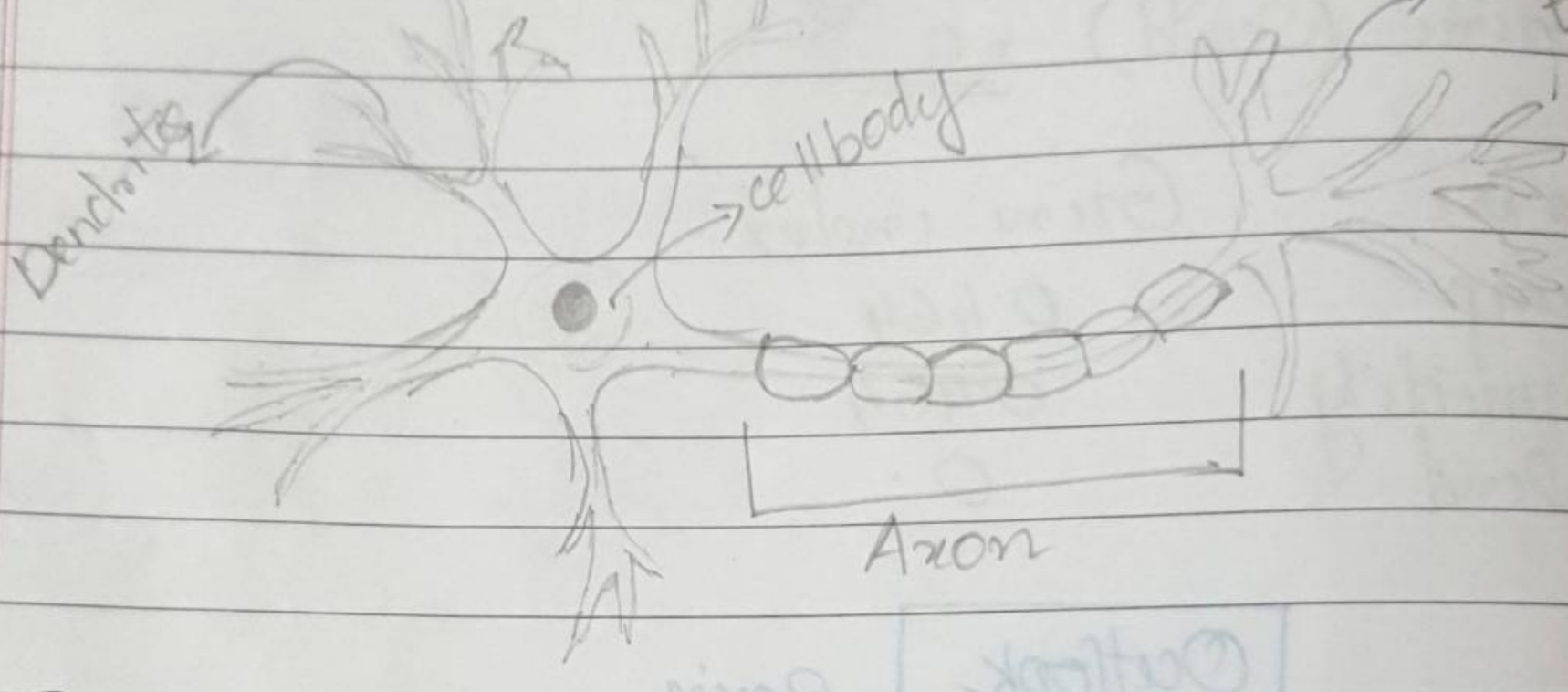
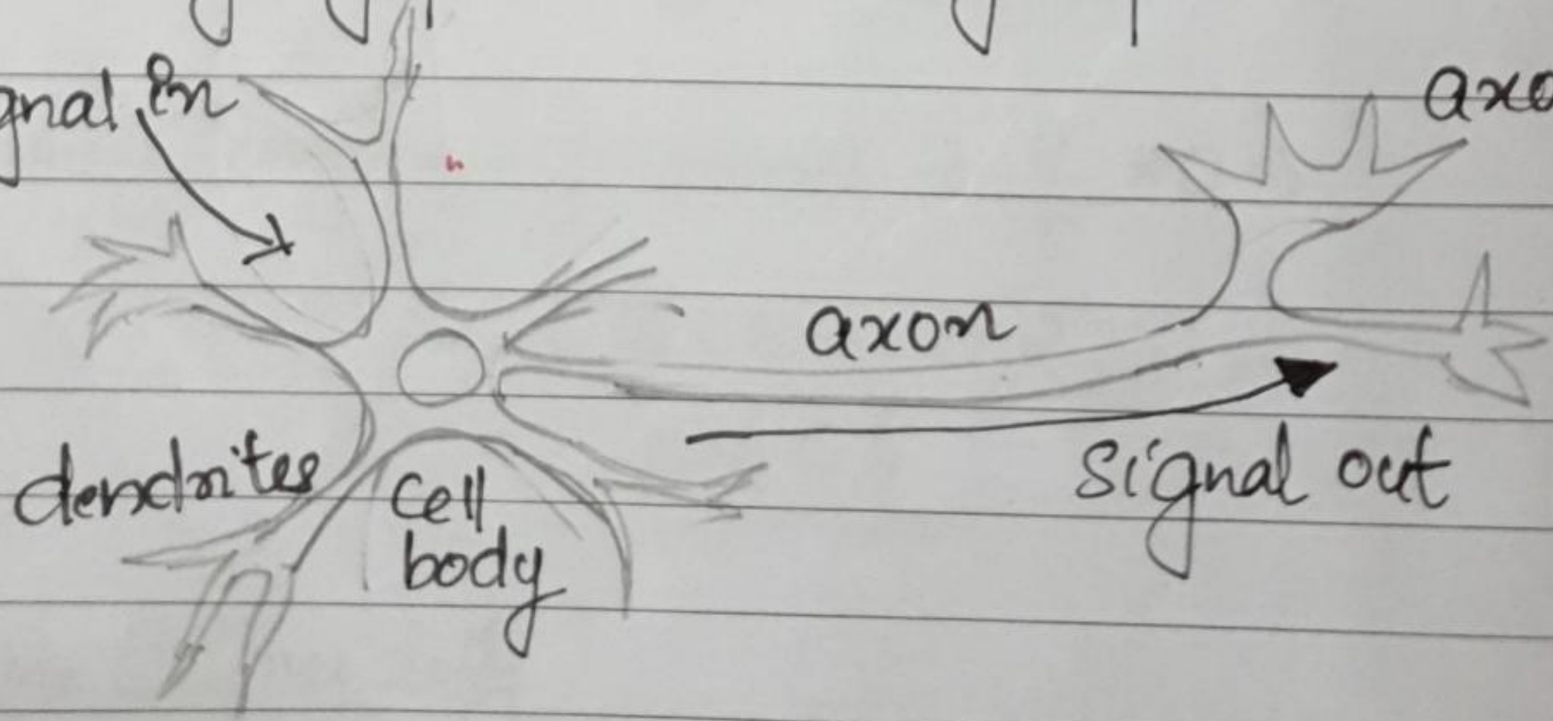
Neural Networks:-

* Motivation behind neural network is human brain. Homain brain contains billion of neurons which are connected to many othee neurone to form a network so that if we sense anything, it recognizes and Processes the output.

*

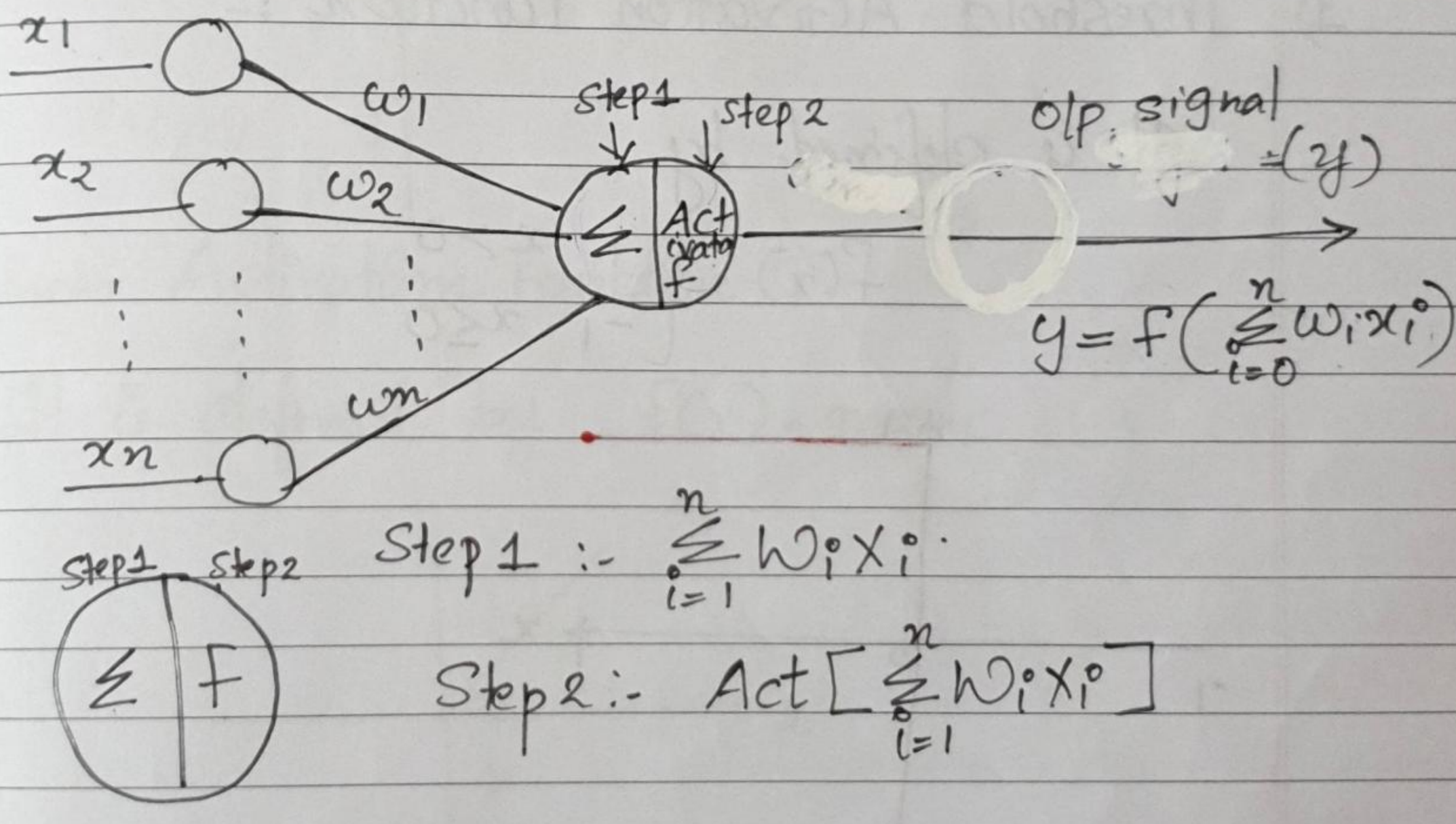


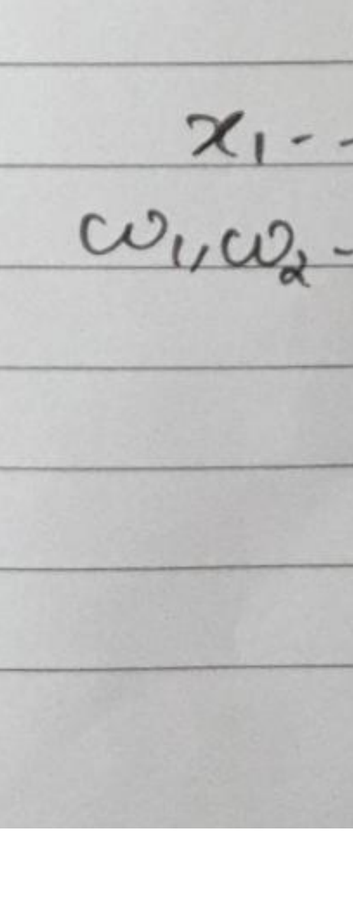
* Dendrites receives signals from other neurons
* Cell body sums the incoming signals to generate input
* When the sum reaches a threshold value, neuron fires and the signal travels down the axon to the other neurons through a tiny gap called Synapse



Artificial Neurons:-

* An artificial neuron is a mathematical function
* Artificial neurons are elementary units in an artificial neural network.
* Artificial neuron receives one or more inputs and sums them to produce an output.
* Each input is seperately weighted, and the sum is passed through a function known as an activation function or transfer function.





Step 1 :- $\sum_{i=1}^{n} w_i x_i$

Step 2 :- $Act\left[\sum_{i=1}^{n} w_i x_i\right]$

$x_1 \dots x_n$ = input signals

$w_1, w_2 \dots, w_n$ = weighted associated with i/p signals

Σ = summation of input signals

f = activation function which produces the output

y = o/p signal

* Circles are the nodes of the neurons
* $x_1, x_2 \ldots x_n$ are the input nodes
* Circle in the second layer outputs the value 'y' is called output node.

