Genetic Algorithm (John Holland)

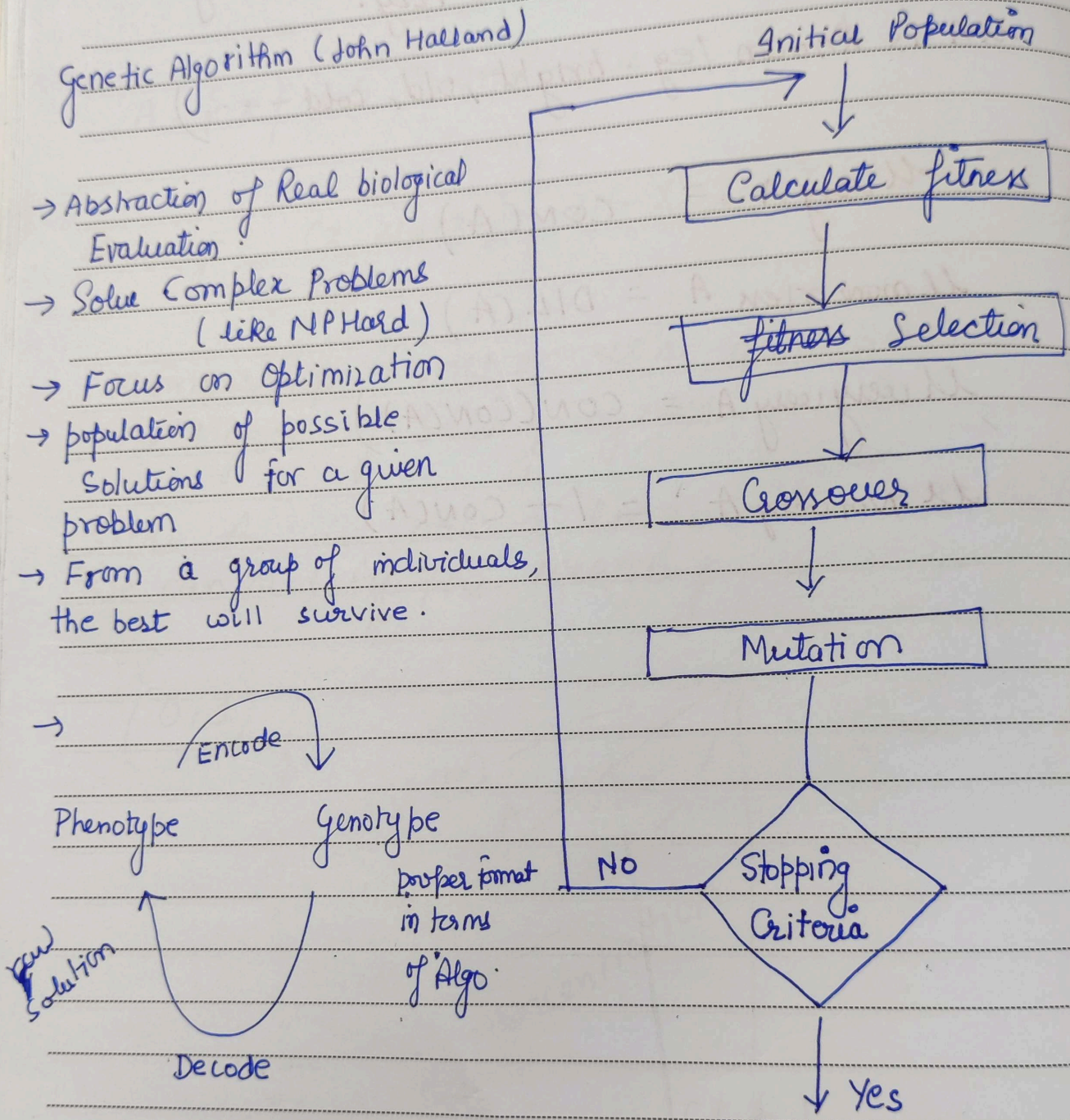
→ Abstraction of Real biological Evaluation

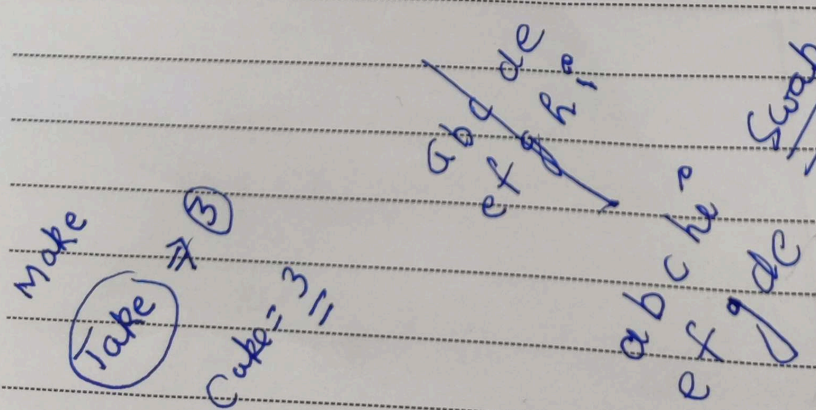
→ Solve Complex Problems (like NP Hard)

→ Focus on Optimization

→ population of possible solutions for a given problem

→ From a group of individuals, the best will survive.





$4x + 8 = 0$ $\quad x = -2, 0$

$2y = 0$

$2 \times 4 + 0 \; 15 + 15$

$8 + 0 - 1 =$

Some benchmark optimization problems.

