 0 \quad \alpha = 0.5 \quad \theta = 1$

| $x_1$ | $x_2$ | $t$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Case I**

$x_1 = 0 \quad x_2 = 0$

$y_{in} = w_1 x_1 + w_2 x_2$
$= 0 * 0 + 0 * 0$
$= 0$

$y = f(y_{in}) = 0$
$\because y = t$
so no wt. change

**Case II** $x_1 = 0 \quad x_2 = 1$

$y_{in} = w_1 x_1 + w_2 x_2$
$= 0 * 0 + 0 * 1$
$= 0$

$y = f(y_{in}) = 0$
but $t = 1$
$\therefore y \neq t$
wt change req.

$w_1(new) = w_1(old) + \alpha (t - y) x_1$
$= 0 + 0.5 (1 - 0) 0$
$= 0$

$w_2(new) = 0 + 0.5 (1 - 0) * 1$
$= 0.5$

**Now**

$y_{in} = 0 + 0.5 * 1$
$= 0.5$

$y = f(y_{in}) = 0.$
$y \neq t$ wt. change req.

$w_1(new) = 0 + 0.5(1 - 0) 0 = 0$
$w_2(new) = 0.5 + 0.5(1 - 0) * 1 = 1$

**Now** $y_{in} = 0 + 1 * 1 = 1$
$y = f(y_{in}) = 1$
$\therefore y = t$
so no wt. change.

Case III $x_1 = 1 \quad x_2 = 0$

$y_{in} = w_1 x_1 + w_2 x_2$

$= 0 * 1 + 1 * 0$

$= 0$

$y = f(y_{in}) = 0$

but $t = 1$

so wt change req.

New

$w_1(new) = 0 + 0.5(1-0) * 1$

$w_1(new) = 0.5$

$w_2(new) = 1 + 0.5(1-0) * 0$

$= 1$

$y_{in} = w_1 x_1 + w_2 x_2$

$= 0.5 * 1 + 1 * 0$

$= 0.5$

ND

$w_1(new) = 0.5 + 0.5(1-0) * 1$

$= 1$

$w_2(new) = 1 + 0.5(1-0) * 0$

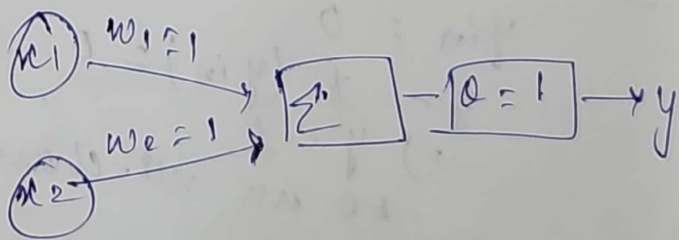
$= 1$

$y_{in} = 1 * 1 + 1 * 0$

$= 1$

$y = f(in) = 1$

$\therefore y = t$




Case IV $x_1 = 1 \quad x_2 = 1$

$y_{in} = 1 * 1 + 1 * 1$

$= 2$

$y = f(y_{in}) = 1$

$y = t$

24/09/25 $w_1 = 0$, $w_2 = 0$ $\alpha = 0.5$ $\theta = 1$

Using AND gate

| $x_1$ | $x_2$ | $t$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**case I** $x_1 = 0$ $x_2 = 0$

$y_{in} = w_1 x_1 + w_2 x_2$
$= 0 * 0 + 0 * 0$
$= 0$

$y = f(y_{in}) = 0$
$\because y = t$
so no wt. change

**case II** $x_1 = 0$ $x_2 = 1$

$y_{in} = w_1 x_1 + w_2 x_2$
$= 0 * 0 + 0 * 1$
$= 0$

$y = f(y_{in}) = 0$
$\Rightarrow y = t$
so no wt. change

**case III** $x_1 = 1$ $x_2 = 0$

$y_{in} = 0$
$y = f(y_{in}) = 0$
$\Rightarrow y = t$
so no wt. change

**case IV** $x_1 = 1$ $x_2 = 1$

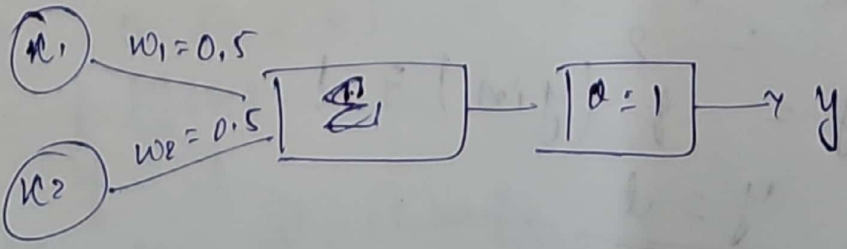
$y_{in} = 0$
$y = f(y_{in}) = 0$
but $y \neq t$.
so $y \neq t$.
so wt. change req.

$w_1(new) = w_1(old) + \alpha(t - y) x_1$
$\Rightarrow w_1(new) = 0 + 0.5(1 - 0)1$
$= 0.5$

$w_2(new) = 0 + 0.5(1 - 0)1$
$= 0.5$

$y_{in} = w_1 x_1 + w_2 x_2$
$= 0.5 * 1 + 0.5 * 1$
$= 1$

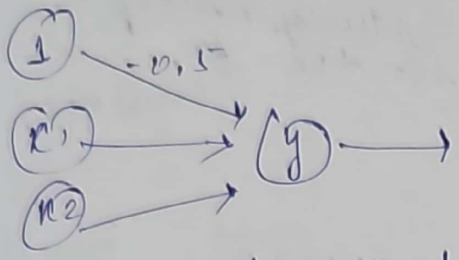
$\Rightarrow y = f(y_{in}) = 1$
& $t = 1$
$\Rightarrow y = t$
$\therefore$ no wt. change req.



Q. Use a perceptron to implement OR gate with the following info. The line equation is $x_1 + x_2 - 0.5 = 0$ and the step activation fn is as follows: $f(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$

Soln

$x_1 + x_2 - 0.5 = 0$



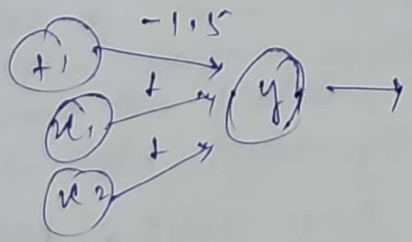

$y = f(-0.5 + x_1 + x_2)$

| $x_1$ | $x_2$ | $x_1 \cup x_2$ | $(-0.5 + x_1 + x_2)$ | $y$ |
|---|---|---|---|---|
| 0 | 0 | 0 | -0.5 | 0 |
| 0 | 1 | 1 | 0.5 | 1 |
| 1 | 0 | 1 | 0.5 | 1 |
| 1 | 1 | | 0.5 | 1 |

Q. Use perception to implement AND gate. The line eqn is $x_1 + x_2 - 1.5 = 0$ and the step activation fn is same as above $f(z) = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases}$

Soln

| $x_1$ | $x_2$ | $t$ | $z = -1.5 + x_1 + x_2$ | $y = f(z)$ |
|---|---|---|---|---|
| 0 | 0 | 0 | -1.5 | 0 |
| 0 | 1 | 0 | -0.5 | 0 |
| 1 | 0 | 0 | -0.5 | 0 |
| 1 | 1 | 1 | 0.5 | 1 |

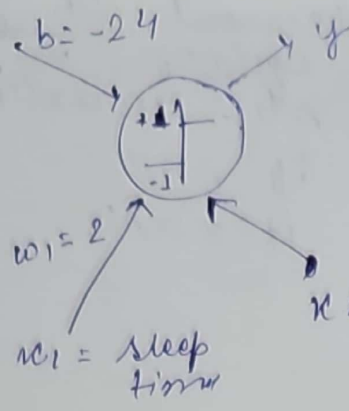



Q. Build a perceptron to predict the student success based on time spent on study and sleep.

Let $x_1$ = sleep time
$x_2$ = study time

$w1 = 2$ $w2 = 6$ $b = -24$

Which feature contributes more towards the prediction
Justify your answer.