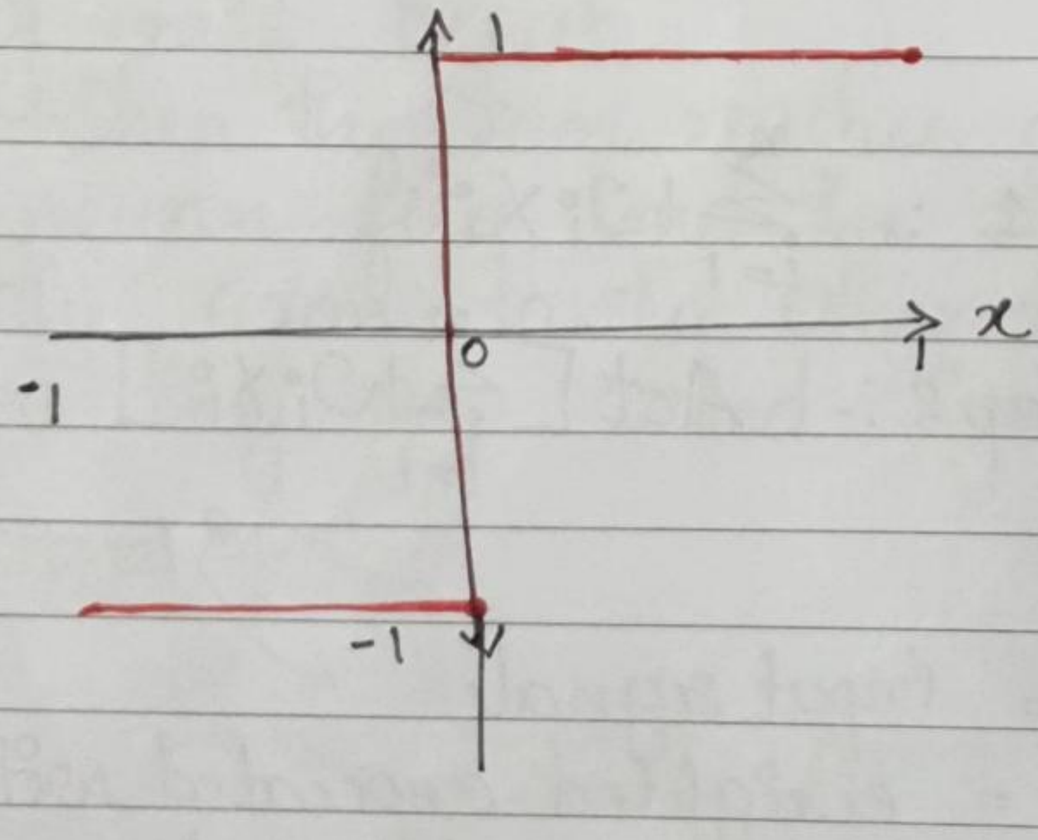
(6) Activation functions :-

* The function which takes the incoming signals as input and produces the output signal is known as activation function

1) Threshold Activation Function :-

It is defined by

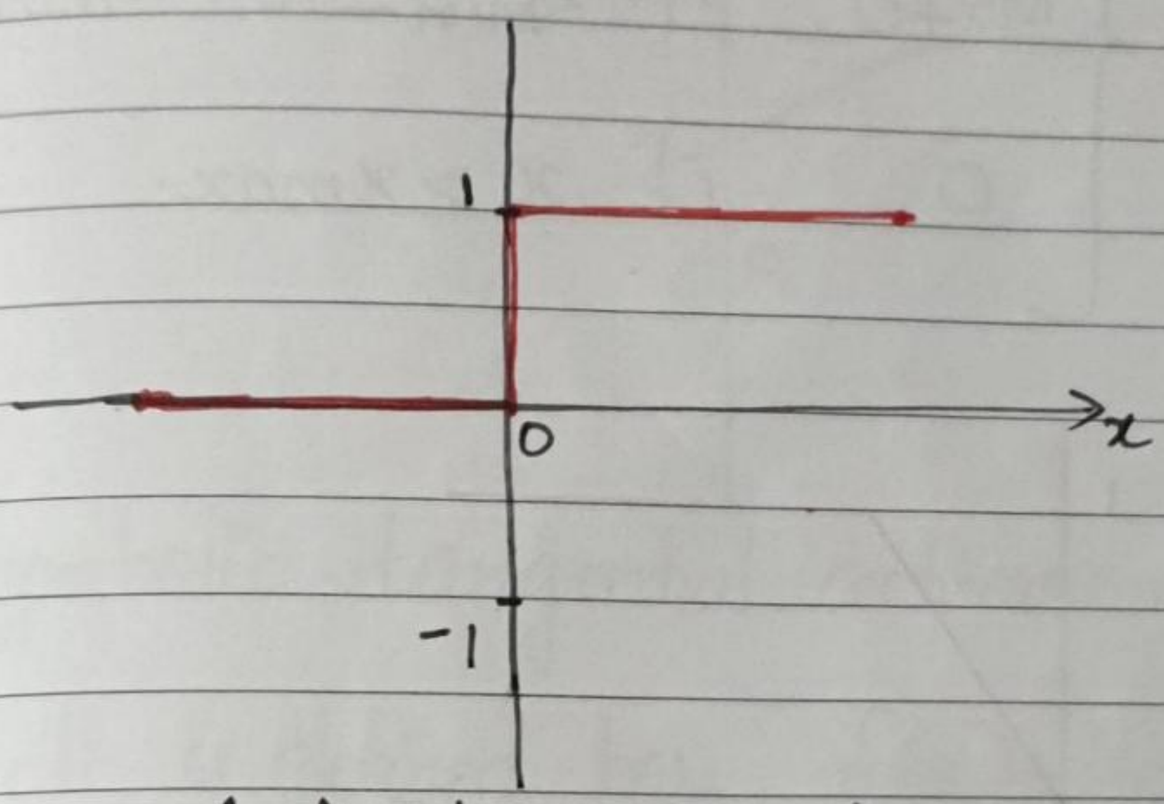
$$f(x) = \begin{cases} 1 & x > 0 \\ -1 & x \leq 0 \end{cases}$$



2) Unit Step Function: Binary Bipolar

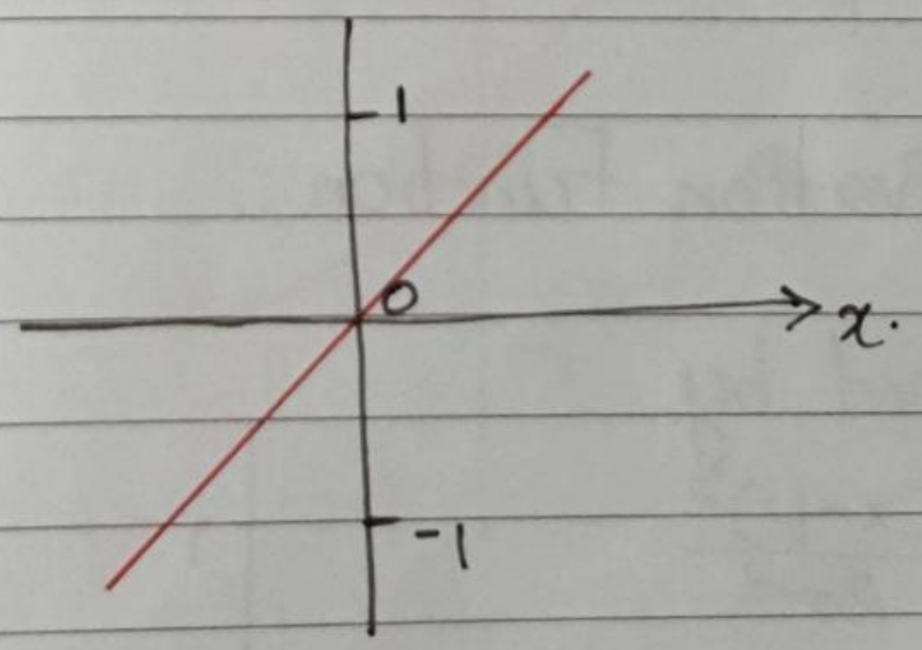
Sometimes the threshold activation function is also defined as a unit step function, in such cases it is called unit step activation function

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



3) Linear Activation Function.

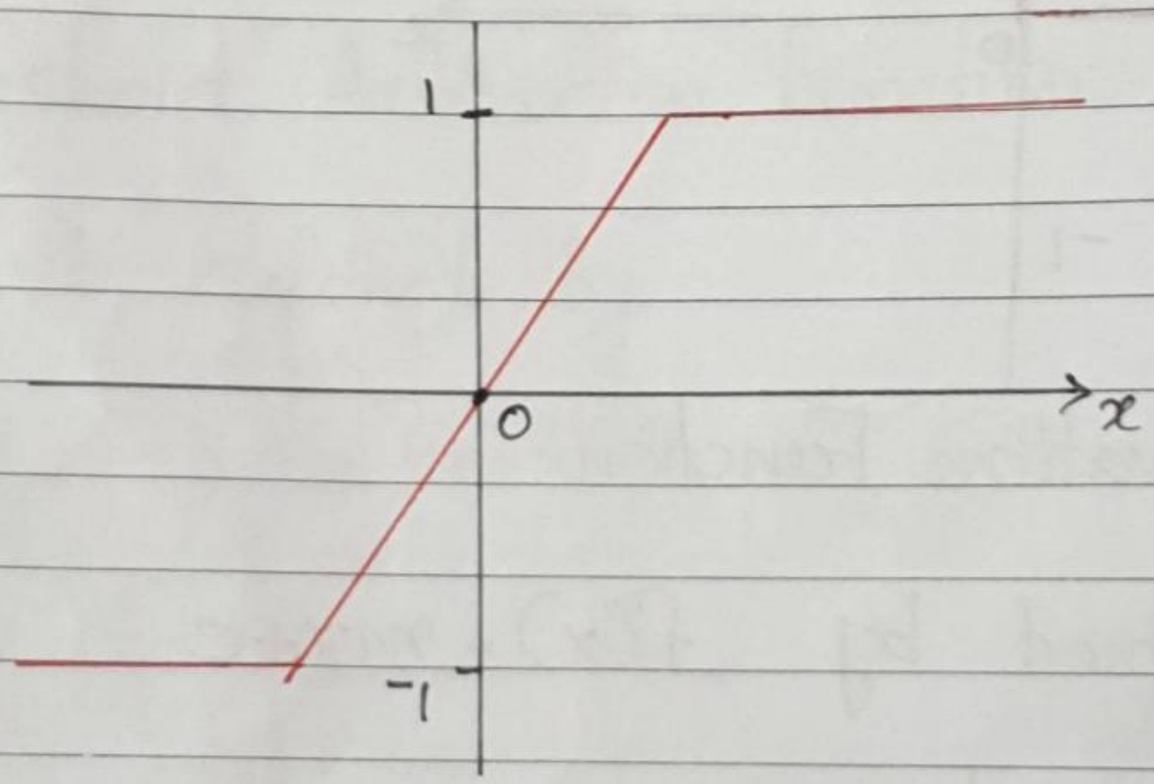
It is defined by $f(x) = mx + c$



4) Piecewise / saturated Linear Activation function

It is defined by

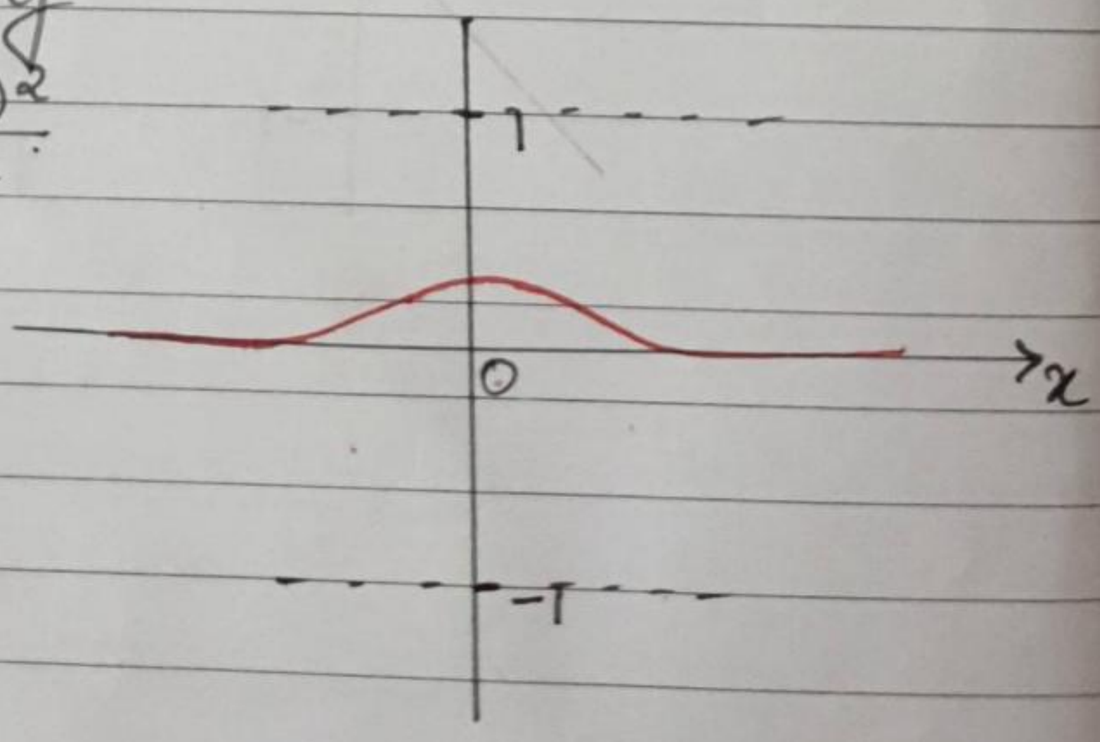
$$f(x) = \begin{cases} 0 & \text{if } x < x_{min} \\ mx+c & \text{if } x_{min} \leq x \leq x_{max} \\ 0. & \text{if } x > x_{max}. \end{cases}$$



5) Gaussian Activation Function :-

It is defined by

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

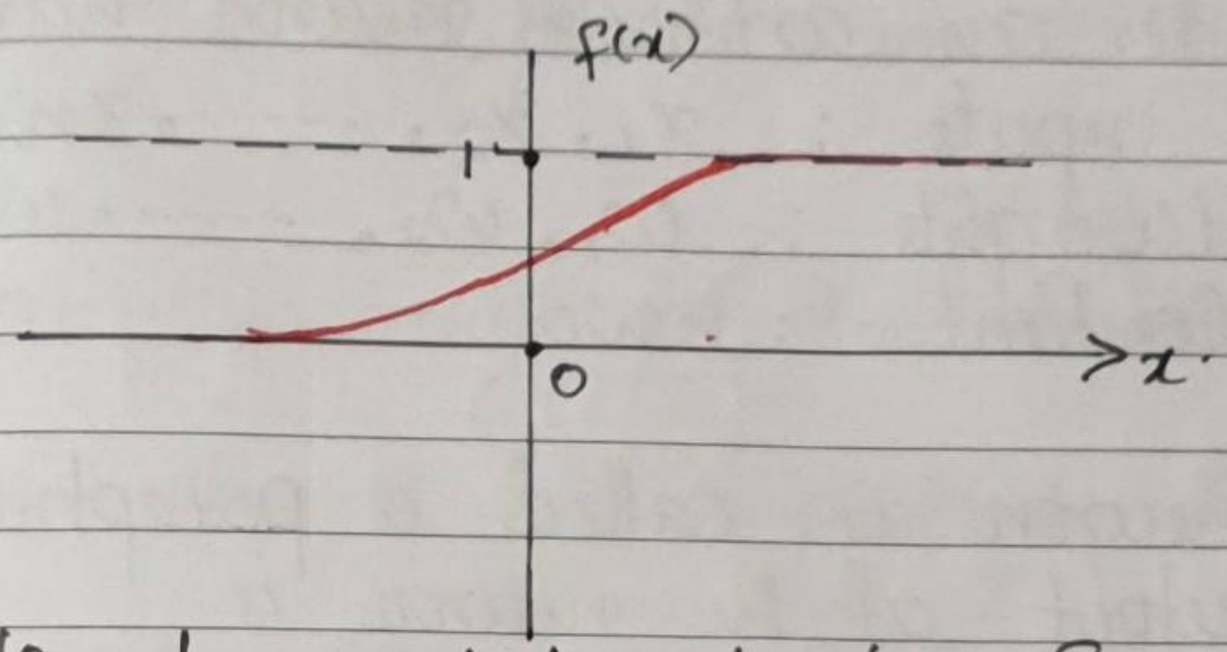


6) Sigmoid activation function [Logistic function]