- ↳ Travelling Salesman Problem
- ↳ Knapsack Problem.
- ↳ Graph Colouring Problem.
- ↳ Job Machine Assignment Problem.
- ↳ Coin change Problem.
- ↳ Binary search tree construction problem.

Proposed Approach :-

we presume that all the sensors are homogeneous and stationary and their location is known.

a) Sensing Coverage representation :-

A set of sensor nodes in a target area A is delineated as

$S = \{s_1, s_2, s_3 \cdots s_m\}$, where s_i is located at coordinates $\{a_i, b_i\}$, where i varies from 1 to M

Let P be the set of PoC's distributed over the area A.

If N is no. of PoC then $\{p_1 p_2 \cdots p_N\}$ where PoC p_j is situated at $[a_j, b_j]$, where j varies from 1 to N.

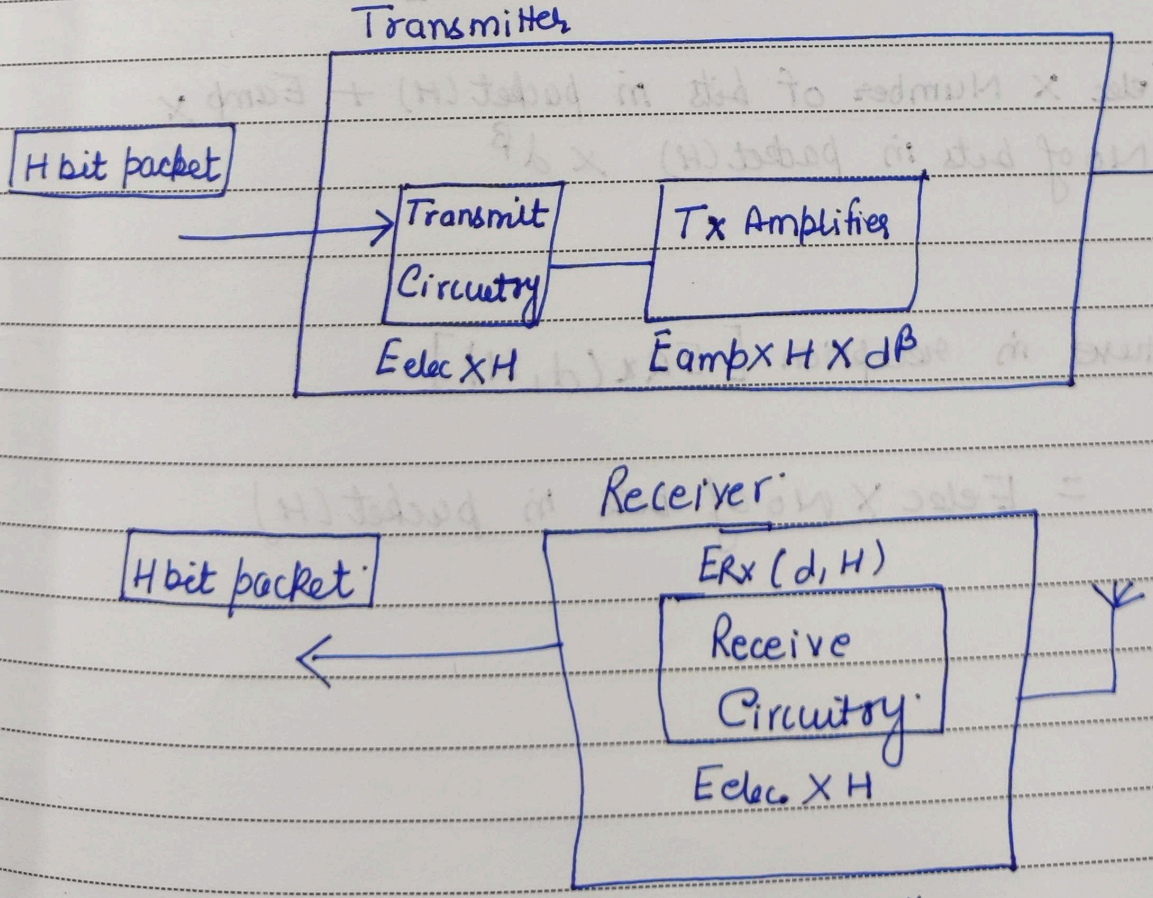
N

A binary Coverage variable C_{ij} which signifies whether sensor S_i covers the POC P_j is defined below

$$C_{ij} = \begin{cases} 1 & \text{if } (a_i - a_j)^2 + (b_i - b_j)^2 \leq R_a^2 \\ 0 & \text{Otherwise.} \end{cases}$$

(b) Energy disbursement Model :-

we exploit the model illustrated by Heinzelman to compute the power disbursement for both kind of communication (transmission as well as reception)



"The energy disbursement model".

In above fig. transmit circuitry or receive circuitry consumes Eelec nano joule energy in transmission or reception of per bit.

Energy expenditure in the power amplification of each bit is depicted by Eamp.

β exponent of path loss.

As a result, when a transmitter transmits a packet of H bit to the receiver, the overall energy expenditure can be determined as follows

Energy Expenditure in Transmission $[E_{Tx}(d, H)]$

= Eelec x Number of bits in packet (H) + Eamp x No of bits in packet (H) $\times d^{\beta}$

Energy Expenditure in reception $[E_{Rx}(d, H)]$

= Eelec x No of bits in packet (H)

Problem formulation :→

to locate such specific set of sensor nodes such that each POC is covered by at least one node.