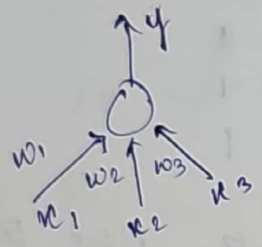

$$h(x/\theta) = \begin{cases} -1 & \text{if } 2x_1 + 6x_2 - 24 < 0 \\ +1 & \text{if } 2x_1 + 6x_2 - 24 > 0 \end{cases}$$

as $w_2 > w_1$ i.e study > sleep so +ve fn.

06/10/25

## Linear Neurons and Their Limitations

$$y(i) = x_1 w_1(i) + x_2 w_2(i) + x_3 w_3(i)$$




Linear Neurons are easy to compute. But they have severe limitations. Because linear neurons can be expressed as a network with hidden layers. Hidden layers enable us to learn important features from the i/p layer.

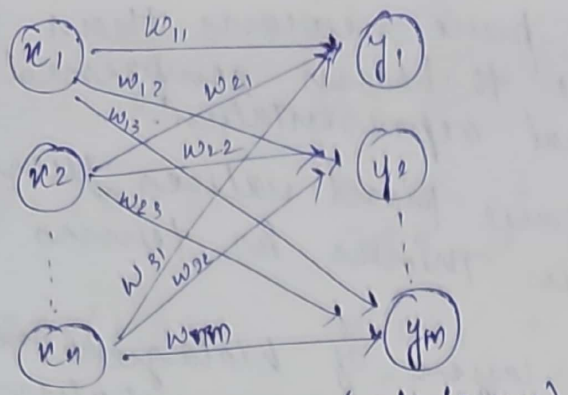
In other words, in order to learn complex relationships, we need to use neurons that employ some sort of non-linearity.

## Feed Forward Neural Network

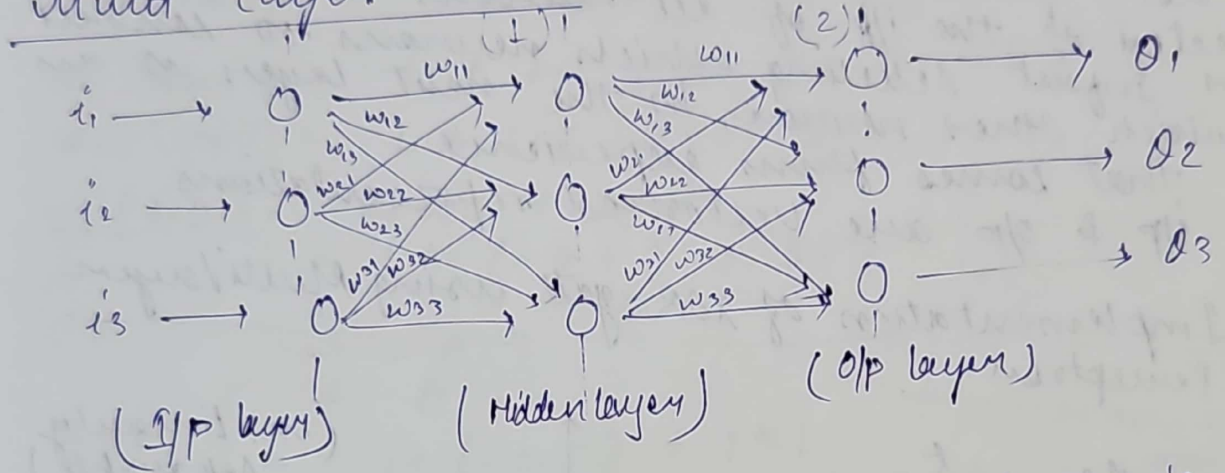
1. Single layer FNN
2. Multi layer FNN

## Single Layer FNN

Despite two layers the n/w is termed as single layer because all the computation being performed in the O/P layer and the i/p layer doesn't perform any computation. Hence the n/w is known as single layer feed forward Neural Network.




## Multi layer FNN



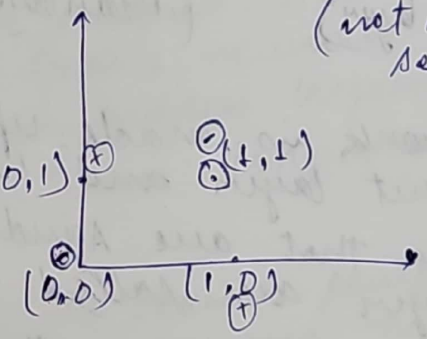

This network is made up of multiple layers i.e input layer, output layer and hidden layer.
The layers that are sandwiched b/w input layer & output layer are called hidden layer.
In this layer most of the magic happens when the neural network tries to solve problems.
e.g- In handwriting digit recognition, we have to spend a lot of time identifying useful features. But this hidden layers automate this process for us. That means the features are automatically extracted from the data through hidden layers.
(Diag :- This is a simple Feed Forward Neural Network with 3 layers and 3 neurons per layer)
Though in this example every layer has the same number of this is neither necessary. nor recommended.

More often hidden layers have more neurons than the i/p layers. To force the nw to learn compressed representation of the original representation.

eg:- while obtain raw pixel values from our surroundings, our brain thinks in terms of edges and contours.
This is because the hidden layer of biological neurons force us to come up with better representation for everything we perceive.
It is not required that every neuron has its output connected to the i/p of all neurons in the next layer. Input selecting which neurons to connect to which other neurons in the next layer is an art that comes from experience.
The i/p & o/p are vectorized representations.

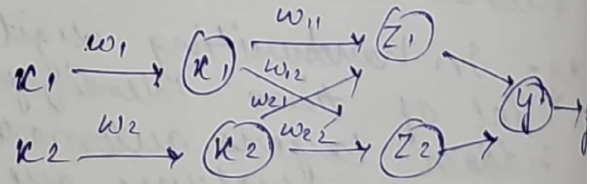
→ Implementation of XOR gate using Multilayer Perceptron

| $x_1$ | $x_2$ | $t$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |
| 1 | | |




$y = 01 + 10 \quad \text{or} \quad y = \frac{\bar{x}_1 x_2}{z_1} + \frac{x_1 \bar{x}_2}{z_2}$

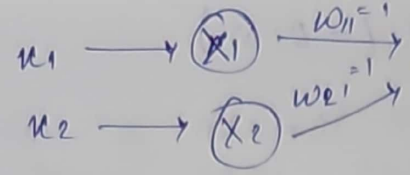
$z_1 = \bar{x}_1 \cdot x_2$ (1st fn)
$z_2 = x_1 \cdot \bar{x}_2$ (2nd fn)
$y = z_1 + z_2$ (3rd fn)