- Most commonly used activation functions
- It is defined as

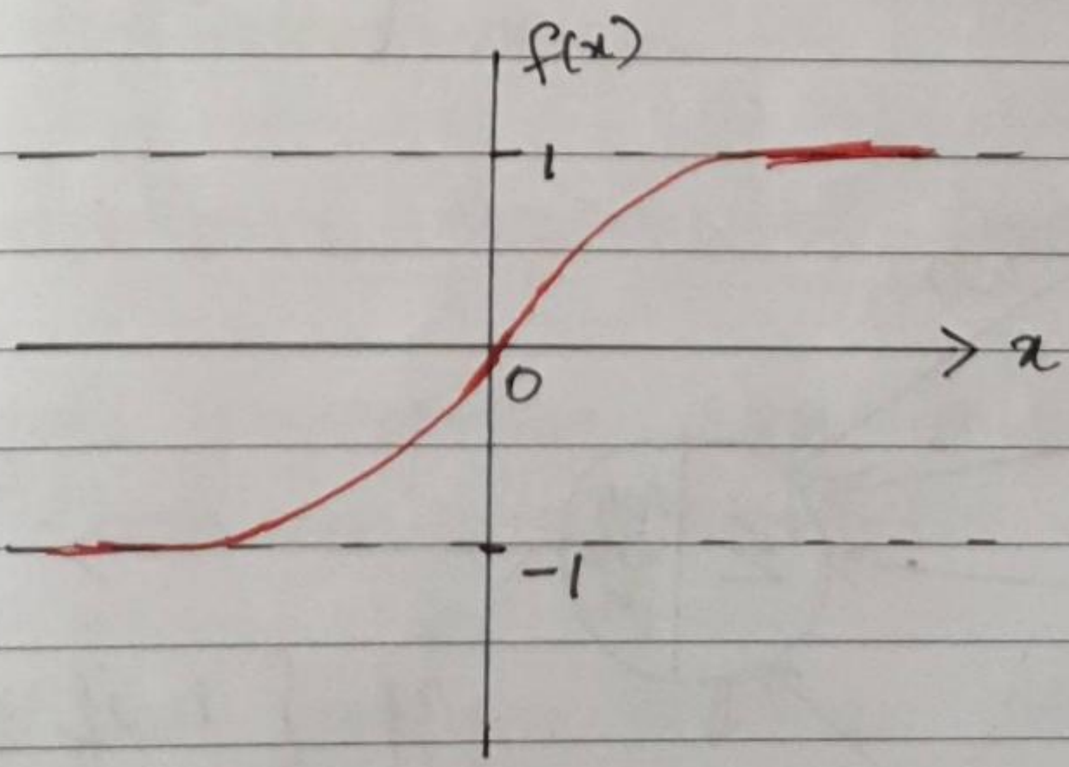
$$f(x) = \frac{1}{1+e^{-x}}$$

Bipolar:
↳ $\frac{2}{1+e^{-x}} - 1$



7) Hyperbolic tangential activation function

- It is defined by $f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$



(15b)

Perceptron | Single-layer N.N.

* The perceptron is a special type of artificial neuron in which the activation function is the threshold function.
* Basic neural network building block
* Consider an artificial neuron having

inputs : $x_1, x_2, \ldots, x_n$

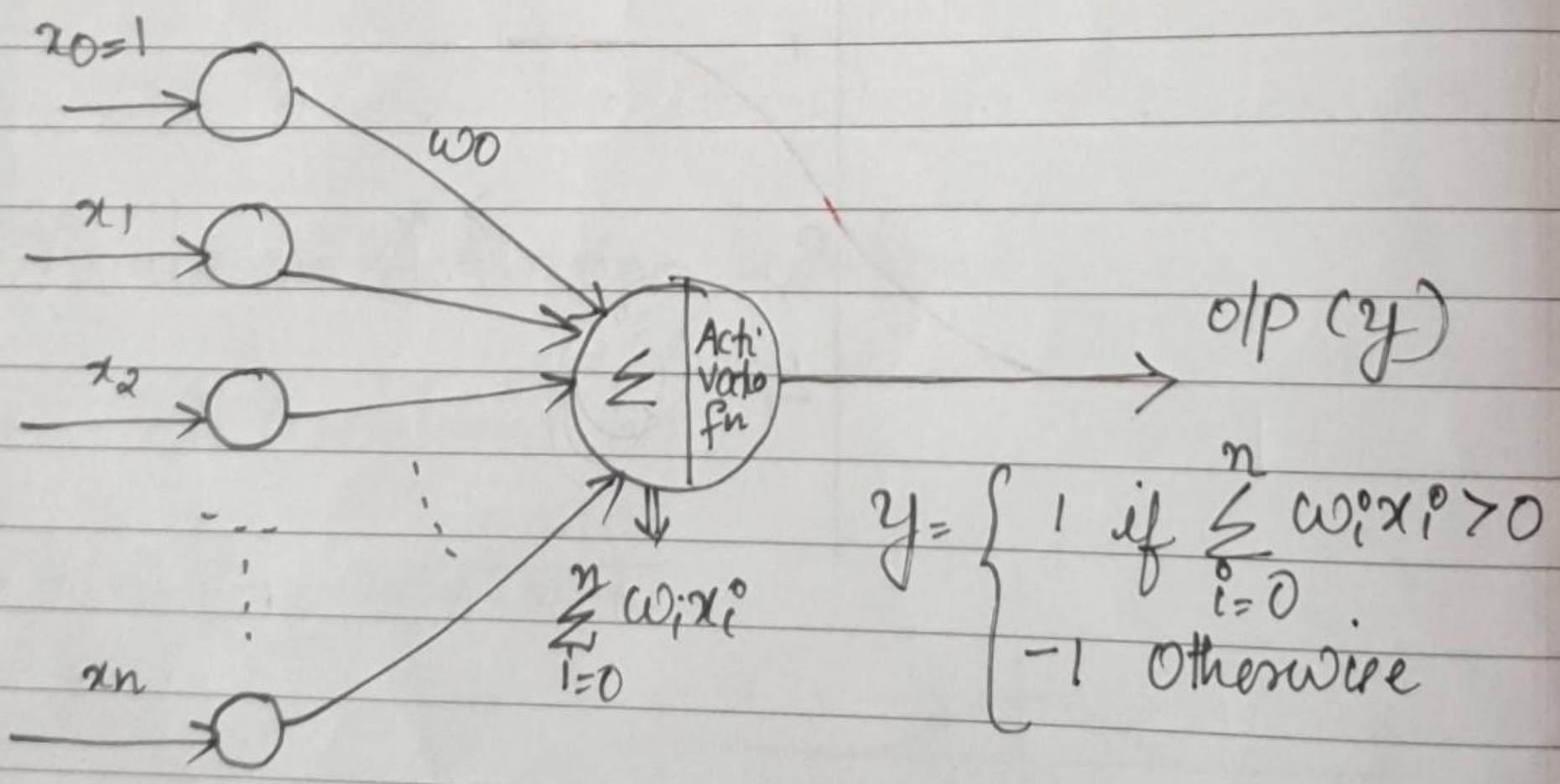
associated weights : $w_1, w_2, \ldots, w_n$

Constant : w_0

The neuron is called a perceptron if the output of the neuron is

Output :-

$$O(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots w_n x_n > 0 \\ -1 & \text{if } w_0 + w_1 x_1 + \cdots w_n x_n \leq 0 \end{cases}$$



$$y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

Working of a perceptron:-

(a) All Inputs 'xi' are multiplied by weights 'wi'

(b) Add all multiplied values and call it as weighted sum

(c) Apply the weighted sum to the activation function

(d) Output the result of the activation function.

weights :- Strength of a particular input

Basic Building blocks of Artificial neural network:-

1) Artificial Neuron
2) Weights
3) Bias → allows to shift the activation function by adding a constant.
4) Activation function

Representational power of perceptrons :-

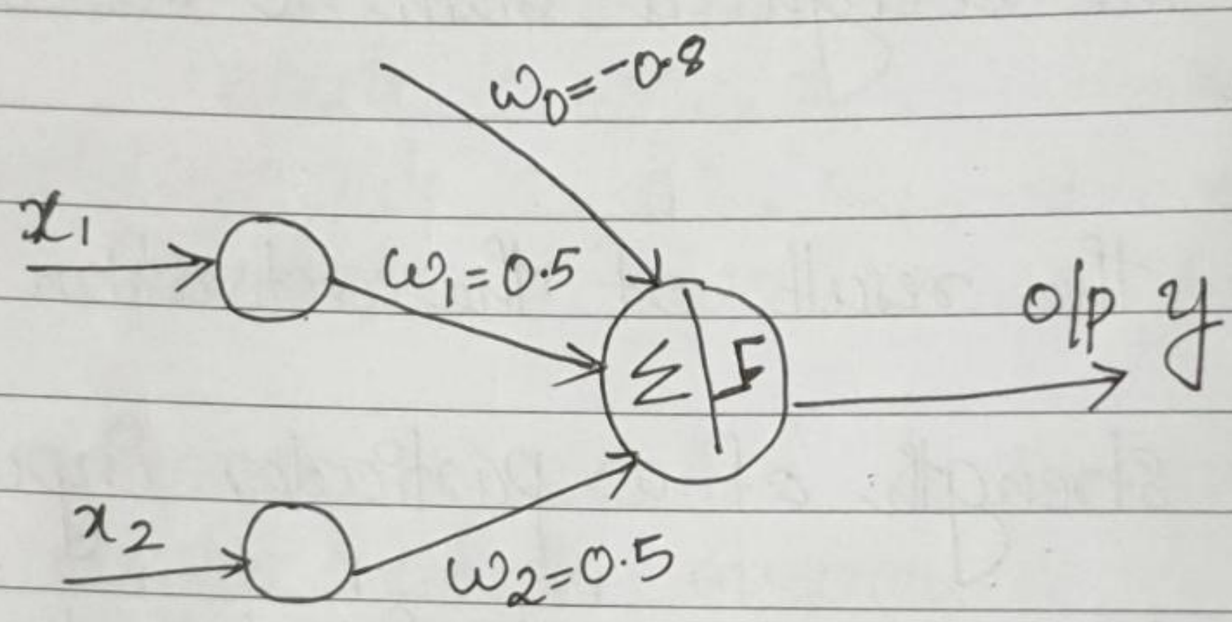
A single perceptron can be used to represent many boolean functions.

eg:- If we assume boolean value of 1 (true) and -1 (false), then one way to represent a '2' input perceptron to implement AND function is to set weights.

$w_0 = -0.8$; $w_1 = w_2 = 0.5$

AND Truth table

A	B	T	Perceptron value
→ 0	0	0	-1 (false)
1	0	0	-1 (false)
0	1	0	-1 (false)
1	1	1	1 (True)



(x_1, x_2)

$(0, 0)$ $\quad w_0 + 0.5\times0 + 0.5\times0 =$

$-0.8 + 0.5\times0 + 0.5\times0 = -0.8 < 0 \Longrightarrow -1$

(x_1, x_2)

$(1\ 0)$ $\quad w_0 + 0.5\times x_1 + 0.5\times x_2$

$-0.8 + 05\times1 + 0.5\times0 = -0.8 + 0.5 = -0.3 < 0 \Longrightarrow -1$

$\begin{pmatrix} x_1, x_2 \\ 0 \quad 1 \end{pmatrix}$ $\quad w_0 + 0.5\times x_1 + 0.5\times x_2$

$-0.8 + 0.5\times0 + 0.5\times1 = 0.8 + 0.5$

$= -0.3 < 0 \Longrightarrow -1$

$\begin{pmatrix} x_1, x_2 \\ 1 \quad 1 \end{pmatrix}$ $\quad w_0 + 0.5x_1 + 0.5x_2$

$-0.8 + 0.5 + 0.5 = 0.2 > 0 \Longrightarrow 1$

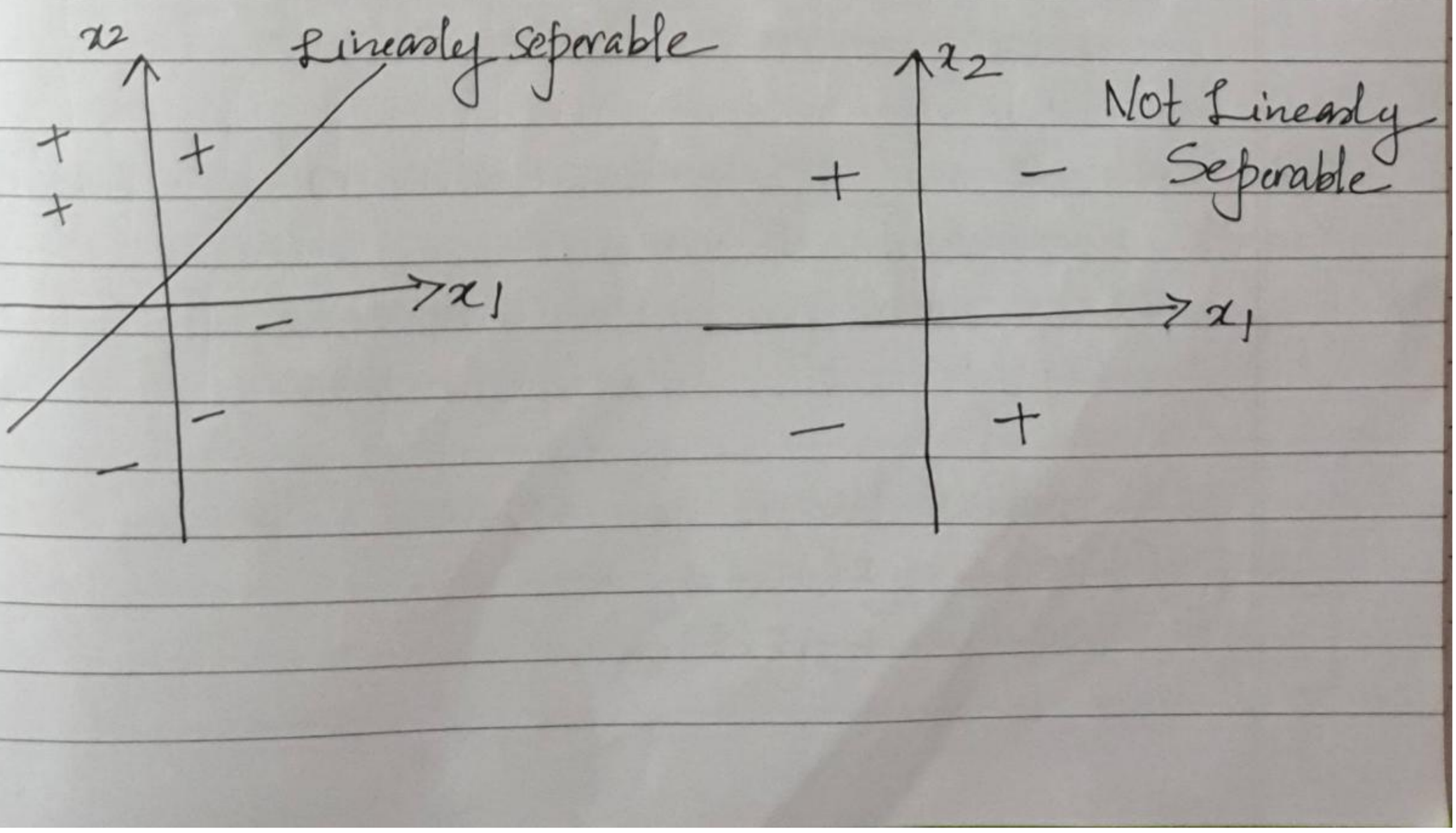
IIIrdly OR function can be represented by modifying $w_0 = -0.3$

- NAND function, $w_0 = 0.8$
- NOR function, $w_0 = 0.3$.