Optimization model :-

$$\text{Min} \sum \text{Cost}_i \cdot a_i \quad i = 1, 2, 3, 4 \ldots M$$

Subject to

$$\sum c_{i,j} \cdot a_i \geq 1 \qquad j = 1, 2, 3 \ldots N$$

$$a_i = 1 \text{ or } 1 \qquad i \in [1, M]$$

"Absolute sensing - Area Coverage Using extended Genetic Algo."

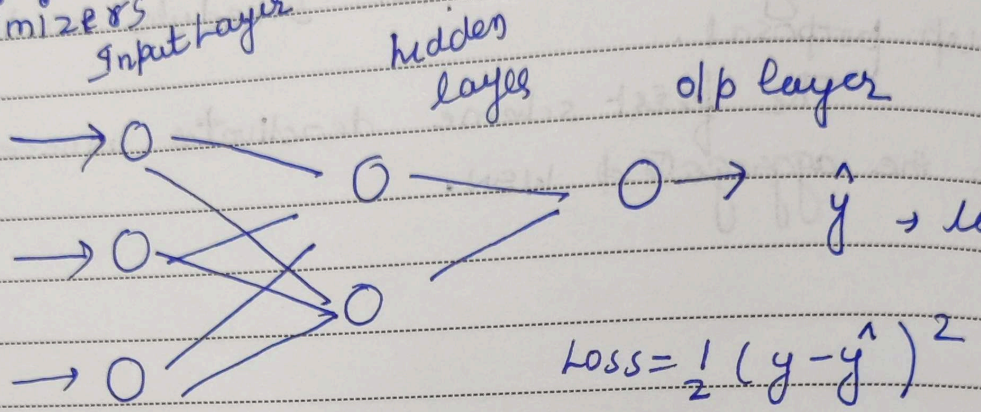
Coverage of the proposed ASCeGA includes 2 optimized strategies: the GA extended approach to determining the sensor node schedule and the stir up proposal.

The first scheme deactivates unnecessary nodes in the aggregated WSN.

←: optimizers :→

1) Gradient Descent.

optimizers



Aim: → To reduce Loss with optimizers to ~~reduce~~ update weights in back propagation.

$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W_{old}}$$ Gradient descent

1) Epoch
2) Iteration

→ dataset 10 K Records
I have to train my model for 20 epochs.

$$Loss = \frac{1}{2} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$ (n = 10K records)

Everyone P2 focus

f_1 f_2 y

In Every epoch 10k records will be take

y_1
y_2
y_{10k}

Front

back

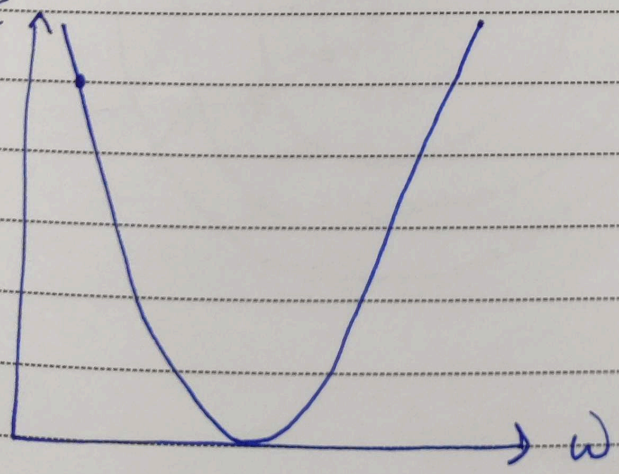
{Epoch Complete dataset is taken}

what if my dataset becomes 1 million record:→

1 million

→ huge RAM
→ weight updatation will take time
→ Convergence will be slow

Loss



More Resource Computationally expensive

So what do we do?

Stochastic G.D.

Instead taking all records

take only 1 record: →

do Front →

back ←

$Loss = \frac{1}{2}(y - \hat{y})^2$

For every epoch (10K)

I have to do 10K iterations

↘

↙

20 epoch.

→ 20 × 10K.

↓

Convergence is slow

Less computationally expensive

Minibatch SGD: →

10k records

for every epoch · Researches take

Some batch size

say 1000

20 Epoch 10

Epoch (1)

how many iterations

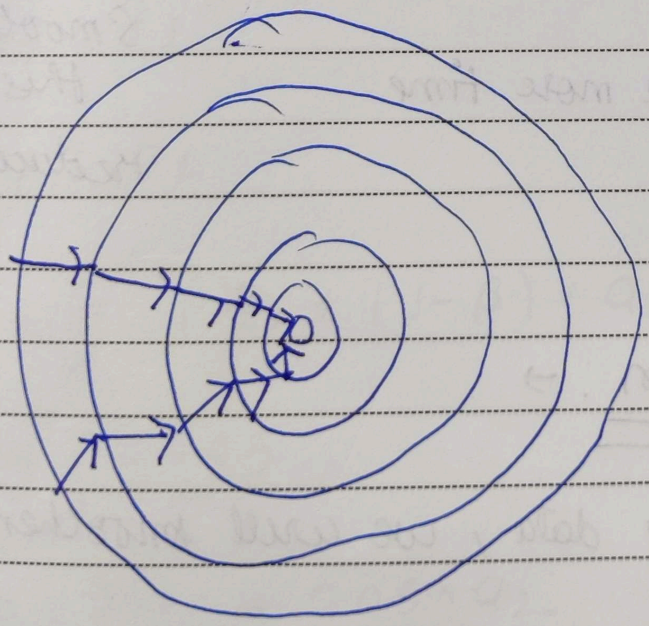
$\frac{10k}{k}$ = 10 iterations

10 Forward and

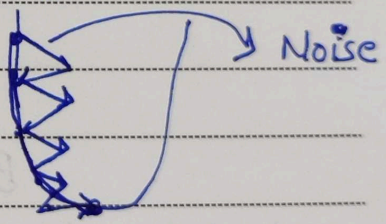
10 backward propagation

Epoch (2)