Assume initial weights $w_{11} = 1$, $w_{12} = 1$, $\theta = 1$, $\eta = 0.5$

1st fn $z_1 = \bar{x}_1 \cdot x_2$

| $x_1$ | $\bar{x}_1$ | $x_2$ | $z_1$ (t) |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |




$Z_1 = x_1 w_{11} + x_2 w_{21}$

$(0,0) = 0*1 + 0*1$

$= 0$ ✓

$Z_1(0,1) = x_1 w_{11} + x_2 w_{21}$

$= 0*1 + 1*1 = 1$ ✓

$Z_1(1,0) = x_1 w_{11} + x_2 w_{21}$

$= 1*1 + 0*1$

$= 1$ (mismatched, need to change)

$w_{11}(\text{new}) = w_{11}(\text{old}) + \eta(t-y)\, x_1$

$w_{21}(\text{new}) = w_{21}(\text{old}) + \eta(t-y)\, x_2$

$w_{11}(\text{new}) = 1 + 1.5(0-1)1 = -0.5$

$w_{21}(\text{new}) = 1 + 1.5(0-1)0 = 1$

$Z_1(1,0) = x_1 w_{11} + x_2 w_{21}$

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

$= -0.5 + 0$

$= -0.5$

$= 0$ ✓

$Z_1(1,1) = 1*(-0.5) + 1*1$

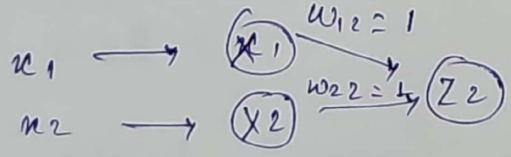
$= -0.5 + 1 = 0.5 = 0$ ✓

$w_{11} = -0.5$

$w_{21} = 1$

2nd fn $Z_2 = x_1 \bar{x}_2$

| $x_1$ | $x_2$ | $\bar{x}_2$ | $Z_2(t)$ |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |




$Z_1(0,0) = x_1 w_{12} + x_2 w_{12}$

$= 0*1 + 0*1 = 0$ ✓

$Z_2(0,1) = x_1 w_{12} + x_2 w_{22}$

$= 0*1 + 1*1 = 1$ (mismatched)

$w_{12}\text{ new} = 1 + 1.5(0-1)0$

$= 1$

$w_{22}\text{ new} = 1 + 1.5(0-1)1$

$= -0.5$

$Z_2(0,1) = 0*1 + 1*(-0.5)$

$= -0.5 \approx 0$ ✓

$z_2(1,0) = 1*1 + 0*(-0.5)$

$= 1 + 0$

$= 1$

$z_2(1,1) = 1*1 + 1*(-0.5)$

$= 1 - 0.5$

$= 0.5 \simeq 0$ ✓

$w_{12} = 1$

$w_{22} = -0.5$

3rd fn

$y = z_1 + z_2$

$= z_1 v_1 + z_2 v_2$

| $z_1$ | $z_2$ | $t$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

$y(0,0) = 0*1 + 0*1$

$= 0$

$y(0,1) = 0*1 + 1*1$

$= 1$

$y(1,0) = 1*1 + 0*1$

$= 1$

$y(1,1) = 1*1 + 1*1$

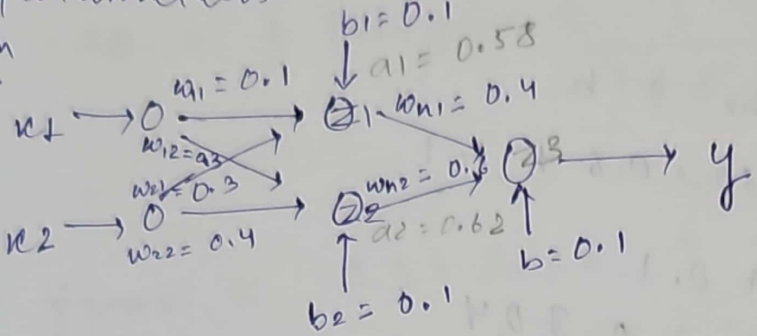
$= 2 \simeq 1$

∴ all 4 condition matched

No need of weight change.

08/10/25

Q(i) Consider a neural n/w with one i/p layer, one hidden-layer with two neurons and one o/p layer with one neuron. Draw the architecture of neural n/w with given data and calculate the total no. learnable parameters.




no. of learnable parameters = 9

$x_1 = 0.5$

$x_2 = 0.6$

(ii) calculate the o/p of the n/w in the forward propogation.

(iii) calculate the error at o/p layer for the actual o/p $y = 1$.

Assume neurons having sigmoid activation fn.

sol$^n$.

$$z_1 = x_1 w_{11} + x_2 w_{22} + b_1$$

$$= 0.5 \times 0.1 + 0.6 \times 0.3 + 0.1$$

$$= 0.05 + 0.18 + 0.1$$

$$= 0.23 + 0.1 = \cancel{0.33} \quad \boxed{0.33}$$

$$a_1 = f(z_1) = \frac{1}{1+e^{-x}} \quad (\because \text{sigmoid fn})$$

$$= \frac{1}{1+e^{-0.33}} = \frac{e^{0.33}}{1+e^{0.33}} = \boxed{0.58}$$

$$z_2 = x_1 \cdot w_{12} + x_2 \cdot w_{22} + b_2$$

$$= 0.5 \times 0.3 + 0.6 \times 0.4 + 0.1$$

$$= 0.15 + 0.24 + 0.1$$

$$= 0.39 + 0.1 = \underline{0.49}$$

$$a_2 = f(x_2) = \frac{1}{1+e^{-x}}$$

$$= \frac{1}{1+e^{-0.49}} = \frac{e^{0.49}}{1+e^{0.49}} = 0.619 \approx 0.62$$

$$Z_3 = a_1 w_{h1} + a_2 w_{h2} + b$$

$$= 0.58 * 0.4 + 0.62 * 0.6 + 0.1$$

$$= 0.232 + 0.372 + 0.1$$

$$= 0.604 + 0.1 = \underline{0.704}$$

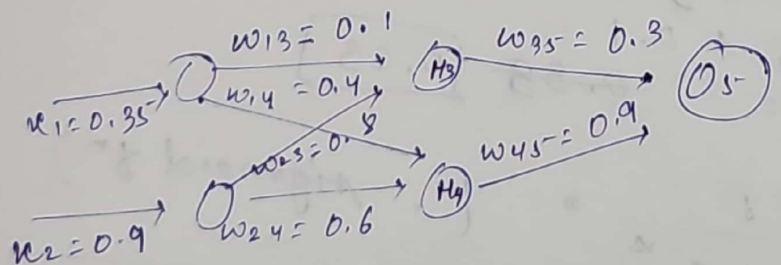
$$y = \frac{1}{1+e^{-0.704}} = \underline{0.668} \approx 0.67$$

$\therefore$ $y_{predicted} = \underline{0.67}$

Also, error $= y_{actual} - y_{pred.}$

$= 0.33 = \underline{33\%}$.