Boolean function	w_0	w_1	w_2
x_1 AND x_2	-0.8	0.5	0.5
x_1 OR x_2	-0.3	0.5	0.5
x_1 NAND x_2	0.8	-0.5	-0.5
x_1 NOR x_2	0.3	-0.5	-0.5

- ∴ Perceptrons can represent all primitive boolean functions lcke AND, OR, NAND, and NOR.

- However XOR requires more than one single perceptron whose o/p is 1 if and only if $x_1 \neq x_2$. It represents the set of linearly non seperable training examples.



Perceptron Learning Algorithm

P ← inputs with label 1;
N ← Pnputs with label 0;
Initialize $w = [w_1, w_2, \ldots, w_n]$ randomly

while ! Convergence do
    pick random $x \in P \cup N$;
    if $x \in P$ and $\sum_{i=0}^{n} w_i * x_i < 0$ then
        $w = w + x$;
    end
    if $x \in N$ and $\sum_{i=0}^{n} w_i * x_i \geq 0$ then
        $w = w - x$;
    end
end

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

w = n+1 dimen
x = n dimen

Perceptron learning Algorithm

n : No: of input variables.
y : o/p from the perceptron
D : Training set of 's' samples.
$x=(x_{j0}, x_{j1}, \cdots, x_{jn})$: n-dimensional input
d_j : Desired o/p value of the perceptron for the input x_j
x_{ji} : value of the i-th feature of the j-th training input
x_{j0} : 1
w_i : weight of the i-th input variable
$w_i(t)$: weight i at the 't-th iteration.

Step 1 : Initialize the weights and the threshold. Weights may be initialized to '0' or a to a small random value.

Step 2 : For each example 'j' in the training set D, perform the following steps over the input x_j and desired output d_j.

: a) Calculate the actual output

$$y_j(t) = f[w_0(t)x_{j0} + w_1(t)x_{j1} + w_2(t)x_{j2} + \cdots + w_n(t)x_{jn}]$$

b) Update the weights:

$$w_i(t+1) = w_i(t) + (d_j - y_j(t))x_{ji}$$

for all features $0 \leq i \leq n$.

Step 3: Step 2 is repeated until the iteration error $\frac{1}{s}\sum_{j=1}^{s} |d_j - y_j(t)|$ is less than a user specified error threshold or a predetermined no: of iterations have been completed.

s = size of the sample set

Artificial Neural Network:

* ANN is based on a collection of connected units called artificial neurons

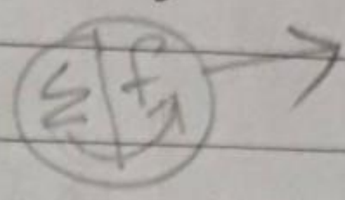
* Each connection between artificial neurons can transmit a signal from one to another

* The artificial neuron that receives the signal can process it and then signal artificial neurons connected to it.

Characteristics of ANN

1) The activation function

* This function defines how a neuron's combined input signals are transformed into a single output signal



2) The network topology

* describes the number of neurons in the model as well as the number of layers and the way they are connected

* It determines the complexity of the task

Different forms of topology can be differentiated by the following characteristics

a) The no: of layers :-

nodes are arranged in layers.

- Input nodes :- (first layer) Those nodes which receive unprocessed signals directly from the input data.
- Output nodes :- Those nodes which generate the final predicted values.
- Hidden nodes : (Second layer) a node that processes the signals from the i/p nodes (or other such nodes) prior to reaching output nodes

b) whether information in the network is allowed to travel backward

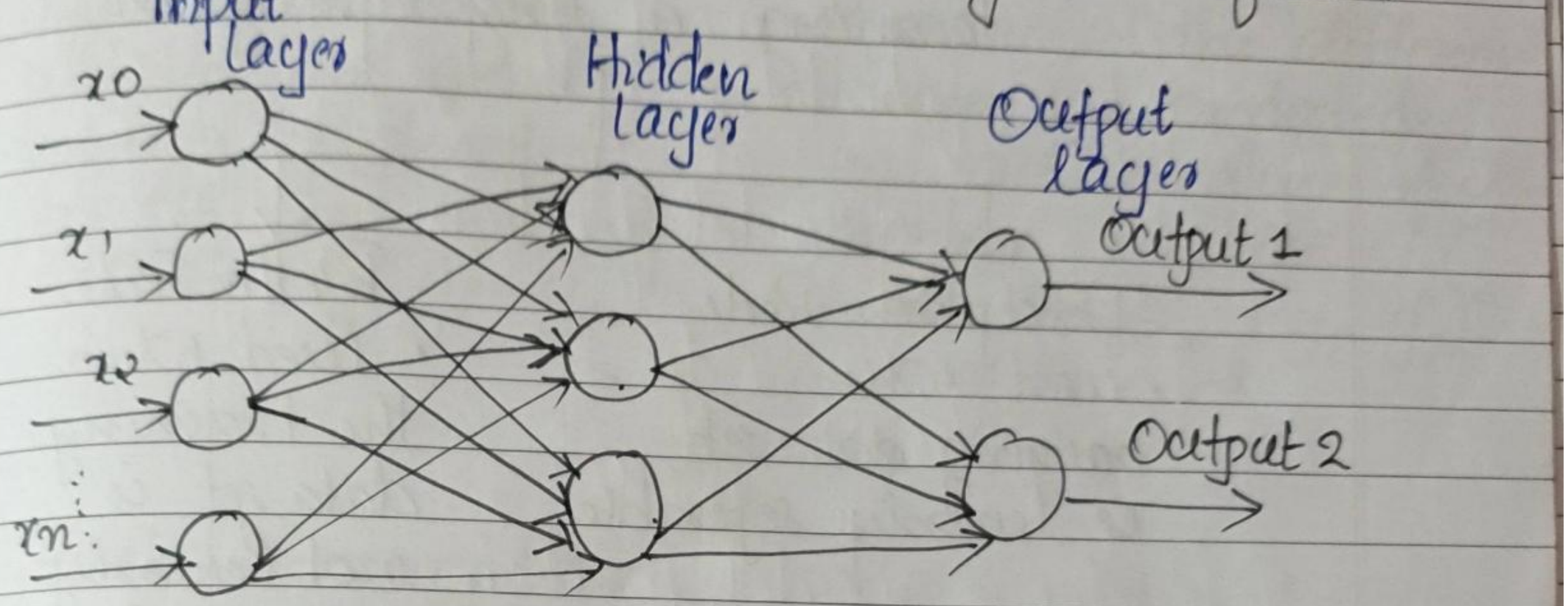
Feedforward networks:-
Networks in which the input signal is fed continously in one direction from connection to connection until it reaches the output layer called feed forward networks.

Feedback networks.
Networks which allows signals to travel in both directions using loops are called feedback networks / Recurrent networks

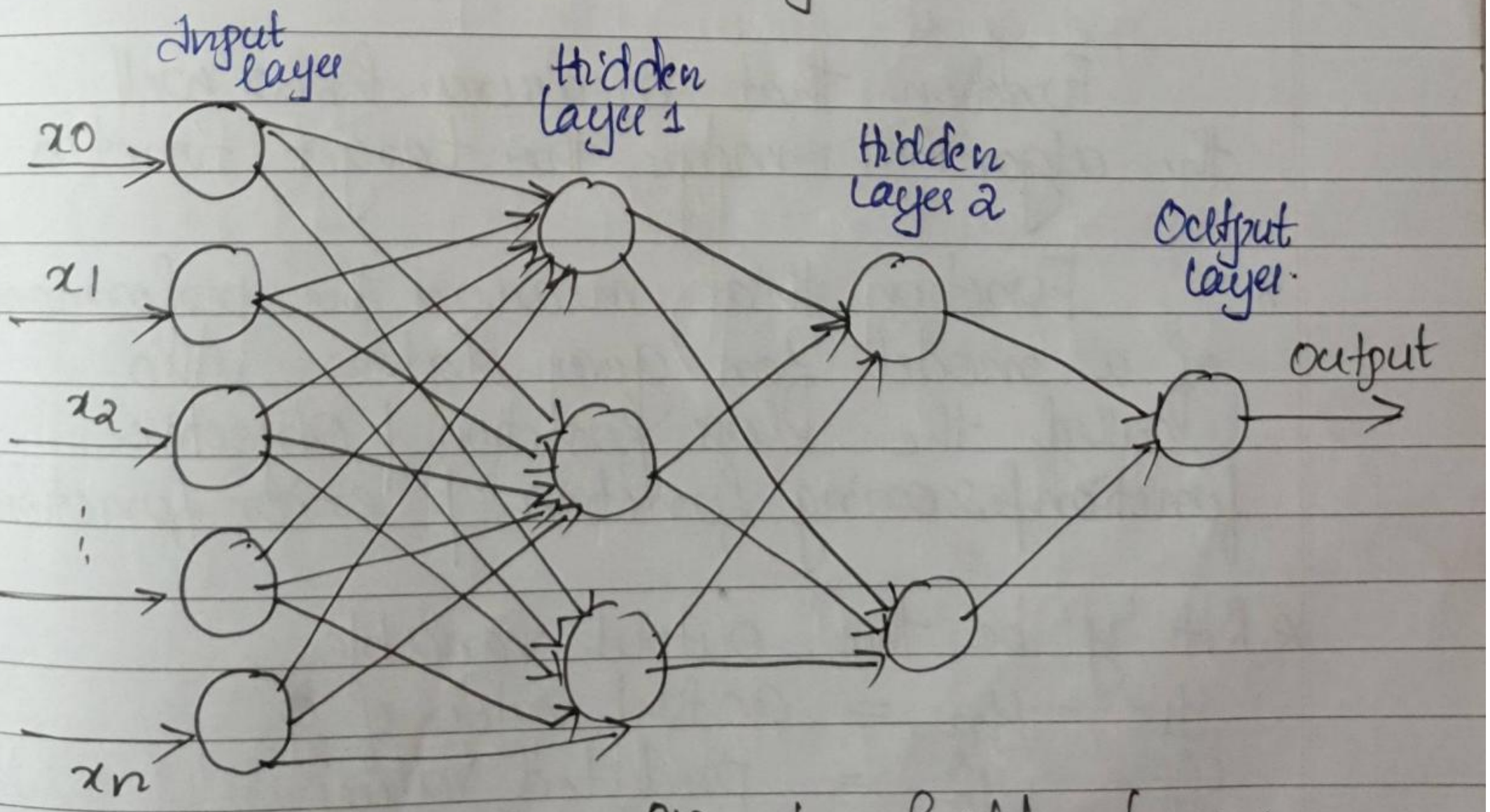
c) The number of nodes within each layer of the network

- Number of features in the input data ⇒ the number of input nodes
- Number of output nodes = no: of outcomes to be modeled
- Number of hedden nodes
  ↳ user to decide prior to training the model

↳ The appropriate number depends on no: of input nodes, amount of training data, amount of noisy data, complexity of the learning task and many other factors.



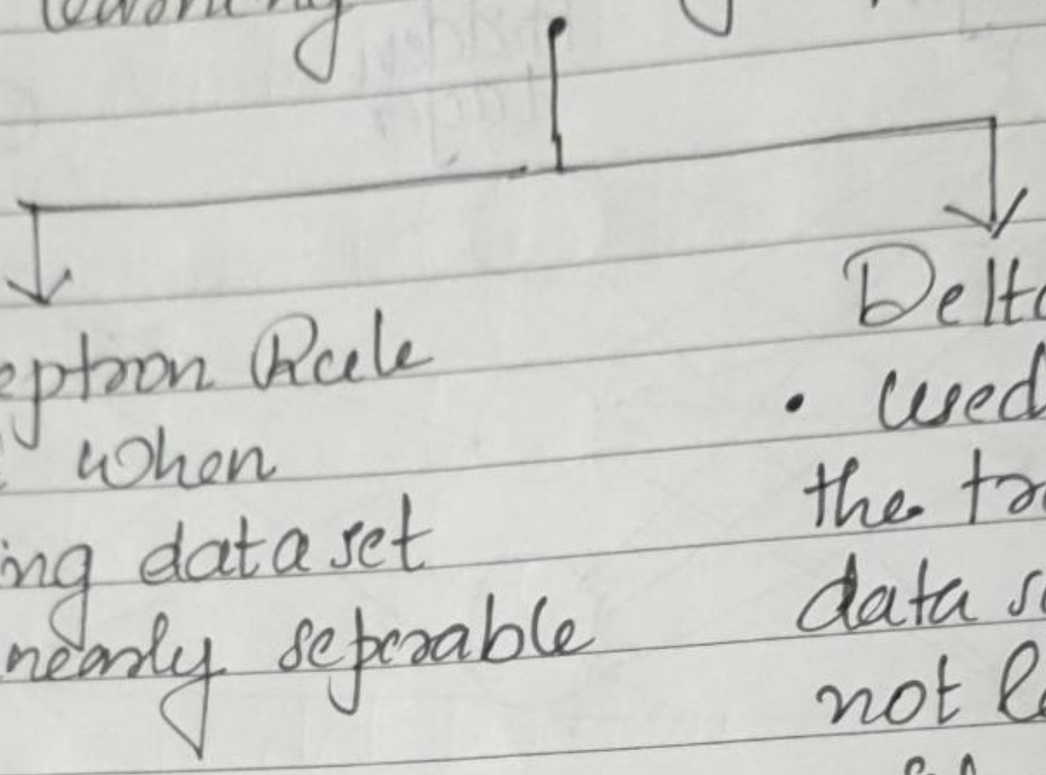
Network with one hidden layer and two output nodes.