→ 10 iteration.



mini batch SGD



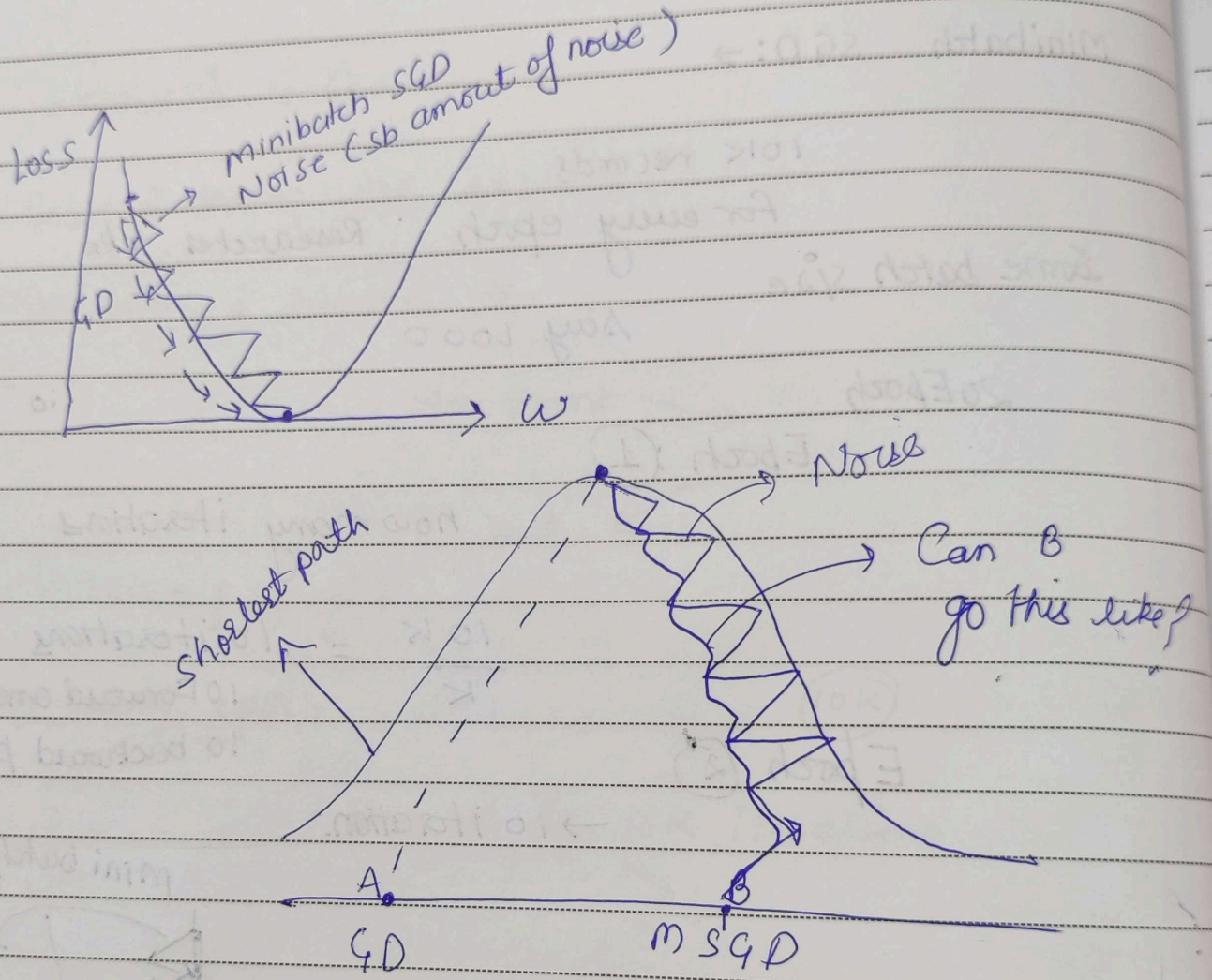
GD Whole dataset

SGD 1 Record

Mini Batch → Batch size

6:30 PM

~~7:00~~ PM



B will take more time

Sb GPS to B

Smoothing this

reducing Noise

SGD with momentum :→

Noisy data, we will smoothen

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

$$w_t = w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}}$$

momentum says: in order to smoothen, we will compute expo. weighted average.

Expoential weighted Average :→

t_1	t_2	t_3	t_4	---	t_η	date time series data
a_1	a_2	a_3	a_4	--	a_n	

$$V_t = a_1$$

$$V_{t_1} = a_1$$

$$V_{t_2} = \beta V_{t_1} + (1-\beta) * a_2$$

Say $\beta = .95$

β is hyperparameter how much importance you want to give to previous data

$$= .95 a_1 + 0.05 * a_2$$

ARIMA ⇒ Smoothening concept

$\beta = .1$

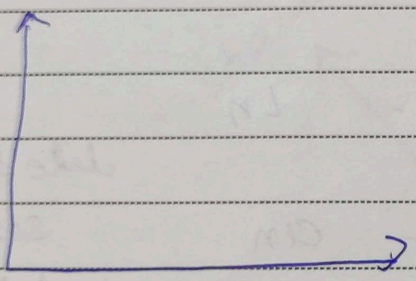
$.1 a_1 + .9 a_2$

Give more importance to a_2 (next value)

$$V_{t_3} = \beta V_{t_2} + (1-\beta) a_3$$

$$= \beta(\beta V_{t_1} + (1-\beta) a_2) + (1-\beta) a_3)$$

Exponetial avg.