①




given sigmoid fn
actual o/p = 0.5

Sol:

$$H_3 = x_1 w_{13} + x_2 * w_{23}$$

$$= 0.35 * 0.1 + 0.9 * 0.8$$

$$= 0.035 + 0.720$$

$$= 0.755$$

$$a_1 = f(H_3) = \frac{1}{1+e^{-0.755}} = 0.68$$

$H_4 = x_1 w_{14} + x_2 w_{24}$

$= 0.35 * 0.4 + 0.9 * 0.6$

$= 0.140 + 0.54$

$= 0.680$

$a_4 = f(H_4) = \frac{1}{1 + e^{-0.680}} = 0.663$

$O_5 = 0.3 * 0.68 + 0.9 * 0.66$

$= 0.204 + 0.594$

$= 0.798$

$y = \frac{1}{1 + e^{-0.798}} = 0.689 \approx 0.69$

$Error = |y_{actual} - y_{pred}|$

$= |0.5 - 0.69|$

$= 0.19 \approx 19\%$

13/10/25

# Gradient Descent & Delta Rule :-

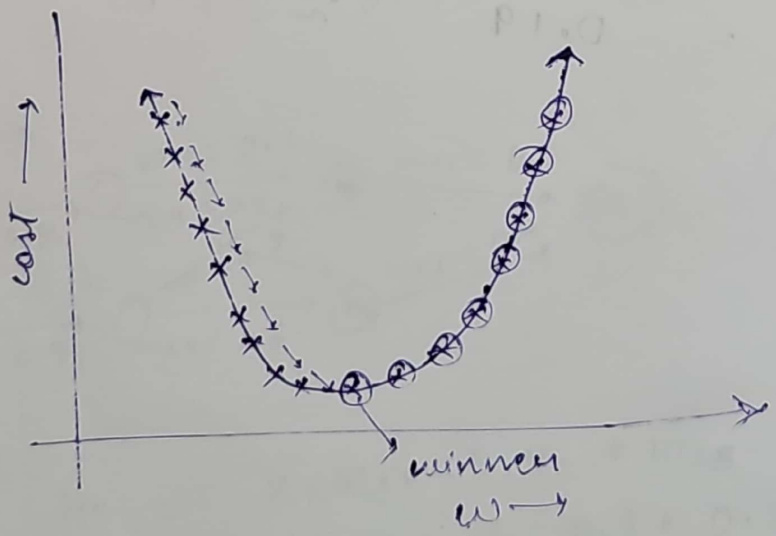
Perceptron rule finds a successful weight vector when the training examples are linearly seperable.

But it fails to converge if the examples are not linearly seperable.

- The Delta Rule is designed to overcome this difficulty
- If the training examples are not linearly seperable, the delta rule converges towards a best fit approximation to the target concept.
- The key idea behind the delta Rule is to use Gradient Descent.

To search the hypothesis space of possible weight vector to find the weights that best fits the training example.

This Rule is imp because this provides the basis for backpropagation algo.




The Delta training Rule is best understood by considering the task of training a single threshold perceptron. i.e a linear unit for which the output is given by

$y = w_0 + w_1 x_1 + w_2 x_2 \dots - w_n x_n$ ;

If or $y(\vec{x}) = \vec{w} \cdot \vec{x}$

* In order to derive weight learning rule for two linear units, let us begin the specifying a measure for the training error of a hypothesis relative to the training examples.

Although there are many ways to define this error one common measure is

$$E(\vec{w}) = \frac{1}{2} \sum_{i \in N} (t_i - y_i)^2$$

$t_i$ → target o/p for ith training example

$y_i$ → calculated o/p (or output of the linear unit for ith training example)

$N$ → set of training examples.

Q. How can we find out the direction of steepest descent along the error surface?

The direction can be found by computing the derivative of $E$ w.r.t $\vec{w}$ and written as $\nabla E(\vec{w})$.

$$\nabla E(\vec{w}) \equiv \left[ \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots \frac{\partial E}{\partial w_n} \right]$$

∴ the gradient specifies the direction of steepest rise of $E$. The training rule for gradient descent is:-

$$\vec{w} \leftarrow \vec{w} + \Delta \vec{w}$$

where, $\Delta \vec{w} = -\eta \cdot \nabla E(\vec{w})$

$\eta$ → learning rate, which determines the step size in the gradient descent surface.

The -ve sign is present because we want to move the weight vector in the direction that ↓es $E$.

This training rule can also be written in component form :-

$w_i \leftarrow w_i + \Delta w_i$

where $\Delta w_i = -\eta \cdot \frac{\partial E}{\partial w_i}$

$$E(\vec{w}) = \frac{1}{2} \sum_i (t_i - y_i)^2$$

$$\frac{\partial E}{\partial w_i} = \frac{1}{2} * 2(t_i - y_i) \frac{\partial}{\partial w_i}(t_i - y_i)$$

$$= \sum (t_i - y_i) \cdot \left[0 - \frac{\partial}{\partial w_i}(w_i x_i)\right]$$

Constant: $t_i \Rightarrow 0$, $y_i = w_i x_i$

$$= \sum (t_i - y_i)(-x_i)$$

$$\boxed{\frac{\partial E}{\partial w_i} = -x_i \sum_{i \in N} (t_i - y_i)}$$

$$\Delta w_i = -\eta \cdot -x_i \sum_{i \in N} (t_i - y_i)$$

$$= \eta x_i \sum_{i \in N} (t_i - y_i)$$

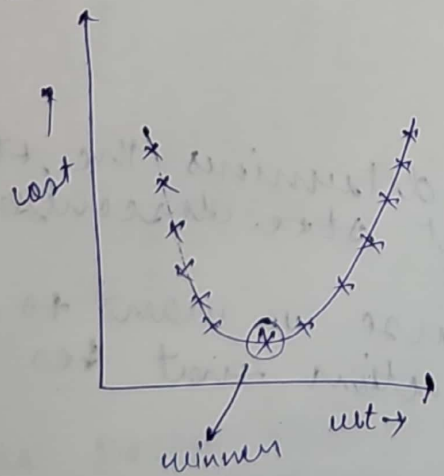
15/10/25

## The delta rule and learning rates

$$E(\vec{w}) = \frac{1}{2} \sum_{i \in N} (t_i - y_i)^2$$

$$w_i \leftarrow w_i + \Delta w_i$$

where $\Delta w_i = -\eta \cdot \frac{\partial E}{\partial w_i}$




ideal value for learning rate = $\eta = 0.001$

if the value is too less then it will take more time steps to reach converging point.

Imp Q What is the effect of learning Rate ($\eta$) on what will happen when $\eta$ is too high or too low?

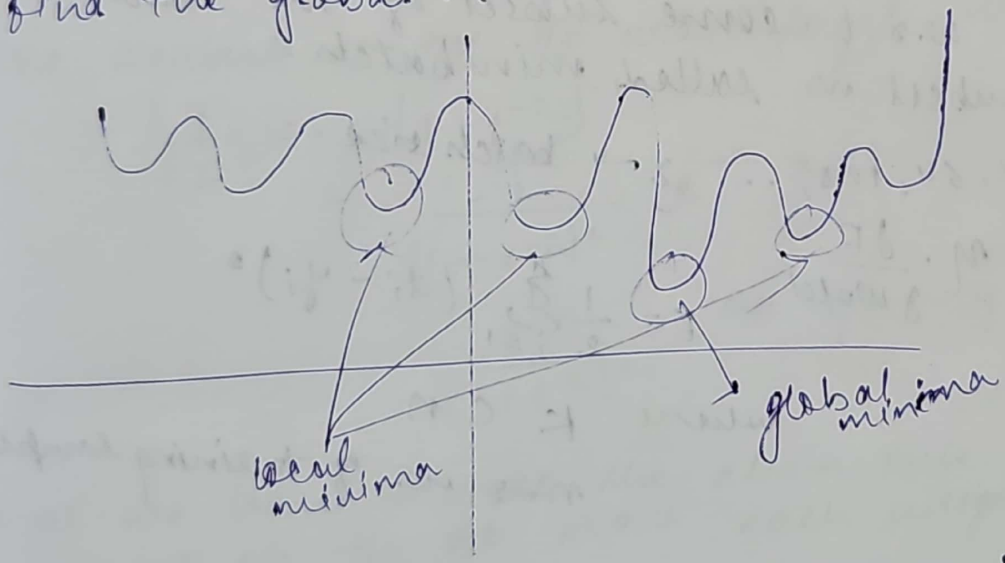
→ If the learning rate is too high or too low then convergence is difficult ~~i.e~~ that means we ~~cannot~~ may not come to the min^m point.