Network with two hidden layers.

3) The training algorithm

- algorithm specifies how connection weights are set in order
- Mainly two algorithms for learning a single perceptron



4) Cost Function:-

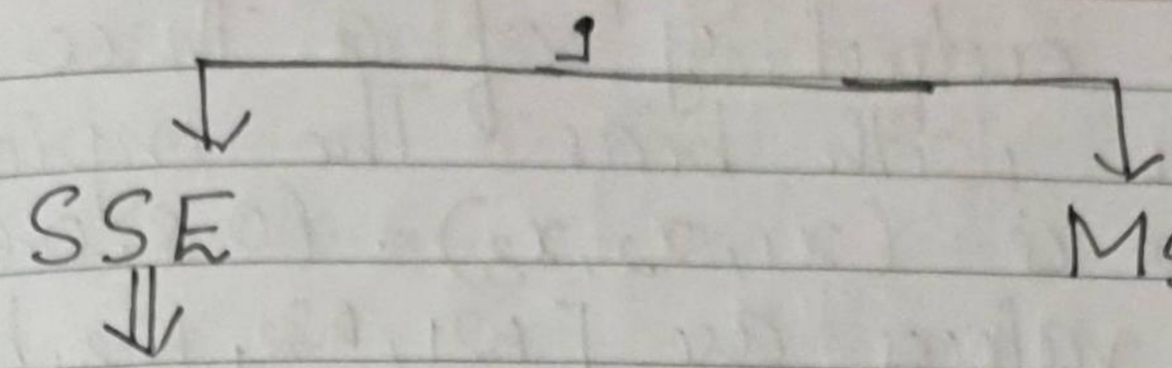
Function that measures how well the algorithm maps the target function

* Function that measures the performance of a model for given data. Also called the loss function / objective function / scoring function / error function

* Let 'y' be the output variable

$y_1 \cdots y_n$ = actual values of y

$\hat{y_1} \cdots \hat{y_n}$ = predicted values



- Sum of squares of differences between the predicted and actual value of 'y' denoted by SSE

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

- Mean of the sum of squares of the difference between the predicted and actual values of y denoted by MSE

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

Back Propagation.

* Backpropagation is an algorithm for supervised learning of artificial neural networks using gradient descent.

* Given an artificial neural neural network and an error function, the method calculates the gradient of the error function with respect to the neural network's weights.

1) Initially the weights are assigned random
2) The algorithm iterates through many cycles of two processes called
   (a) forward phase
   (b) Backward phase.

(a) A forward phase :-

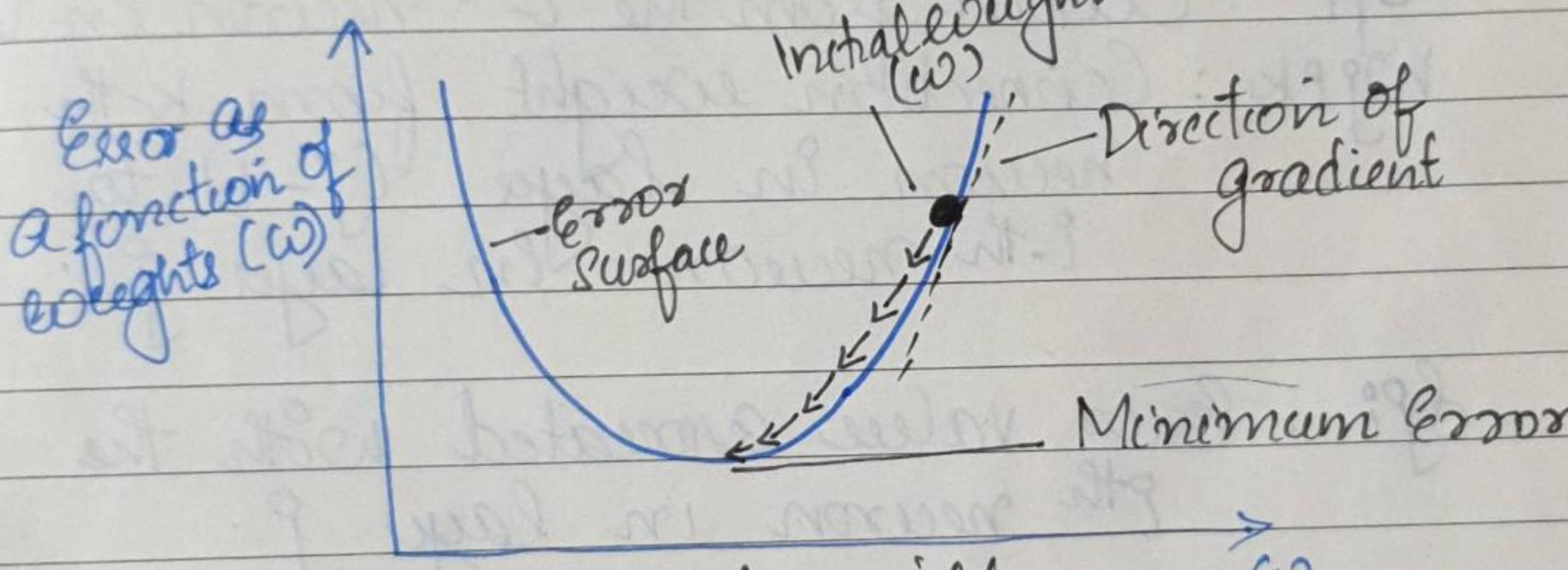
* Neurons are activated in sequence from the input layer to the output layer

* Apply neurons weights and activation function along the way. An output signal is produced from the final layer.

(b) A backward phase:-

* Output signal compared with the true target value
* Difference in results is ~~zero~~ error is propagated backwards in the network to modify the connection weights

3) The technique used to determine how ~~much~~ much a weight can be changed is known as gradient descent method



Error against weights :-

- At any point on the surface of the curve, the gradient suggests how steeply the error will be reduced or increased for a change in the weight
* Algorithm helps to change the weights that result in the greatest reduction in error.

Algorithm:-

M:→No: of layers (excluding the input layer which is assigned the layer no: 0

N_j : No: of neurons in the j^{th} layer.

$X_p = (X_{p1}, X_{p2} \cdots X_{pN_0})$: p-th training sample

$T_p = (T_{p1}, T_{p2} \cdots T_{pN_M})$: Known output corresponding to pth training sample

$O_p = (O_{p1}, O_{p2}, \cdots O_{pN_M})$: Actual output to the p^{th} training sample

y_{ji} : Output from the i^{th} neuron in layer j

w_{jik} : Connection weight from k-th neuron in layer $(j-1)$ to i-th neuron in layer j.

δ_{pi} : Error value associated with the i^{th} neuron in layer j.

Step 1 : Initialize connection weights into small random variables

Step 2: Present the p^{th} sample input vector of pattern

$$X_p = (X_{p1}, X_{p2} \cdots X_{pN_0})$$

and the corresponding o/p target

$$T_p = (T_{p1}, T_{p2}, \cdots T_{pN_M})$$

Step 3: Pass the input values to the first layer 1.
For every input node 'i' in layer 0;

$$y0_i = x_{pi}$$

Step 4: for every neuron 'i' in every layer
$j = 1, 2, \ldots M$. find the output

$$y_{jp} = f\left(\sum_{k=1}^{N_{j-1}} (y_{(j-1)k} \, w_{jpk})\right)$$

Algorithm:-

Step 1 : Initialize connection weights into small random variables.

Step 2 : Feed forward network

$$O_{ij} = \sum w_{ij} O_i + \Theta$$

Θ = Bais

· Apply Sigmoid Activation function

$$O_j = \frac{1}{1 + e^{-O_{ij}}}$$

→ Error = $\sum E_{O_1} + E_{O_2} \dots$

Step 3 : Back Propagated network

⇒ Error calculation of output layer

$$E_O = O_j (1 - O_j) (T_j - O_j)$$

O_j = Output value

T_j = Target value

→ Error Calculation of hidden layer

$$E_H = O_j (1 - O_j) \sum_K E_K \cdot W_{jK}$$

Step 4 : Weights are updated by the Equation

$$\Delta W_{ij} = \text{Learning rate} * \text{Error} * O_{ij}$$

$$W_{ij} = W_{ij} + \Delta W_{ij}$$

Biases are updated by

$$\Delta \theta_j = L * E$$

$$\theta_j = \theta_j + \Delta \theta_j$$

Step 3 and 4 are repeated until stopping criteria meets.

MODULE 1.

Perceptron:-

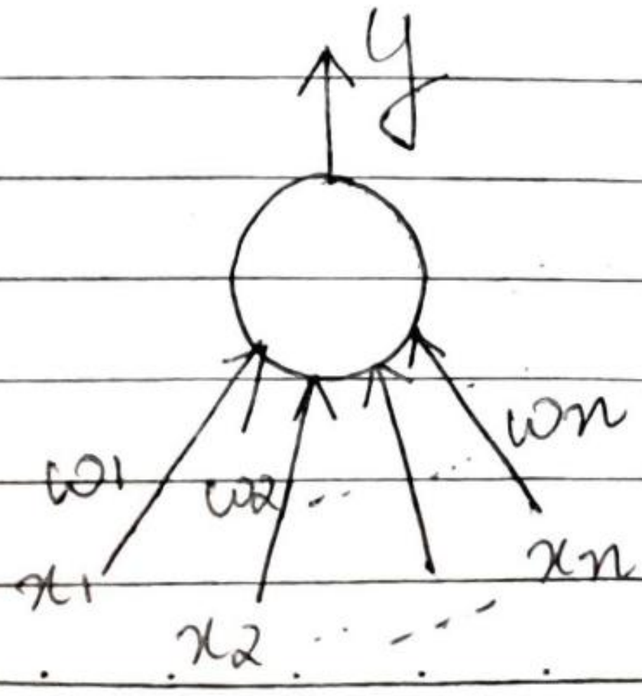
* Mchlloch pitts gives solution for Boolean inputs, also hand code the threshold

* If it is non-boolean inputs, we need a way to learn our problem.

* All inputs are not equal means (weight) priority will be given for each input and based on that our decision lies on

* Perceptron will perform the decision making based on some learning algorithm and weights associated with each input

* Frank Rosenblatt, an American psychologist, proposed the classical perceptron model (1958)

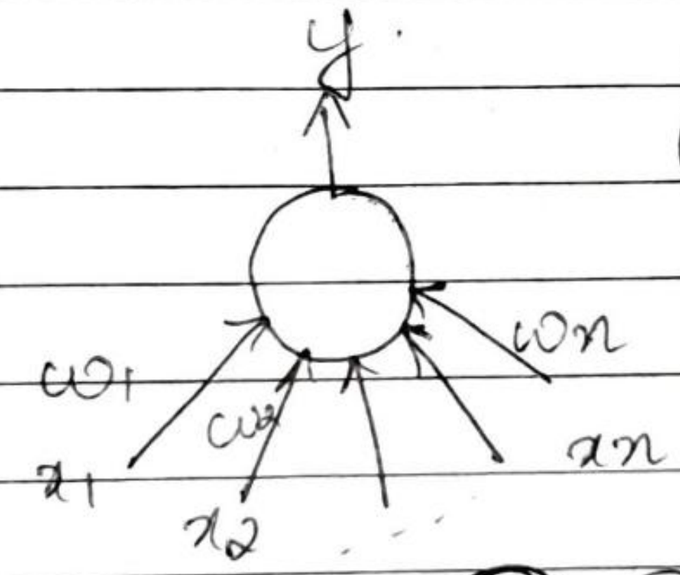


* A more general computational model than McCulloch-Pitts Neurons.

* Main Difference:
Introduction of numerical weights for inputs and a mechanism for learning these weights.

* Inputs are no longer limited to boolean values.

* Refined and carefully analyzed by Minsky and papert (1969) - their model is referred to as the Perceptron model.



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

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

(2) Rewriting the above,

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

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

(3) → A more accepted convention,

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

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

→ where $w_0 = \theta$ and $x_0 = 1$

URBAN EDGE

Eg:

Consider the task of predicting whether we would like a movie or not?

* Suppose, we have our decision based on 3 ips (binary)

* Based on our past viewing experience (data), we may give a high weight to is Director Nolan as compared to other inputs.

* Specifically, even if the actor is not Matt Damon and the genre is not thriller we would still want to cross the threshold (θ) by assigning a high weight to is Director Nolan.

* 'w_0' is called the bias as it represents the prior.

* A movie buff may have a very low threshold and may watch any movie irrespective of the genre, actor / director [$\theta \neq 0$]

* On the other hand, a selective viewer may only watch thrillers starring

URBAN EDGE

Matt Damon and directed by Nolan
[Θ = 3]

* The weights ($w_1, w_2, \ldots w_n$) and the bias (w_0) will depend on the data.

McCulloch Pitts Neuron (assuming no inhibitory i/ps)	Perceptron
$y = 1$ if $\sum_{i=0}^{n} x_i \geq \theta$	$y = 1$ if $\sum_{i=0}^{n} w_i * x_i \geq \theta$
$= 0$ if $\sum_{i=0}^{n} x_i < \theta$.	$= 0$ if $\sum_{i=0}^{n} w_i * x_i < \theta$

Continue perceptron boolean fns

Algorithm: Perceptron learning Algorithm:-

P ← inputs with label 1.
N ← inputs with label 0
Initialize $w = [w_0, w_1, w_2, \ldots, w_n]$ randomly;

while !Convergence do (until not making any more errors on the training data)

pick random $x \in P \cup N$;
if $x \in P$ and $\sum_{i=0}^{n} w_i * x_i < 0$ then
$w = w + x$;
end
if $x \in N$ and $\sum_{i=0}^{n} w_i * x_i \geq 0$ then
$w = w - x$
end
end

↳ Consider 2 vectors 'w' and 'x'

$$w = [w_0, w_1, \ldots, w_n]$$
$$x = [x_0, x_1, \ldots, x_n]$$

$$w \cdot x = w^T x = \sum_{i=0}^{n} w_i * x_i$$

↳ Rewriting the perceptron rule as

$y = 1$, if $w^T x \geq 0$
$= 0$, if $w^T x < 0$

URBAN EDGE

* Finding the line $w^T x = 0$, which divides the input space into two halves.