$\Rightarrow$

Exponential weighted average

$$w_t = w_{t-1} - \eta V_{dw}$$

↑ exponential weighted avg.

$$b_t = b_{t-1} - \eta V_{db}$$

$$V_{dw_t} = \beta V_{dw_{t-1}} + (1-\beta) \times \frac{\partial L}{\partial w}$$

$V_t = a_1 =$
$\beta V_t + (1-\beta) a_2$

Implementation details

Initially $V_{dv} = 0$ $V_{db} = 0$

On iteration ~~and~~ inside epoch

Compute dw, db ~~:~~ on the current : mini batch

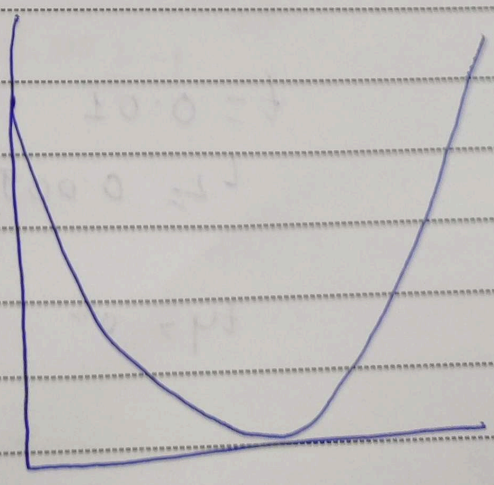
$$V_{dw_t} = \beta V_{dw_{t-1}} + (1-\beta) \frac{\partial L}{\partial w_{t-1}}$$

$$V_{db_t} = \beta V_{db_{t-1}} + (1-\beta) \frac{\partial L}{\partial b_t^{-1}}$$

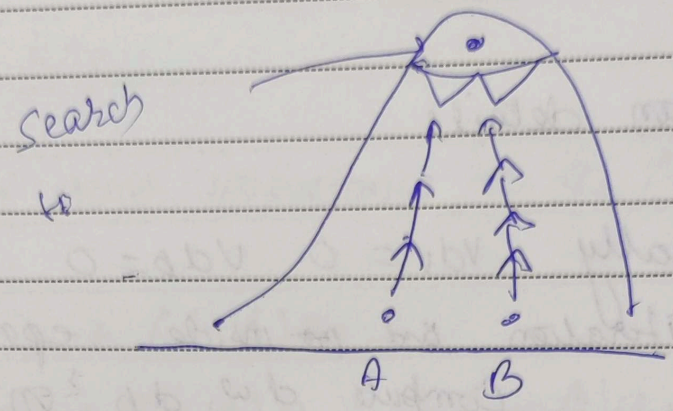
$$w = \beta - \eta V_a V_{dw}$$
$$b = b - \eta V_{db_t}$$

(4) Adaptive ~~Adpa~~ gradient descent-

η = fixed



N



weight updation formula

$$w_t = w_{t-1} - \eta \frac{\partial L}{\partial old}$$

$$w_t = w_{t-1} - \eta \frac{\partial L}{\partial}$$

η ← initial learning rate

$$\eta'_t = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}}$$

$$= \frac{\eta}{\sqrt{\sum_{j=1}^{k} + \varepsilon}}$$

Good one

$t = 0.01$

$t_2 = 0.009$ $t_3 = \infty$

$t_4 = \infty$

$\gamma_t \sum_{i=1}^{t} \left(\frac{\partial L}{\partial w_t}\right)^2$

divide by big no. η' will reduce

we need to reduce learning rate

disadvantage; After some time
dt is very very big no
if deep N.N

{.01} starting
{.0000001 (learning rate) Now}

. disadvage:-
γ_t is becoming very very high

(5) and
Adadelta ~~with~~ RMS(ROP)

weight updation formula

$\eta_t = \eta$

$N \; S_{dw} + \varepsilon$

$= \beta$

$S_{dw_t} = \beta S_{dw_{t-1}}$

Adam

Adapative Moment Action Adaptive

Momentum
RMSPROP } Combine

V_{dw} V_{db} S_{dw} S_{db}

$\frac{\partial w}{\partial w}$, $\frac{\partial L}{\partial b}$ using current mini batch

$$V_L = \beta_1 V_{dw} + (1-\beta)\frac{dh}{\partial w}$$

$$V_{dw} = \beta_1 V_{ow} + (1-\beta)$$

$$S_{db} = \beta_L S_{db} + (1-\beta)\left(\frac{\partial L}{\partial w}\right)^2$$