If learning rate is too slow then it will take too long during the training process.

## Limitations of Gradient Descent

Converging to a local min^m can sometimes be quite slow (it may require min^m 1000 gradient descent steps)

If there are multiple local min^m in the surface, then there is no guarantee that the procedure will find the global minima.




Imp Stochastic Gradient Descent (SGD) & Minibatch Gradient Descent :-

In gradient descent, we use our entire dataset to compute the error surface and then follow the gradient to take the path of the steepest descent

$$w_{new} = w_{old} - \eta \frac{dE}{dw_{old}}$$

where $E = \frac{1}{2} \sum_{i=1}^{n} (t_i - y_i)^2$

n → no. of training sample.

* Hence training gradient descent algo. requires more resource and more calculation and it is very complex.

* gradient descent also called as batch gradient descent

In SGD, the error surface is ~~estimated~~ estimated using single training sample. ~~w.r.t single it~~ at each iteration.

$$w_{new} = w_{old} - \eta \frac{\partial E}{\partial w_{old}}$$

where, $E = \frac{1}{2}(t - y)^2$

But the ~~measure~~ major pitfall of SGD is looking at the error incurred one example at a time may not be a good enough approximation of the error surface.

Minibatch :- In minibatch GD, we compute the error surface w.r.t some subset of the total dataset. This subset is called minibatch.

like {16, 32, 64, 128, ...} → batch size.

$$w_{new} = w_{old} - \eta \cdot \frac{\partial E}{\partial w_{old}}$$

$$E = \frac{1}{2} \sum_{i=1}^{k} (t_i - y_i)^2$$

where $k < n$

$n \rightarrow$ no. of training sample

training (80%) . testing (20%)

overfitting :- if it passed training but failed in testing eg:- deep learning

underfitting :- if failed in training only.

## → Gradient Descent with Sigmoidal Neurons

Logistic neurons computes their output values from their i/ps.

First neurons computes the weighted sum of its i/p. Then it feeds its logit into its i/p f^n to compute the final o/p y.

$$z = \sum_k w_k x_k \quad , \quad \frac{\partial z}{\partial w_k} = x_k$$

$$y = \frac{1}{1+e^{-z}} \qquad \frac{\partial z}{\partial x_k} = w_k$$

For learning we have to compute the gradient of the error f^n w.r.t weights.
To do so we have to find the derivate of the logit w.r.t i/p and weights.

The derivate of the o/p w.r.t logit is given by

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

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

$$= \frac{e^{-z}}{(1+e^{-z})^2} \quad \text{or} \quad y(1-y)$$

Now we have to use the chain rule to get the derivate of the o/p w.r.t each weight.

$$\frac{\partial y}{\partial w_k} = \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w_k} = \frac{e^{-z}}{(1+e^{-z})^2} \times x_k$$

$$= y(1-y)x_k$$

Putting all these together, we can now compute the derivative of error f^n w.r.t each weight.

$$\frac{\partial E}{\partial w_k} = \sum_i \frac{\partial E}{\partial y^{(i)}} \cdot \frac{\partial y^{(i)}}{\partial w_k}$$

$$= -\sum_i x_k^{(i)} \cdot y^{(i)}(1-y^{(i)})(t^{(i)} - y^{(i)})$$

$$\boxed{\Delta w_k = \sum_i \eta_i x_k^{(i)} y^{(i)}(1-y^{(i)})(t^{(i)} - y^{(i)})}$$

$$\left\{ \because \Delta w = -\eta \frac{\partial E}{\partial w} \right.$$

Q Compute the derivative of sigmoid fn $f(x) = \frac{1}{1+e^{-x}}$ at $x = 0$

$$\left\{ f'(x) = f(x)(1-f(x)) \right.$$

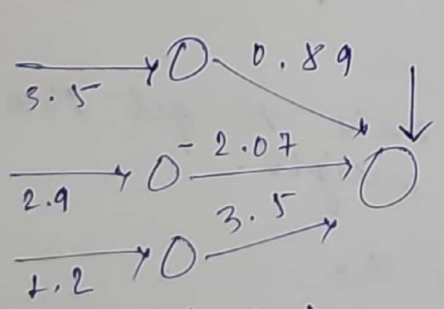
Soln

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

$$\frac{dy}{dx} = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{\frac{1}{e^0}}{\left(1+\frac{1}{e^0}\right)^2} = \frac{1}{2(1+1)^2} = \frac{1}{4}$$

Soln




(i) sigmoid (z)

$$Z = w_1 x_1 + w_2 x_2 + w_3 x_3 + 0.5$$

$$= 3.5 * 0.89 + 2.9 * -2.07 + 1.2 * 3.5 + 0.5$$

$$= 3.115 + (-6.003) + 4.2 + 0.5$$

$$= 3.115 - 6.003 + 4.2 + 0.5$$

$$= 7.315 - 6.003 + 0.5$$

$$= 1.312 + 0.5 = \boxed{1.812}$$

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

$$y = \frac{1}{1+e^{-1.812}}$$

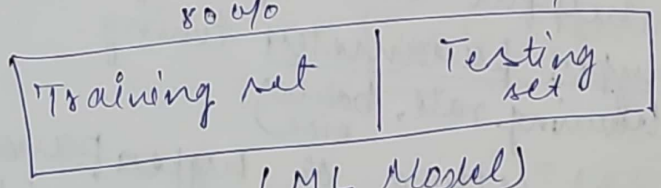
(ii) **tanh**

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

(iii) $\boxed{ReLU(z) = \max(0, z)}$

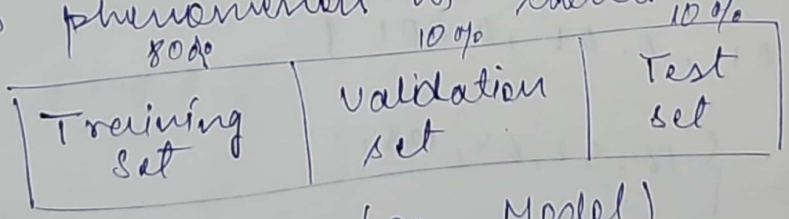
$= \max(0, 1.812) = \underline{\underline{1.812}}$

→ Test set, validation set and overfitting




(ML Model)

Sometimes the model performs well with the training data but does not perform well with the test data. This phenomenon is called Model Overfitting.




(DL Model)

29/10/25

In ML we split the data into training set and test set. In DL we include a validation set to prevent overfitting during the training process.