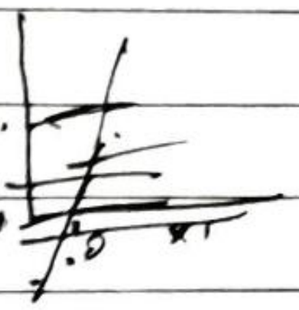
↳ i.e every point 'x' on this line satisfies the equation $w^T x = 0$

eg:- $x_1 + x_2 = 0$.

points - (1, -1) = 0

(2, -2) = 0

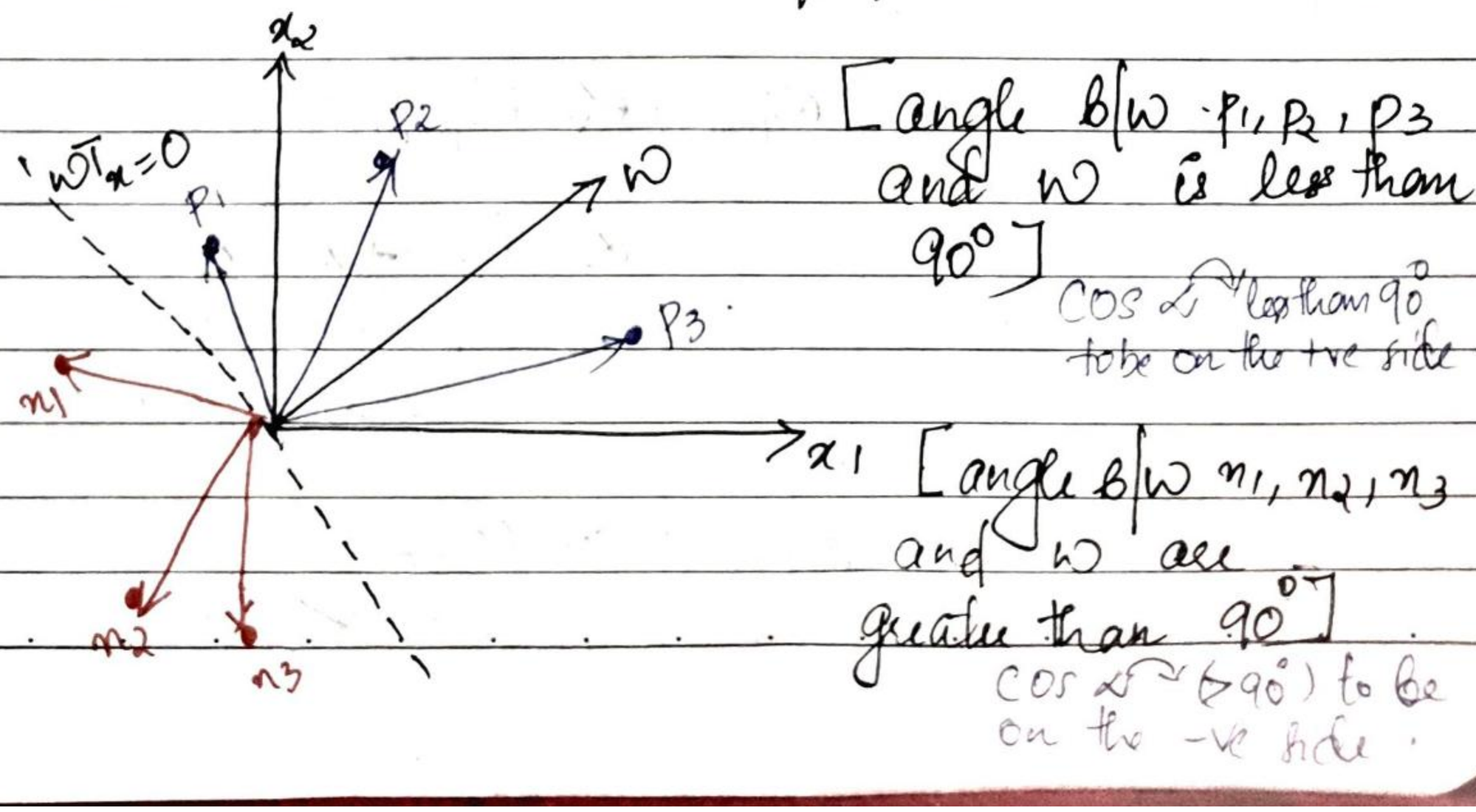
[$w^T x$ ie dot product ($w \cdot x = 0$), then they are orthogonal means every points lie on the line & perpendicular to the axis]



[Dot product of 2 vectors is equal to the product of the magnitude of the 2 vectors and the cosecant of the angle b/w the two vector]

i.e The angle b/w point on line and the 'w' is 90° so

$$\cos\alpha = \frac{w^T x}{|w||x|} = 0$$



[angle b/w P_1, P_2, P_3 and w is less than 90°]

cos ∠ less than 90° to be on the +ve side.

[angle b/w n_1, n_2, n_3 and w are greater than 90°]

cos ∠ (> 90°) to be on the -ve side.

P ← inputs with label 1;
N ← inputs with label 0;
Initialize w randomly;

While ! Convergence do
    Pick Random x ∈ PUN;
        if x∈P and w·x<0 then
            w = w+x;
        end
        if x∈N and w·x ≥ 0 then
            w = w-x;
        end
// the algorithm converges when all the inputs are classified correctly.

95 > 90
cos 95 = -0.087
cos 90 = 0
cos 95 < cos 90
95 > 90 + ve.

* For x∈P if w·x<0, then it means the angle(α) b/w w and x is greater than 90°. [But we want 'α' to be less than 90°]

* Compute $w_{new} = w + x$.

$\cos(\alpha_{new}) \propto w_{new}^T x$

$\propto (w+x)^T x$

$\propto w^T x + x^T x$

$\propto \cos\alpha + x^T x$.

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

∴ α_{new} is $< 90^\circ$

URBAN EDGE ___/___/___

* For $x \in N$, if $w.x \geq 0$ then it means that the angle (α) b/w the x and the current w is less than 90°. (but we want 'α' to be greater than 90°)

$\cos(\alpha_{new}) \propto w_{new}^T x$

$\propto (w - x)^T x$

$\propto w^T x - x^T x$ +ve value.

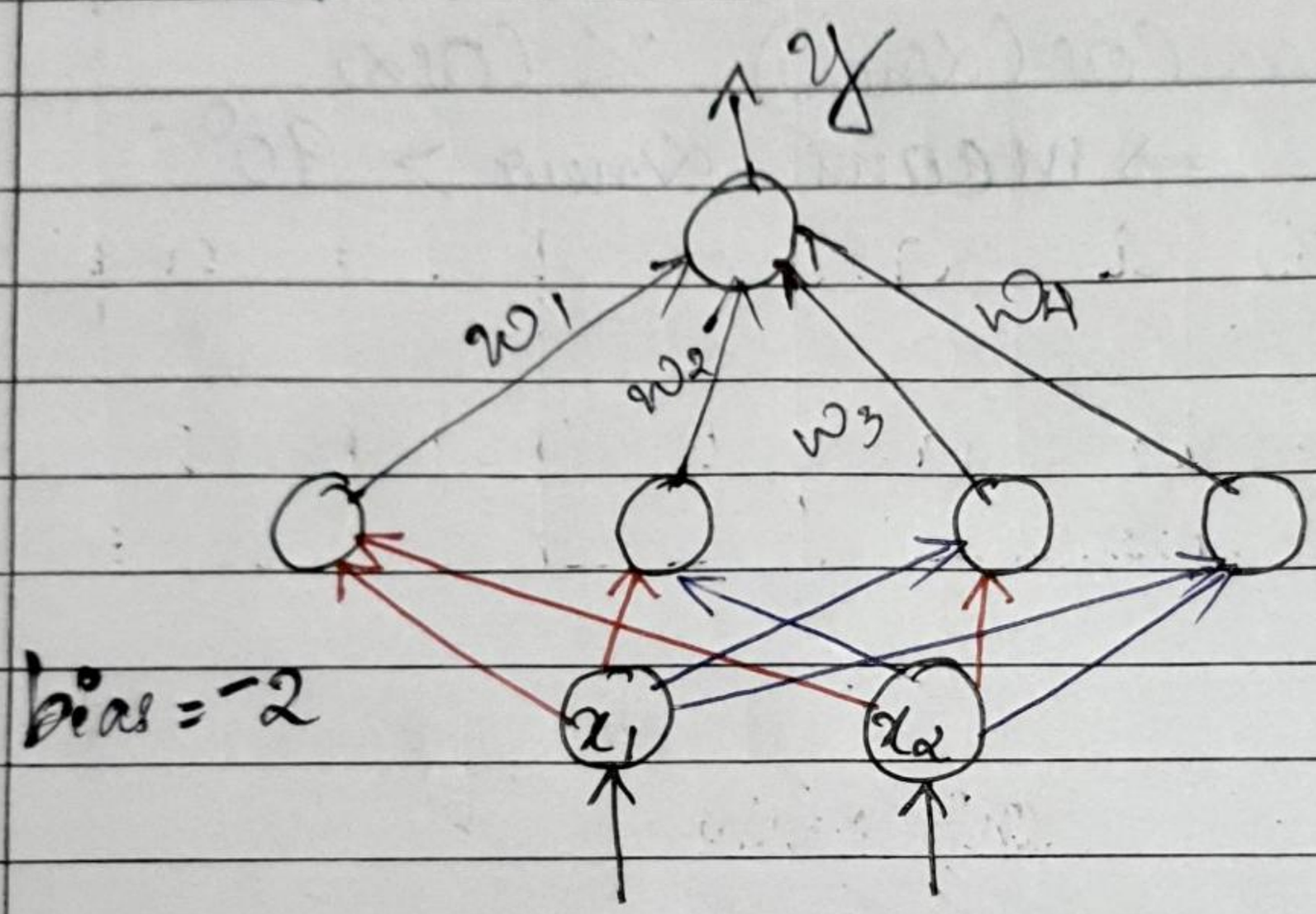
$\propto \cos\alpha - x^T x$

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

$\Rightarrow$ means $\alpha_{new} > 90^\circ$.

Network of Perceptrons :-
OR
Multilayer Perceptrons :-

*



red indicates w = -1
blue edge indicates w = +1

* Consider 2 inputs and 4 Perceptrons
* Each input is connected to all the 4 perceptrons with specific weights
* The bias (w0) of each perceptron is -2 (ie each perceptron will fire only if the weighted sum of its input is ≥ 2)
* Each of these perceptrons is connected to an output perceptron by weights (which need to be learned)

Terminology :-

Input layer
hidden layer
O/p layer

* O/ps of 4 Perceptrons in the hidden layer are denoted by h_1, h_2, h_3, h_4
* The red and blue edges are called layer 1 weights
* w_1, w_2, w_3, w_4 are called layer 2 weights
* This n/w can be used to implement any boolean functions (linearly separable or not)
* Perceptron 1 will fire when the input is (-1, -1)
* Perceptron 2 " → (-1, 1)
* Perceptron 3 " → (1, -1)
* Perceptron 4 " → (1, 1)

eg XOR

Let w_0 be the bias o/p of the neuron (ie, it will fire if $\sum_{i=1}^{4} w_i h_i \geq w_0$)

x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i \geq w_0$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2
1	0	1	0	0	1	0	w_3
1	1	0	0	0	0	1	w_4

* To implement XOR:

$w_1 < w_0$

$w_2 \geq w_0$

$w_3 \geq w_0$

$w_4 < w_0$.

* Each w_i is now responsible for one of the 4 possible inputs and can be adjusted to get the desired o/p for that i/p.

URBAN EDGE ___/___/_____

* If 3 i/ps: bias = 3 and hidden layer containing 2^3 neurons ie (8)

* If 'n' i/ps hL will 2^n neurons.

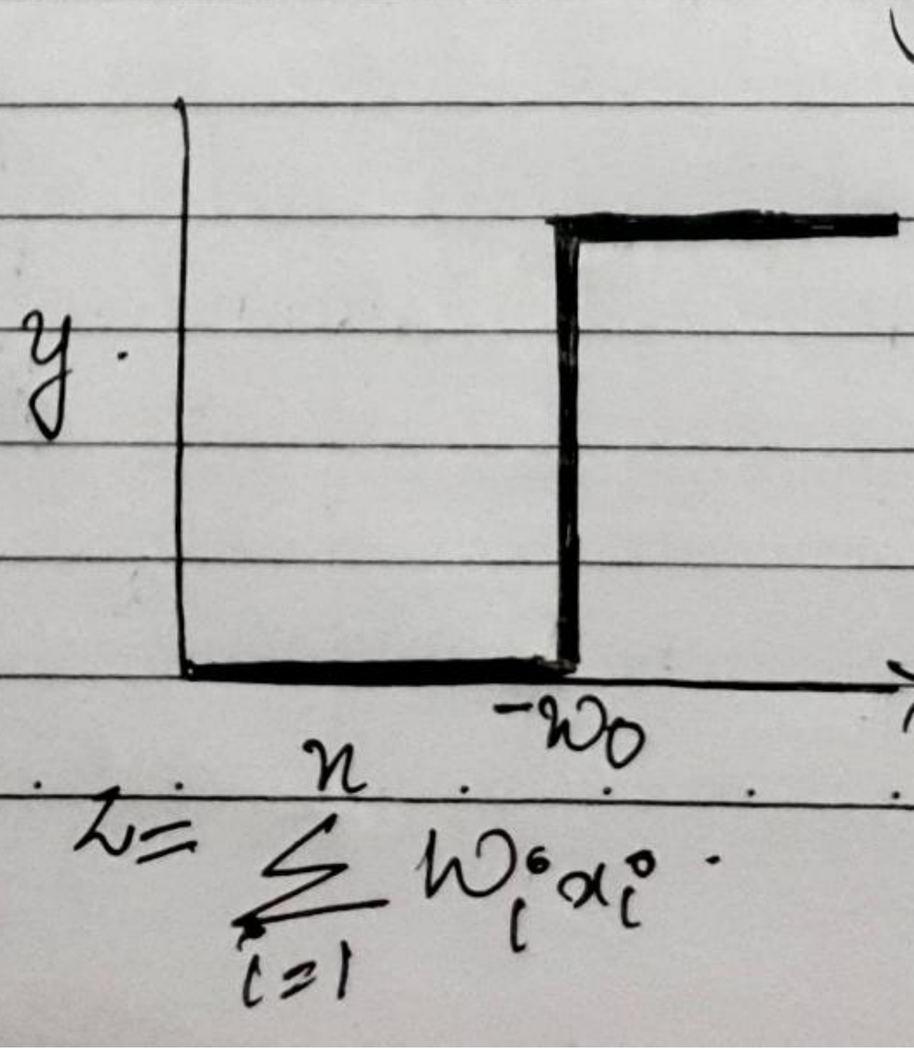
* Any boolean fn of 'n' inputs can be represented exactly by a n/w of perceptrons containing 1 hidden layer with 2^n perceptron and one o/p layer containing 1 Perceptron

* As 'n' increases the no: of perceptrons in the hidden layers obviously increases exponentially.