$$S_{db} = \beta S_{db}$$

Bias Correction

initial LR

$$w_t = w_{t-1} - \eta \times \frac{V_{dw}}{\sqrt{S_{dw} + \epsilon}}$$

$$b_t = b_{t-1} - \eta \times \frac{V_{db}}{\sqrt{S_{db} + \epsilon}}$$

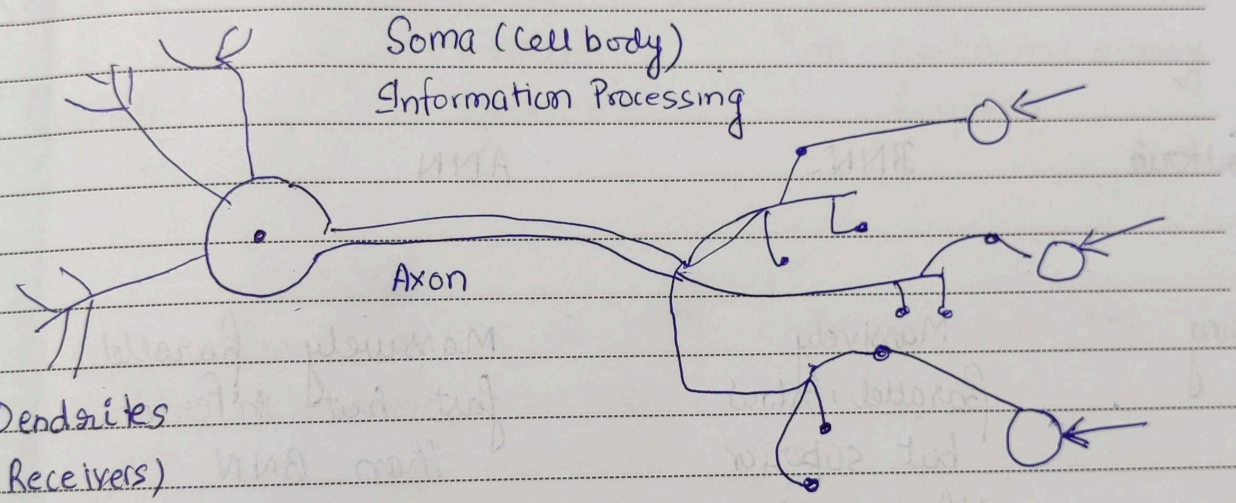
Adam

→ Smoothing factor
→ RMSProp

Biological Neuron

A nerve cell neuron is a special biological cell that processes information.

According to an estimation, there are huge number of neurons, approximately 10^{11} with numerous interconnections, approximately 10^{15}



Dendrites :-

Soma :-

Axon :-

Synapses

ANN Vs BNN

BNN	ANN
Soma	Node
Dendrites	Input
Synapse	Weights or interconnections
Axon	Output

Criteria	BNN	ANN
Processing	Massively parallel, slow but superior than ANN	Massively parallel, fast but inferior than BNN
Size	10^{11} Neurons and 10^{15} ~~Con~~ nter Connections	10^2 to 10^4 nodes
Learning	they can tolerate ambiguity	Very precise, structured and formatted data is required to tolerate ambiguity.

Fault tolerance	Performance degrades with even partial damage	it is capable of robust performance, hence has the potential to be fault tolerant.
Storage Capacity	Stores the information in synapse	Stores the information in continuous memory locations.

Processing of ANN depends upon the following 3 building blocks

- ↳ Network Topology
- ↳ Adjustments of weights or learning
- ↳ Activation Functions

Network Topology is the arrenagement of a network along with its nodes and connecting lines. According to the topology. ANN can be classified as

Feedforward Network

Single layer feed forward Network

Multilayer feedforward Network

Feedback Network :→

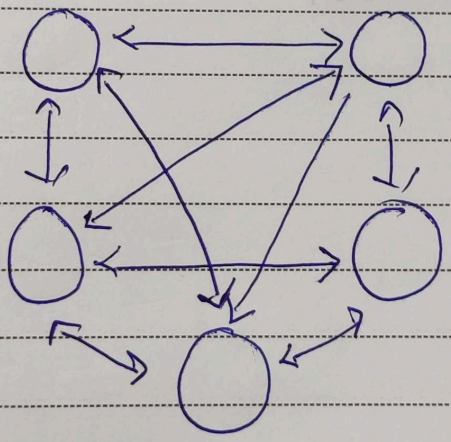
a feedback network has feedback paths, which means the signal can flow in both directions using loops. following types

Recurrent Networks

2 types.