In DL we have to stop the training process as soon as model start overfitting to prevent poor generalization

To do this, we divide our training process into epochs

Epochs: An epochs is a single iteration over the entire training set.

eg:- Suppose we have a training set of size 'd' and we are doing minibatch GD with batch size 'b', then an epochs would be ~~equal to~~ equivalent to $d/b$.

At the end of each epoch, we have to measure how well our model is generalised. To do this we use an additional validation set. The validation set will tell us how the model does ~~on~~ unseen data.

→ If the accuracy on the training set continously ~~The~~ while the accuracy on the validation set remains the same or ↓es, the training has to stop because the model is overfitting.

The validation set is helpful as a proxy measure of accuracy during hyper parameter tuning (learning rate, batch size)

To find the optimal setting of hyper parameters grid search is applied. Where we pick a value for each hyper parameter from a finite set of ~~values~~ options.

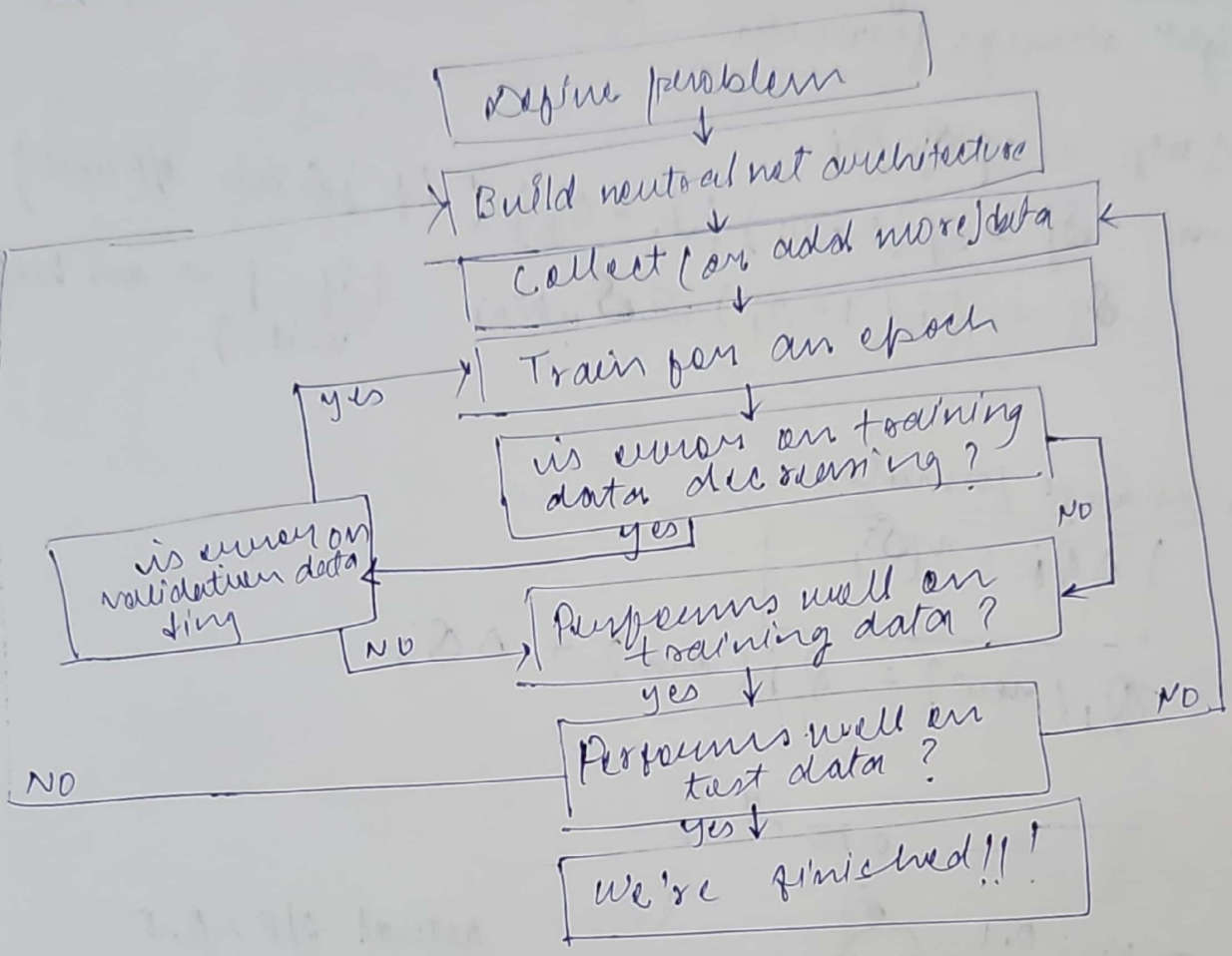
eg $\eta$ (learning rate) $\in \{0.01, 0.01, 0.001\}$

batch size(k) $\in \{16, 32, 64, 128, \ldots\}$

Then we train the model with every possible permutation of hyper parameter choices.

Then we elect the combination of hyper parameters with the best performance validation set and report the accuracy of the model trained with best combination on the test set.

→ Detailed workflow of Training & Validating a DL Model




→ Preventing overfitting in Deep Neural Network:

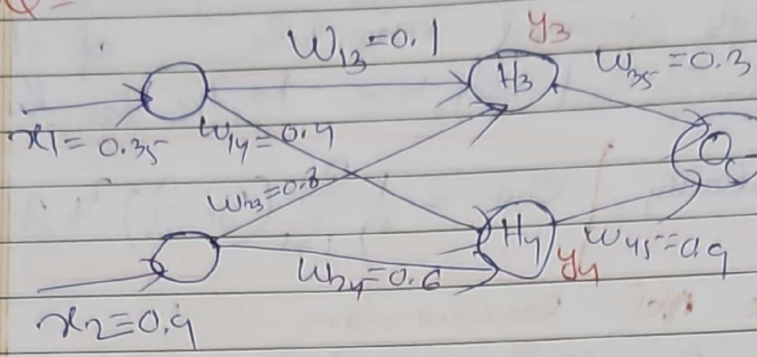
① Regularization → L1 Regularization (Lasso regularization)
→ L2 Regularization (Ridge ")

② Drop out

.

## Back propagation Algorithm :-

(1) Q-




Assume actual op $y = 0.5$

Learning rate $(\eta) = 1$

### Forward Pass (Compute op for $y_3$, $y_4$ and $y_5$)

$H_3 = x_1 * w_{13} + x_2 * w_{23}$

$= (0.35 * 0.1) + (0.9 * 0.8) = 0.755$

$y_3 = \frac{1}{1+e^{-H_3}} = \frac{1}{1+e^{-0.755}} = 0.68$

$H_4 = x_1 * w_{14} + x_2 * w_{24}$

$= 0.35 * 0.4 + 0.9 * 0.6 = 0.68$

$y_4 = \frac{1}{1+e^{-H_4}} = \frac{1}{1+e^{-0.68}} = 0.6637$

Now, $O_5 = y_3 * w_{35} + y_4 * w_{45}$

$= 0.68 * 0.3 + 0.6637 * 0.9$

$= 0.801$

$y_5 = \frac{1}{1+e^{-O_5}} = \frac{1}{1+e^{-0.801}} = 0.69$

Error $= y_{target} - y_{actual}$

$= 0.5 - 0.69$

$= -0.19$

## Weight change Formula

$\delta_j$: Error measure for unit j

$$\Delta w_{ji} = \eta \delta_j O_i$$

where

$$\delta_j = O_j(1-O_j)(t_j - O_j) \quad \text{(if } j \text{ is an o/p unit)}$$

$$\delta_j = O_j(1-O_j)\sum \delta_k w_{kj} \quad \text{(if } j \text{ is a hidden unit)}$$