Sigmoid Neuron:-

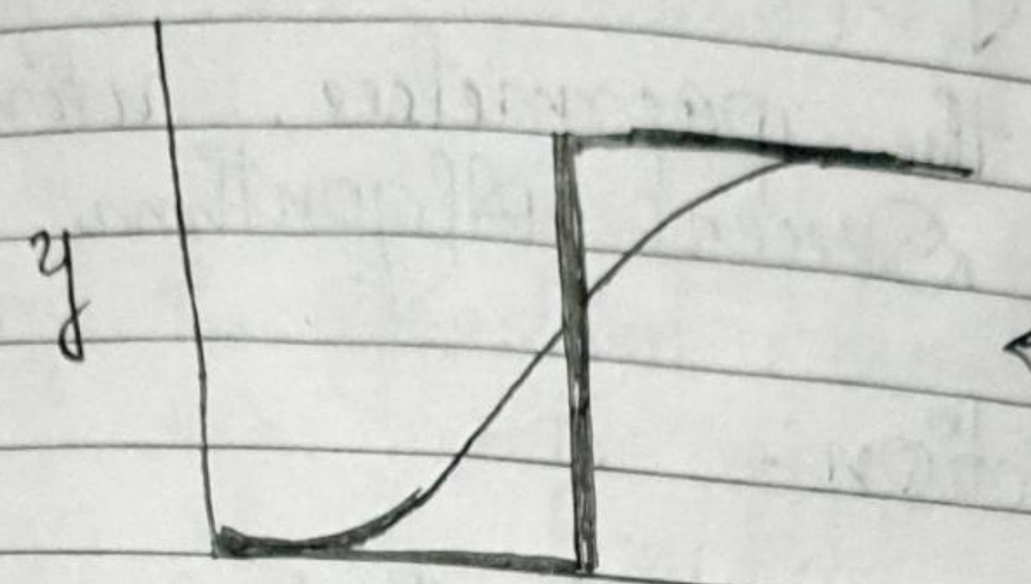
* A sigmoid neuron is similar to the perceptron neuron, for every input 'x_i' it has weight 'w_i' associated with the input.
* The weights indicate the importance of the input in the decision making process.
* The output from the sigmoid is not 0 or 1 like the perceptron model instead it is a real value b/w 0-1 which can be interpreted as a probability.
* The most commonly used sigmoid fn is the logistic function, which has the characteristics of an 'S' shaped curve.
* It is an arbitrary function of the of form $y = f(x)$ where $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$.

↳ Perceptron function itself which behaves like a step function



* There will always be this sudden change in the decision (from 0 to 1) when z. crosses the threshold ($-w_0$)

* For most real world application we would expect a smoother decision fn which gradually changes from 0 to 1



← One form of the sigmoid fn called the logistic fn.

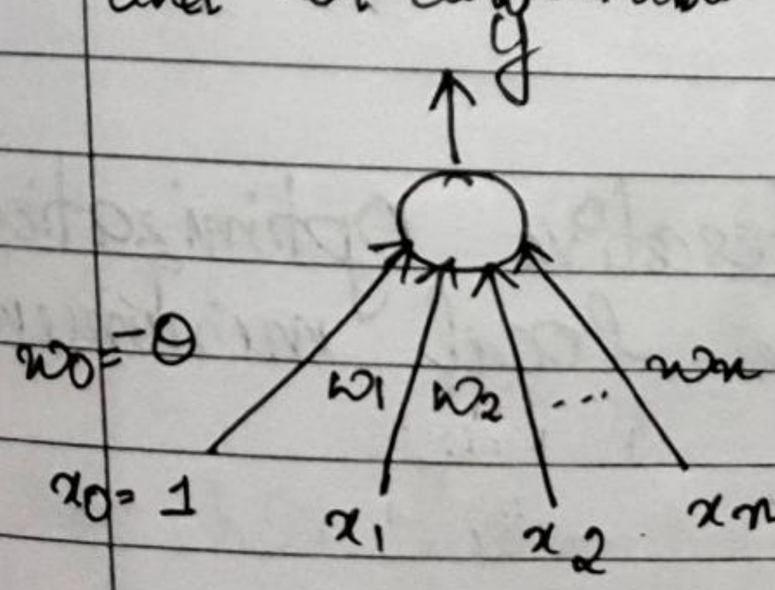
$$y = \frac{1}{1 + e^{-(w_0 + \sum_{i=1}^{n} w_i^o x_i^o)}}$$

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

when $w^T x = \infty$,
$y = 1$
when $w^T x = -\infty$
$y = 0$.

Perceptron

* Not smooth, not continuous and not differentiable

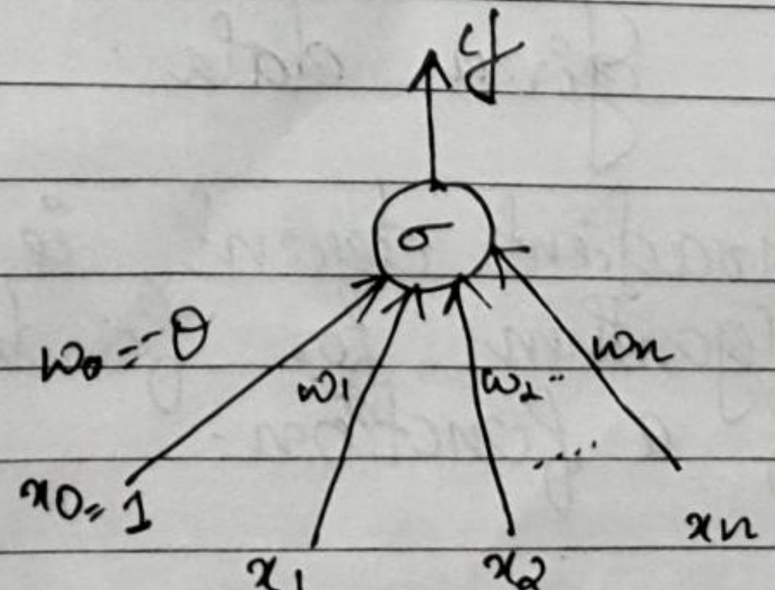


$y = 1$ if $\sum_{i=0}^{n} w_i^o * x_i^o \geq 0$

$= 0$ if $\sum_{i=0} w_i^o * x_i < 0$

Sigmoid (logistic) Neuron

* Smooth, Continuous and differentiable



$$y = \frac{1}{1 + e^{-\sum_{i=0}^{n} w_i x_i}}$$

Gradient Descent Algorithm

* We had an algorithm for learning the weights of a perceptron, we also need a way of learning the weights of a sigmoid neuron.

* To learn the parameters, using Gradient Descent Algorithm.

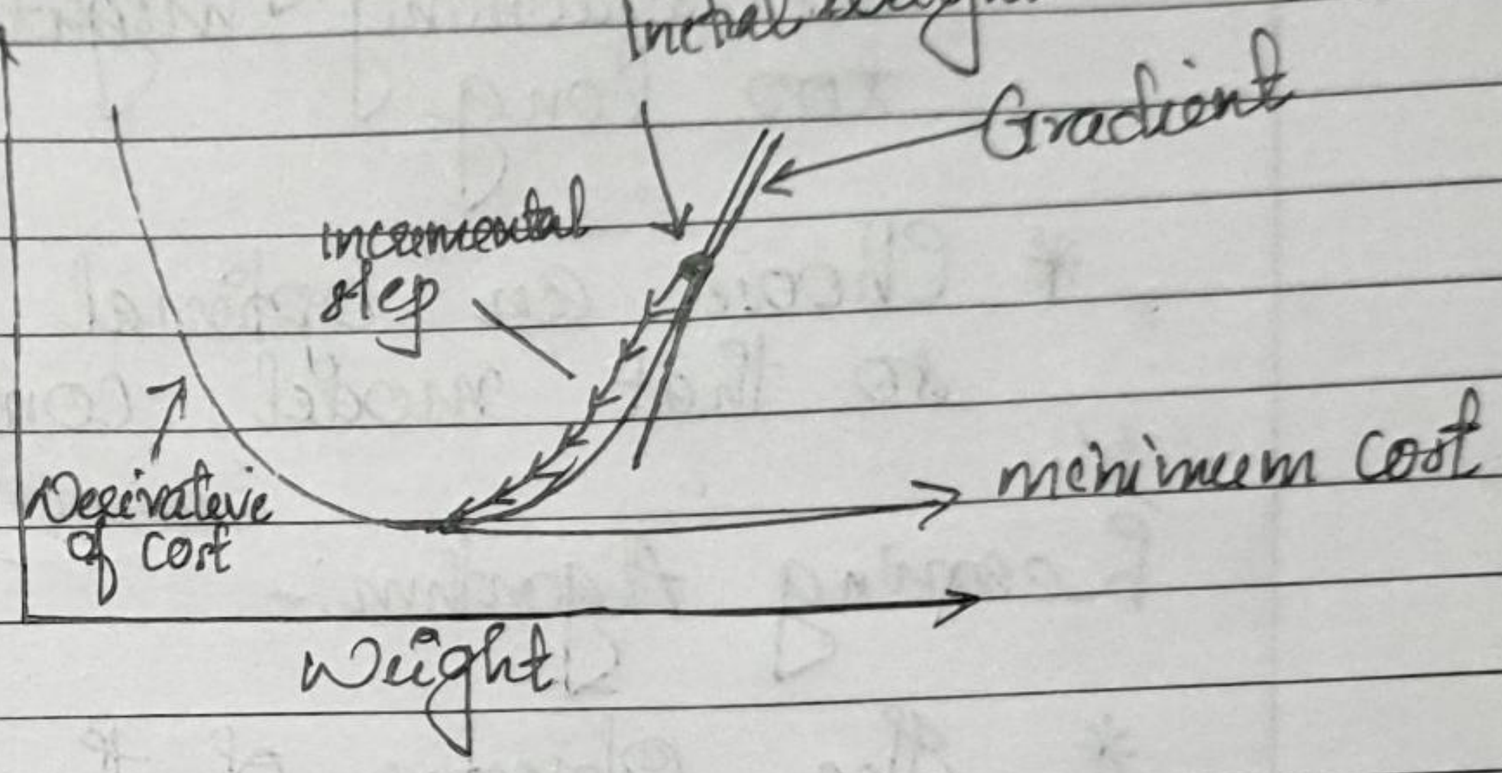
Cost Function:-

- It is a function that measures the performance of a model for any given data.
- Cost fn quantifies the error b/w predicted values and expected values and presents it in the form of a single real number.
- After making a hypothesis with initial parameters, we calculate the cost fn.
- Goal is to reduce the cost fn, for that modify the parameters by using the Gradient Descent Algorithm over the given data.

* Gradient descent is an iterative Optimization Algorithm for finding the local minimum of a function.

* To find the local minimum of a fn using gradient Descent, takes steps proportional to the negative of the gradient (move away from the gradient) of the fn at the current point

- If take steps proportional to the tve of the gradient (moving towards the gradient), we will approach a local maximum of the function. This procedure is called Gradient Ascent.



* The goal of the gradient Descent Algorithm is to minimize the given function
  (1) Compute the gradient (slope), the first order derivative of the fn at that point

  (2) Make a step in the direction opposite to the gradient.

* We have now the direction to move in, next is to decide the size of the step

Learning rate (η) - a tuning parameter in the optimization process
- It decides the length of the steps.

* If the learning rate is too high, it might overshoot the minima and keep bouncing without reaching the minima

* If the learning rate is too small, the training might turn out to be too long

* Choose an optimal learning rate, so that model converges to the minimum.

Learning Algorithm:-

* The objective of the learning algorithm is to determine the best possible values for the parameters (w and b), such that the overall loss of the model is minimized as much as possible

repeated until converge:
- Initialize w, b
- Iterate over data:
  - Compute $\hat{y}$
  - Compute $\mathcal{L}(w, b)$
  - $w_{t+1} = w_t - \eta \Delta w_t$
  - $b_{t+1} = b_t - \eta \Delta b_t$

URBAN EDGE ___/___/___

, where $\Delta w_t = \frac{\partial L(w,b)}{\partial w}$ at $w=w_t, b=b_t$

$\Delta b_t = \frac{\partial L(w,b)}{\partial b}$ at $w=w_t, b=b_t$.

Feed Forward Neural Network:-

* Feed forward neural networks are also called deep feedforward networks.

* The goal of a feedforward network is to approximate some function f^*.