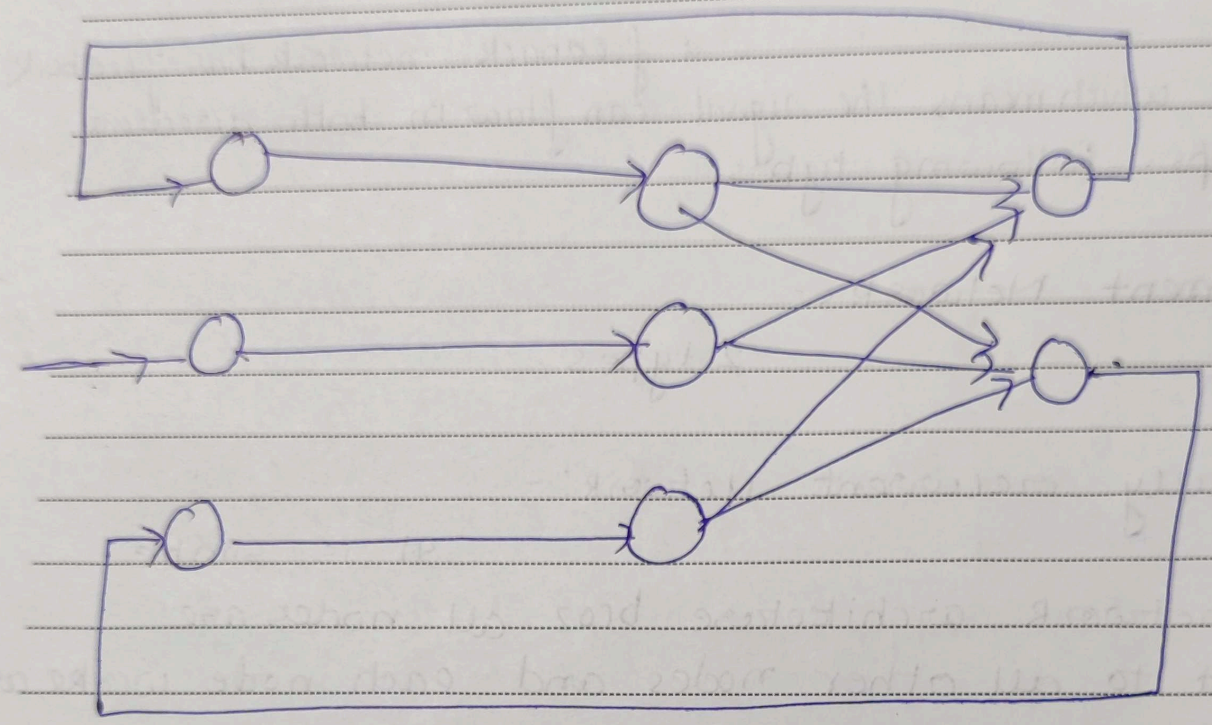
Fully recurrent Network :-

It is simple neural network architecture bcoz all nodes are connected to all other nodes and each node works as both input and output.



Jordan Network

It is closed loop network in which the output will go to the input again as feedback as shown



Inputs Hidden Outputs

← Adjustments of weights or learning: →

Learning in artificial neural network, is the method of modifying the weights of connections between the neurons of a specified network. Learning can be classified into 3 categories: →

Supervised learning
Unsupervised "
Reinforcement "

Activation function: -

Activation function

Advantages:→

1. ANN has the ability to learn and model non linear and complex relationships as many relationships between input and output are non linear.

2. After training, ANN can infer unseen relationships from unseen data, and hence it is generalised.

3. Unlike many machine learning models, ANN does not have restrictions on datasets data should be Gaussian dist^n or any other dist.

Applications

1. Image Processing and Character Recognition
2. Forecasting
3. Credit Rating
4. Fraud Detection
5. Portfolio Management

```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
```

```
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
```

Compiling keras model:

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Fitting the keras model

```
model.fit(X, y, epochs=150, batch_size=10)
```

(5) Evaluate Keras Model

```
accuracy = model.evaluate(X, y)
print("Accuracy: %.2f%% of predictions")
```

(6) Make Predictions.

```
predictions = model.predict(X)
rounded = [round(x[0]) for x in predictions]
```

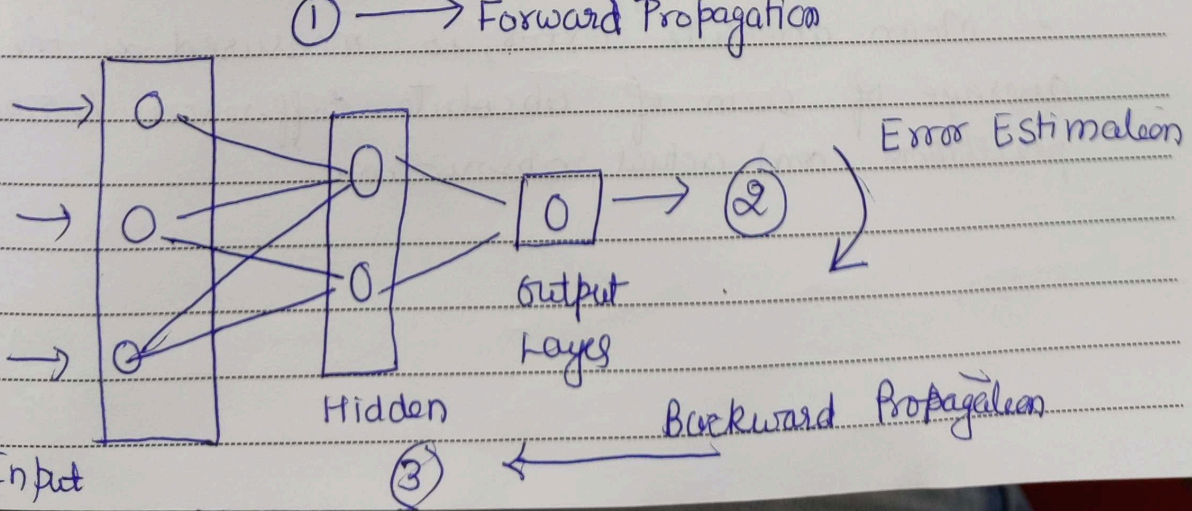
Loss function and its type:

Loss helps us to understand how much the predicted value differ from actual value

Function used to calculate the loss is called as "Loss function".

Loss function is a method of evaluating "how well your algorithm models your dataset".
If your predictions are totally off, your loss function will output a higher number.
If they are pretty good, it will output a lower number. As you tune your algorithm to try and improve your model, your loss function will tell you if you are improving or not.

LOSS FUNCTION are helpful to train a neural Network.



Types of Loss function

1. Regression Loss Function
2. Binary Classification Loss functions
3. Multi class Classification Loss Functions

1) Regression Loss Function

Regression models deals with predicting a continuous value for example given floor area, no of rooms, size of rooms, predict the price of room. Loss function used in regression problem are called "Regression Loss Function".

Most used Regression Loss function are below:→

1.1 Mean Absolute Error (MAE) / L1 Loss function :

- Mean absolute error, is measured as the average of sum of absolute differences between predictions and actual observations.