## Backward Pass :

For o/p unit :

$$\delta_5 = y_5(1-y_5)(y_{target} - y_5)$$
$$= 0.69(1-0.69)(0.5-0.69) = -0.0406$$

For Hidden unit :

$$\delta_3 = y_3(1-y_3) w_{35} * \delta_5$$
$$= 0.68(1-0.68) * 0.3 * (-0.0406)$$
$$= -0.00265$$

$$\delta_4 = y_4(1-y_4) w_{45} * \delta_5$$
$$= 0.6637(1-0.6637) * 0.9 * -0.0406$$
$$= -0.0082$$

## Compute new weights

$$\Delta w_{ji} = \eta \delta_j O_i$$

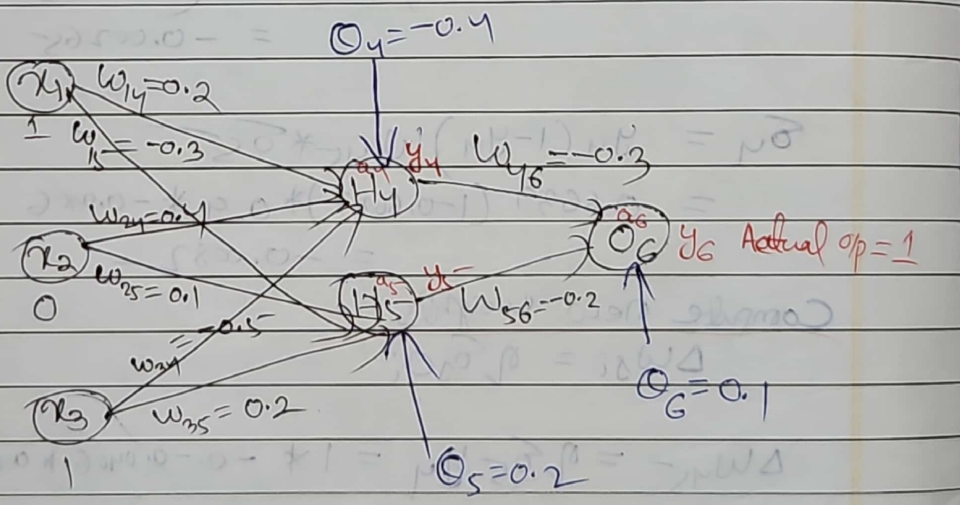
$$\Delta w_{45} = \eta \delta_5 y_4 = 1 * -0.-0.0406 * 0.6637 = -0.0269$$
$$w_{45}(new) = \Delta w_{45} + w_{45}(old) = -0.0269 + 0.9 = 0.8731$$

$$\Delta w_{14} = \eta \delta_4 x_4 = 1 * -0.0082 * 0.35 = -0.00287$$
$$w_{14}(new) = w_{14}(old) + \Delta w_{14} = 0.4 - 0.00287$$
$$= 0.3971$$

Update all other weights

| i | j | $w_{ij}$ | $\delta_j$ | $x_i$ | $\eta$ | Updated $w_{ij}$ |
|---|---|---|---|---|---|---|
| 1 | 3 | 0.1 | -0.00265 | 0.35 | 1 | 0.0991 |
| 2 | 3 | 0.8 | -0.00265 | 0.9 | 1 | 0.7976 |
| 1 | 4 | 0.4 | -0.0082 | 0.35 | 1 | 0.3971 |
| 2 | 4 | 0.6 | -0.0082 | 0.9 | 1 | 0.5926 |
| 3 | 5 | 0.3 | -0.0406 | 0.68 | 1 | 0.2724 |
| 4 | 5 | 0.9 | -0.0406 | 0.6637 | 1 | 0.8731 |

(2) Q. Assume the neurons have a sigmoid activation function, perform a forward pass and a backward pass on the network.
Assume that the actual o/p of y is 1 and learning rate is 0.9. Perform another forward pass.




Forward Pass

Compute o/p for $y_4, y_5, y_6$

$$a_4 = w_{14} * x_1 + w_{24} * x_2 + w_{34} * x_3 + \Theta_4$$
$$= (0.2 * 1) + (0.4 * 0) + (-0.5 * 1) + (-0.4)$$
$$= -0.7$$

$O(H_4) = y_4 = f(a_4) = \frac{1}{1+e^{-a_4}} = \frac{1}{1+e^{-0.7}} = 0.332$

$a_5 = w_{15} * x_1 + w_{25} * x_2 + w_{35} * x_3 + \theta_5$

$= (-0.3 * 1) + (0.1 * 0) + (0.2 * 1) + 0.2 = 0.1$

$O(H_5) = y_5 = f(a_5) = \frac{1}{1+e^{-0.1}} = 0.525$

Now, $a_6 = w_{46} * H_4 + w_{56} * H_5 + \theta_6$

$= (-0.3 * 0.332) + (-0.2 * 0.525) + 0.1$

$= -0.105$

$O(O_6) = y_6 = f(a_6) = \frac{1}{1+e^{0.105}} = 0.474$

Error $= y_{target} - y_6 = 1 - 0.474 = 0.526$

Back ward Pass

Compute $\delta_4$, $\delta_5$ and $\delta_6$

For o/p unit:

$\delta_6 = y_6(1-y_6)(y_{target} - y_6)$

$= 0.474(1-0.474)(1-0.0474) = 0.1311$

For Hidden unit:

$\delta_5 = y_5(1-y_5) w_{56} * \delta_6$

$= 0.525 * (1-0.525) * (-0.2 * 0.1311)$

$= -0.0065$

$\delta_4 = y_4(1-y_4) w_{46} * \delta_6$

$= 0.332 (1-0.332)(-0.3 * 0.1311)$

$= -0.0087$

$\Delta w_{46} = \eta \cdot \delta_6 y_4 = 0.9 * 0.1311 * 0.332 = 0.03917$

$w_{46}(new) = w_{46}(old) + \Delta w_{46} = -0.3 + 0.03917$

$= -0.261$

$\Delta w_{14} = \eta \delta_4 x_1 = 0.9 * -0.0087 * 1 = -0.0078$

$w_{14}(new) = \Delta w_{14} + w_{14}(old) = -0.0078 + 0.2 = 0.192$

Similarly, update all other weights.

| i | j | $w_{ij}$ | $\delta_j$ | $x_i$ | $\eta$ | updated $w_{ij}$ |
|---|---|---|---|---|---|---|
| 4 | 6 | -0.3 | 0.1311 | 0.332 | 0.9 | -0.261 |
| 5 | 6 | -0.2 | 0.1311 | 0.525 | 0.9 | -0.138 |
| 1 | 4 | 0.2 | -0.0087 | 1 | 0.9 | 0.192 |
| 1 | 5 | -0.3 | -0.0065 | 1 | 0.9 | -0.306 |
| 2 | 4 | 0.4 | -0.0087 | 0 | 0.9 | 0.4 |
| 2 | 5 | 0.1 | -0.0065 | 0 | 0.9 | 0.1 |
| 3 | 4 | -0.5 | -0.0087 | 1 | 0.9 | -0.508 |
| 3 | 5 | 0.2 | -0.0065 | 1 | 0.9 | 0.194 |