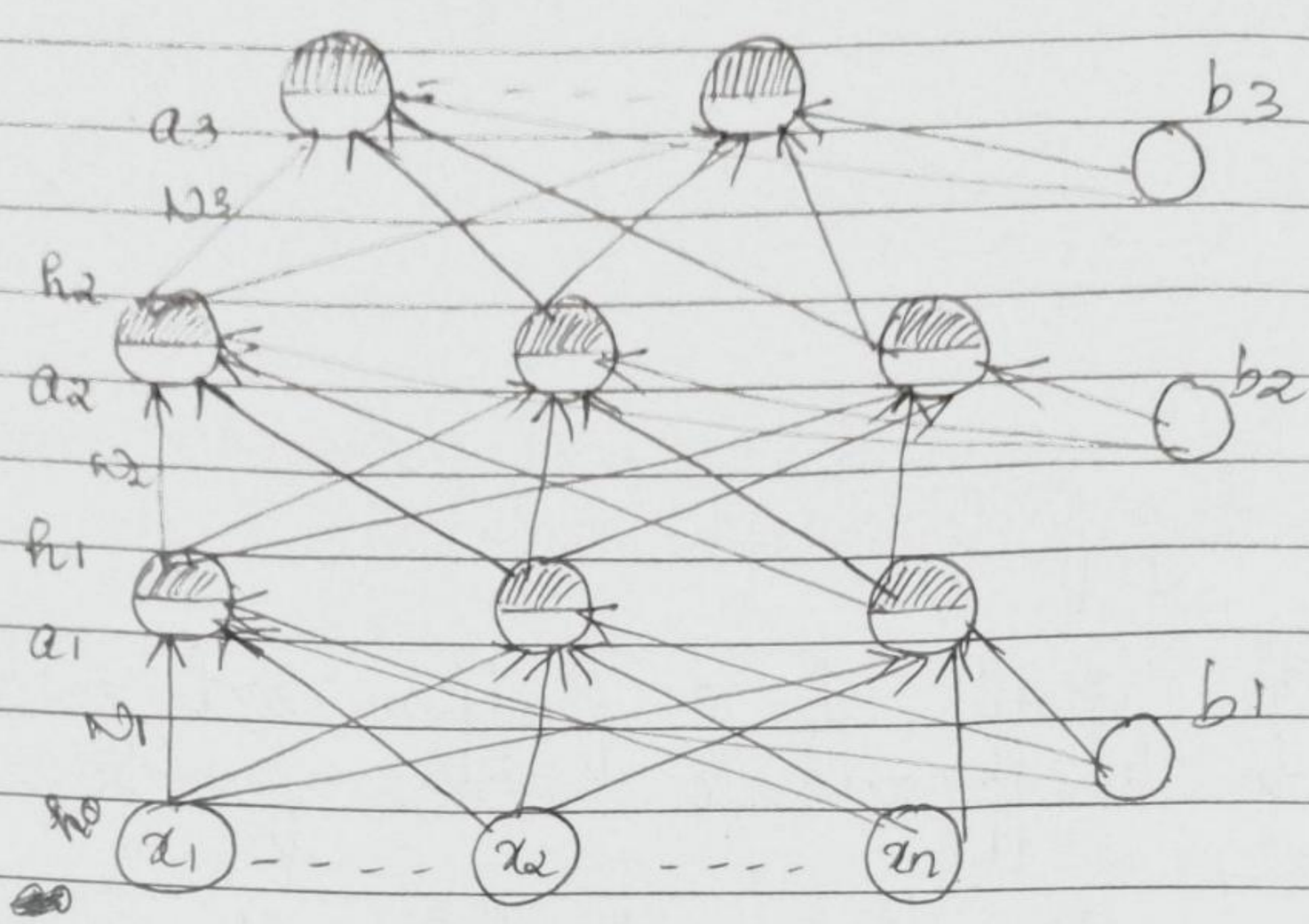
  eg:- for a classifier, $y = f^*(x)$ maps an input x to a category y.

* A feedforward n/w defines a mapping
  $y = f(x, \theta)$
  └→ 'θ' is the parameter to learn that results in the best function approximation.

* Such models are called feedforward bcz information flows through the function being evaluated from 'x', through the intermediate computations used to define 'f' and finally to the op 'y'.

*

$$h_\theta = \hat{y} = f(x)$$



* The input to the n/w is a n-dimensional vector

* The n/w contains 'L-1' hidden layers (2 in the above case) having 'n' neurons each

* Finally, there is one output layer containing 'K' neurons (corresponding to 'K' classes)

* Each neuron in the hidden layer and

and output layer can be split into two parts :

- Pre activation and
- Activation

* Each neuron in the hidden the input layer can be called the '0th' layer and o/p layer can be called the 'L+1th' layer

* $W_i \in \mathbb{R}^{n \times m}$ and $b_i \in \mathbb{R}^n$ are the weight and bias b/w layers $i-1$ and i $(0 < i < L)$

* $W_L \in \mathbb{R}^{n \times k}$ and $b_L \in \mathbb{R}^k$ are the weight and bias b/w the last hidden layer and the o/p layer

* The pre-activation at layer 'i' is given by

$$a_i(x) = b_i + W_i h_{i-1}(x)$$

eg:- $a_1 = b_1 + W_1 h_0$

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

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

* The activation at layer 'i' is given by

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

where 'g' is called the activation function (eg: logistic, tanh, linear etc)

* The o/p layer 'i' is given by

$$f(x) - f_{NN}(x) = O(a_L(x))$$

where 'O' is the o/p activation fn (eg:- softmax, linear etc)

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

Model :

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

Parameters : $\theta = W_1, \ldots W_L, b_1, b_2 \ldots b_L$

Algorithm : Gradient Descent with Backpropagation

Objective / Loss / Error fn : Say,

$$\min \frac{1}{N} \sum_{i=1}^{N} (\hat{y_i} - y_i)^2$$

In general $\min L(\theta)$, where $L(\theta)$ is some fn of parameters

URBAN EDGE ___/___/____

Output Functions and Loss functions

	Output	
	Real values	Probabilities
Op Activation	Linear	Softmax
Loss function	Squared Error	Cross Entropy

Computing gradients with respect to the output units :-

$$\frac{\partial L(\theta)}{\partial w_{111}} = \underbrace{\frac{\partial L(\theta)}{\partial \hat{y}}}_{\text{Talk to the o/p layer}} * \frac{\partial \hat{y}}{\partial a_3} \frac{\partial a_3}{\partial h_2} \underbrace{\frac{\partial h_2}{\partial a_2}}_{\text{Talk to the previous hidden layer}}$$

Talk to the weight directly

$$\frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_{111}}$$

[$\hat{y}$ is the predicted o/p]

→ Talk to the previous hidden layer

talk now to the weights

* A computational graph is a directed graph where the nodes correspond to operations or variables
* Variable can feed their value into operations and operations can feed their O/P into other operations

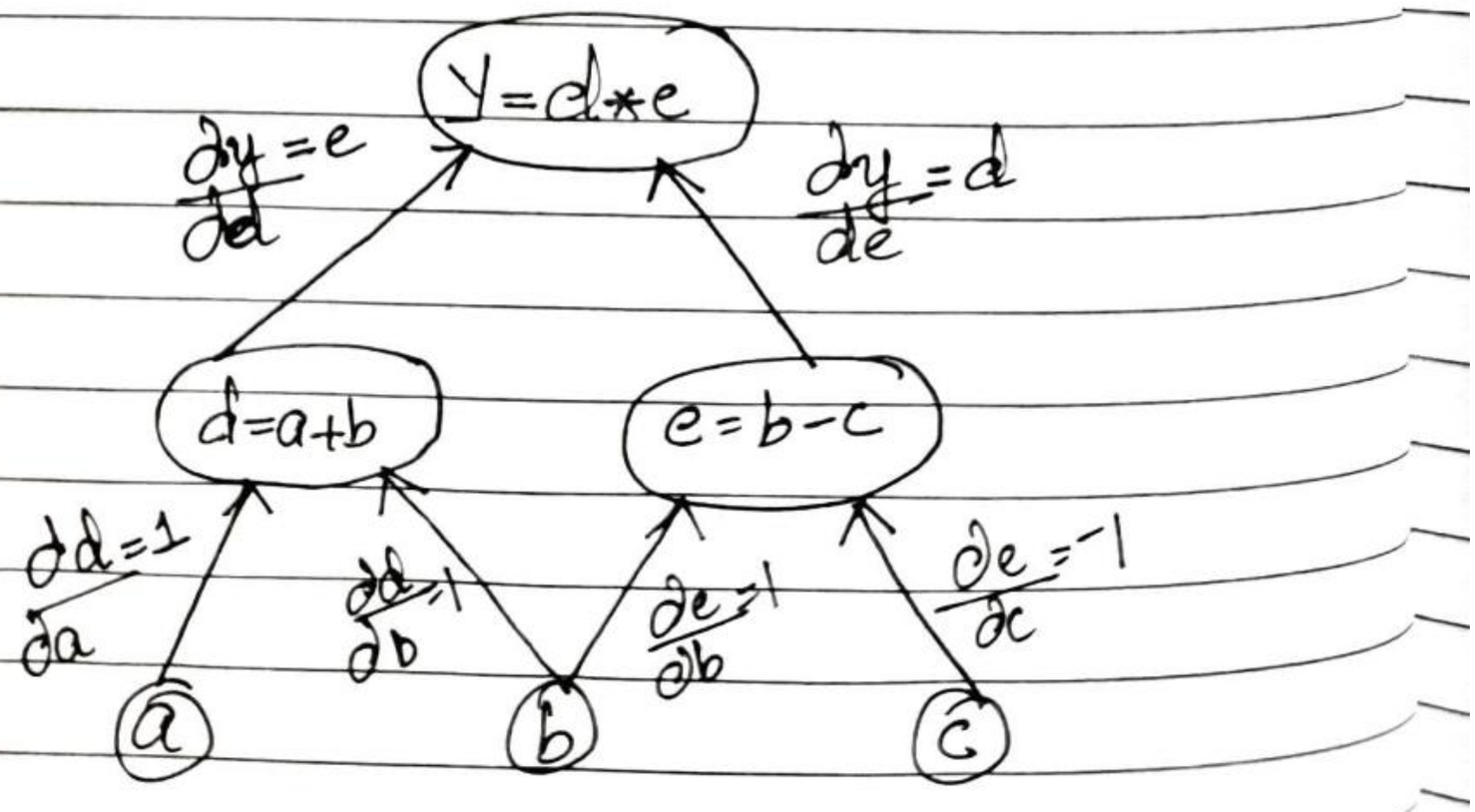
Computational Graphs :-

* Computational graph is a directed graph which is used for expressing and evaluating the mathematical expression.

* Computations of the neural n/w are organized in terms of a forward pass or forward propagation step in which we compute the O/P of the neural n/w followed by a backward pass or backward propagation step, which we use to compute gradients / derivatives.

* To understand derivatives in a computation graph, the key is to understand how a change in one variable brings change on the variable that depends on it.

eg :-



* We can easily apply chain rule to evaluate

partial derivates of final o/p variable w.r.t. input variable a, b, c.

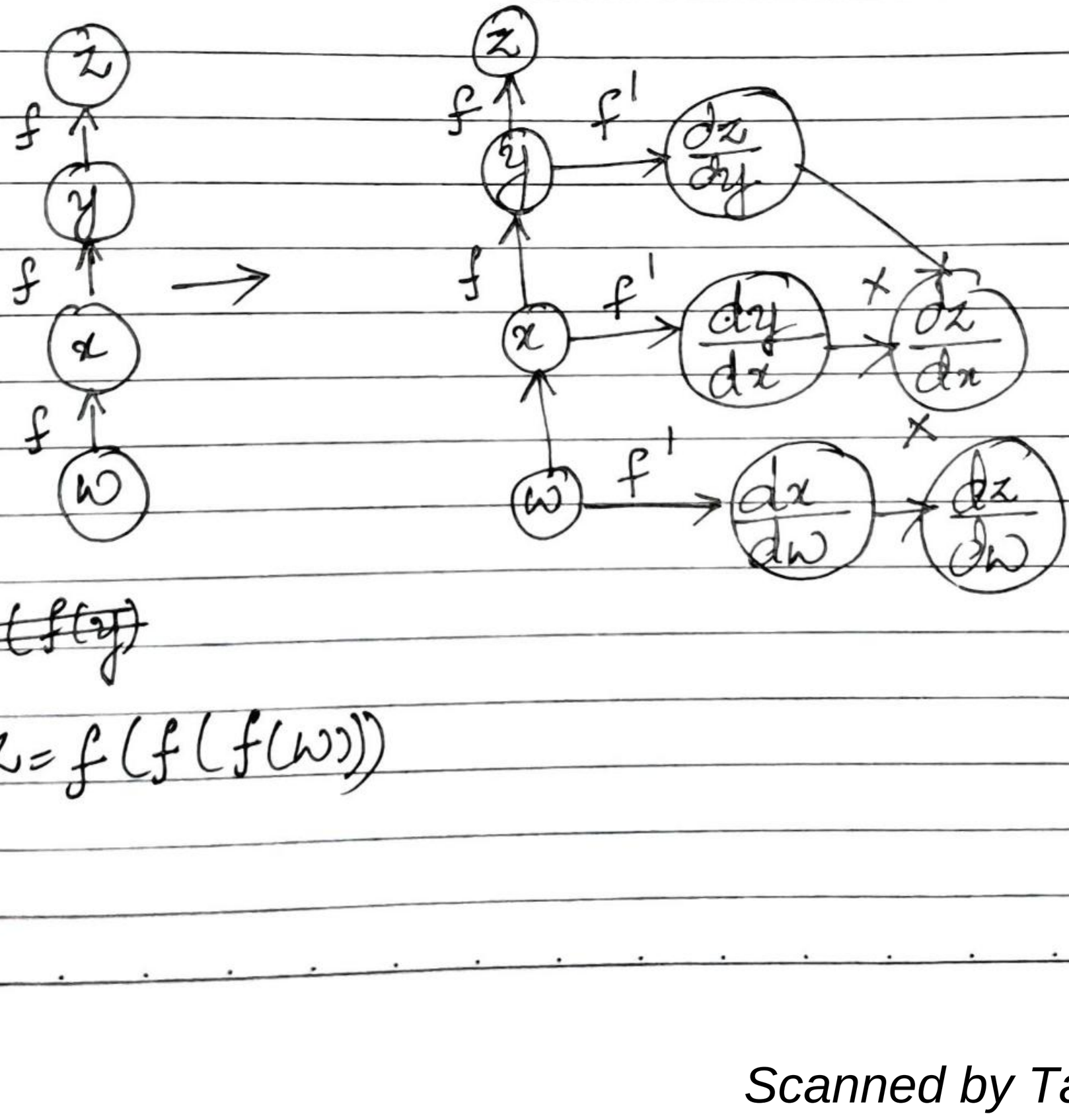
$$\frac{\partial Y}{\partial a} = \frac{\partial y}{\partial d} \times \frac{\partial d}{da} = e \times 1 = e$$

$$\frac{\partial y}{\partial b} = \frac{\partial y}{\partial d} \times \frac{\partial d}{\partial b} = e \times 1 = e$$

$$\frac{\partial y}{\partial c} = \frac{\partial y}{\partial e} \times \frac{\partial e}{\partial c} = d \times -1 = -d$$

* These computational graphs make it easier to get the derivatives using backpropagation

eg:



$z = f(f(f(w)))$