Similarly, update the biases

| $\theta_j$ | Prev. $\theta_j$ | $\delta_j$ | $\eta$ | updated $\theta_j$ |
|---|---|---|---|---|
| $\theta_6$ | 0.1 | 0.1311 | 0.9 | 0.218 |
| $\theta_5$ | 0.2 | -0.0065 | 0.9 | 0.194 |
| $\theta_4$ | -0.4 | -0.0087 | 0.9 | -0.408 |

$$\Delta\theta_j = \eta \cdot \delta_j$$

$$\theta_j(new) = \Delta\theta_j + \theta_j(old)$$

Forward pass:

Compute op for $y_4$, $y_5$ and $y_6$.

$$a_4 = (w_{14} * x_1) + (w_{24} * x_2) + (w_{34} * x_3) + \theta_4$$
$$= (0.192 * 1) + (0 * 0) + (-0.508 * 1) + (-0.408)$$
$$= -0.724$$

$$y_4 = f(a_4) = \frac{1}{1+e^{0.734}} = 0.327$$

$$a_5 = (w_{15} * x_1) + (w_{25} * x_2) + (w_{35} * x_3) + \theta_5$$
$$= (-0306 * 1) + (0.1 * 0) + (0.191 * 1) + 0.194$$
$$= 0.082$$

$$y_5 = f(a_5) = \frac{1}{1+e^{-0.082}} = 0.520$$

$$a_6 = w_{46} * H_4 + w_{56} * H_6 + \theta_6$$
$$= -0.261 * 0.327 + (-0.138 * 0.520) + 0.218$$
$$= 0.061$$

$$y_6 = f(a_6) = \frac{1}{1+e^{-0.061}} = 0.515$$

$$\text{Error} = y_{target} - y_6 = 0.485$$

# Chapter - 6

Date
Page No.

## Challenges with Gradient Descent:

* Vanishing gradient
* Saddle point

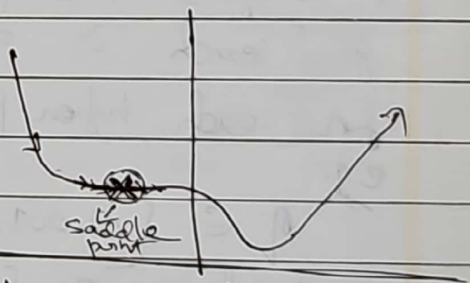
### ⇒ Vanishing gradient

→ When training deep NN with gradient descent and back propagation, we have to calculate the partial derivatives by moving from the final o/p layers to the initial layer.
→ Due to this process, vanishing gradient problem arises.

### ⇒ Saddle point:

⇒ It is a point on the surface of the graph of a function where the slopes (derivatives) in orthogonal directions are all zero. but which is not a local extremum of the function.

→ Saddle point injects confusion into the learning process.




→ Learning of the model becomes slow.

→ This saddle point gets into focus when the gradient descent (GD) works in multi dimensions (i.e., we have more & more parameters)

### Avoiding vanishing gradient

→ Use ReLU or Leaky ReLU as activation function
→ Reduce the model complexity (Reducing no. of layers)
→ Better optimizer with a well-tuned learning rate

## Momentum based Gradient optimization

→ It is an extension of Batch GD.
→ Momentum is a strategy for accelerating the convergence of the optimization process by including a momentum element in the update rule.
→ Momentum considers previous step weights gradient changes when updating current weights.
→ This momentum factor assists the optimizer in continuing to go in the same direction of the search space even if the gradient changes direction or becomes zero.
→ It overcome the oscillations of noisy gradients and coast across flat stats of the search space.
→ Instead of stopping when the gradient is 0 at the 1st local minimum, momentum will continue to move forward because it considers the prior steep in the slope.

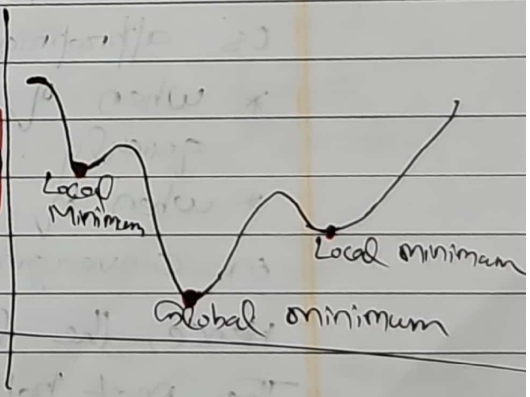
Mathematically,

$$V(t) = \beta * V(t-1) + (1-\beta) \cdot \nabla(w)$$

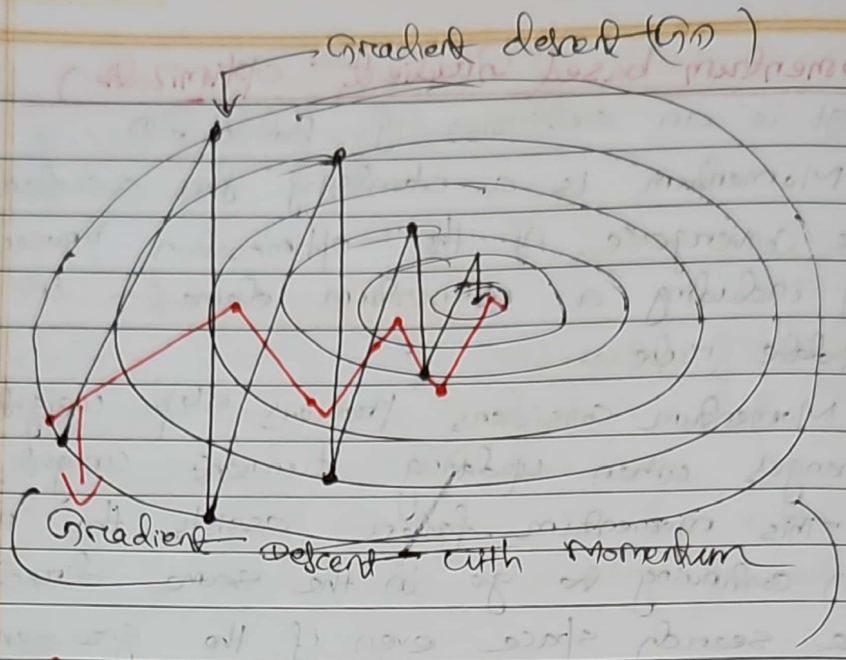
$$\nabla(w) = \frac{\partial E}{\partial w_t}$$

$$w = w - \alpha \cdot V(t)$$

where V : velocity
β : momentum factor (ranges from 0 to 1) (typically as 0.9)
α : learning rate




→ The optimizer calculates the gradient of the cost function at each iteration, and updates the momentum term as the exponentially weighted average of the previous gradients.




## Advantages:

→ It helps the algorithm to escape from local minima and saddle points.
→ It makes the algorithm to converge faster by avoiding oscillations.

## Learning Rate Adaptions

→ The major challenge for training deep NN, is appropriately selecting the learning rate.
* when $\eta$ is too small, it doesn't learn quickly.
* when $\eta$ is too large, it may have difficulty in converging to a local minimum.

Hence, the learning rate adaption is very vital.
The most popular adaptive learning rate algos are
① AdaGrad (Adaptive Gradient) optimization
② RMSProp (Root Mean Squared Propagation) optimization
③ Adam (Adaptive Moment Estimation) optimization