- It measures the average "magnitude" of errors in a set of predictions, "without considering their directions".
- Robust to outlier.
- MAE output ranges between 0 and ∞
- Derivatives of MAE are not continuous, making it inefficient

MAE is also known as "least Absolute Deviations" or "LAD".

$$\text{Loss} = \frac{\sum_{i=0}^{n} |y_i - \hat{y}_i|}{n}$$

MAE LOSS

Mean Square Error (MSE) / L2 Loss Function:-

Mean square error, is measured as the avg of sum of squared differences between predictions and actual observations.

It measures the average "magnitude" of errors in a set of predictions, "without considering their "directions".

Less robust to outliers

MSE loss is stable than MAE.

L2 Loss ranges between 0 and ∞.

Derivatives of MSE are continuous, making it effecient to find the ~~solution~~ solution.

"MSE is the most commonly used regression loss function"

$$\text{Loss} = \frac{\sum_{i=0}^{n} (Y_i - \hat{Y}_i)^2}{n}$$

1.3 Root Mean Square Error (RMSE) / RMS Error :-

Root mean square error is the extension of MSE - measured as the average of square root of sum of squared differences between predictions and actual observations.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

1.4 Mean Bias Error : →

MBE is less commonly used in practice.

MBE is similar

Hebbian learning rule, perception learning rule
Delta learning rule, Widrow Hoff Learning rule
Correction learning rule, winner take all
learning all.

Supervised learning
Unsupervised learning
Reinforcement learning

L1 Fundamentals of Neural Networks

McCulloch and Pitts → L6

Supervised learning
classification
regression

L2 Perceptrons

Linear classification

Hebbian learning and Associative Memory

L3 L4 Feed Forward multiple layer networks and Backpropagation

Perception

Unsupervised learning

L5 Recurrent Neural Networks (RNN)

Sequence and temporal data

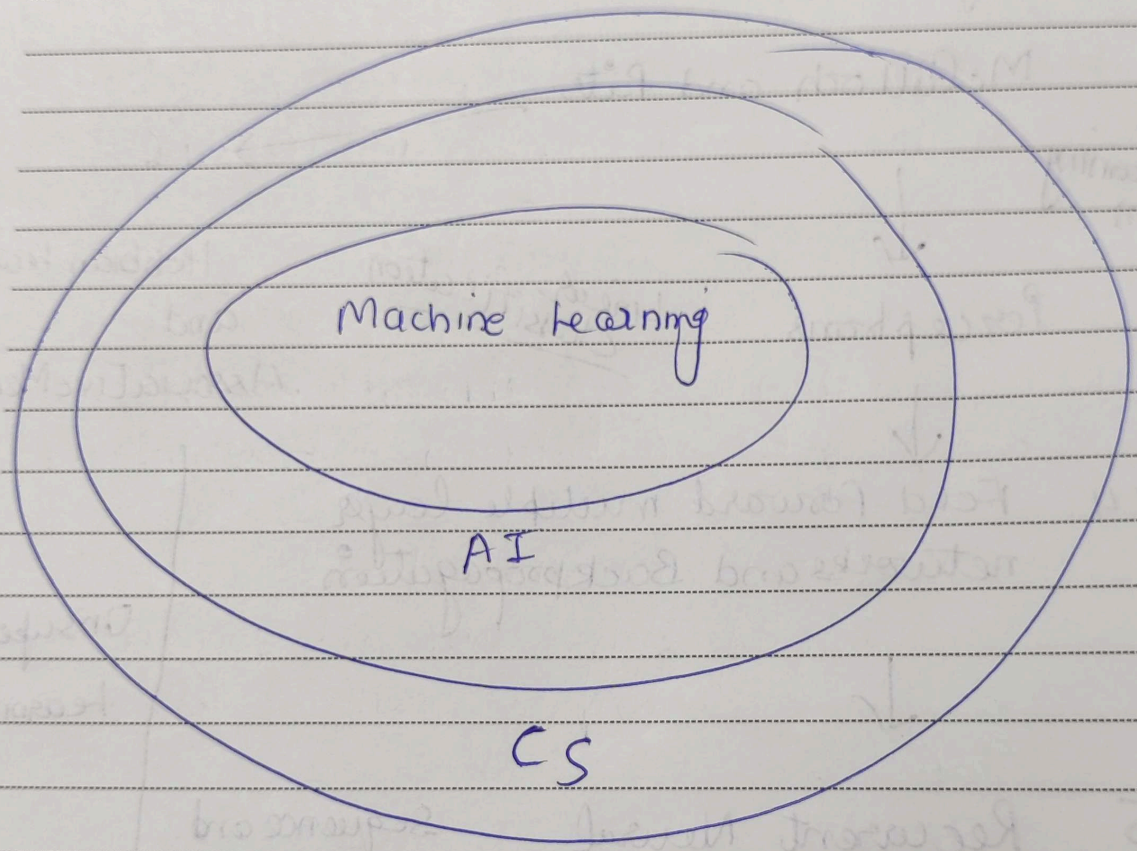
L8 Convolutional Neural Networks (CNN)

L7 Hopfield Networks and Boltzman machines

Deeplearning

L9

Development of ANN field


Machine learning
AI
CS

Perceptrons

The perceptron is an algo for learning a linear binary classifier in the form of a threshold function that maps its input X, a real valued vector, to a single output binary value Y.

In the context of neural networks, a perceptron is a single artificial neuron using the Heaviside step function as the activation fn.

As a linear classifier, the single layer perceptron is the simplest feed forward neural Network.

y_j = if $W * X = \text{Sum } W_{ij} * X_{ij} > 0$ then 1
otherwise 0
$i = 1 -- N$

W is a real valued vector, j ≠ of iteration

The binary value of f(x) is used to classify a data item as either a positive or a negative instance.

Intuitions for Perceptron learning

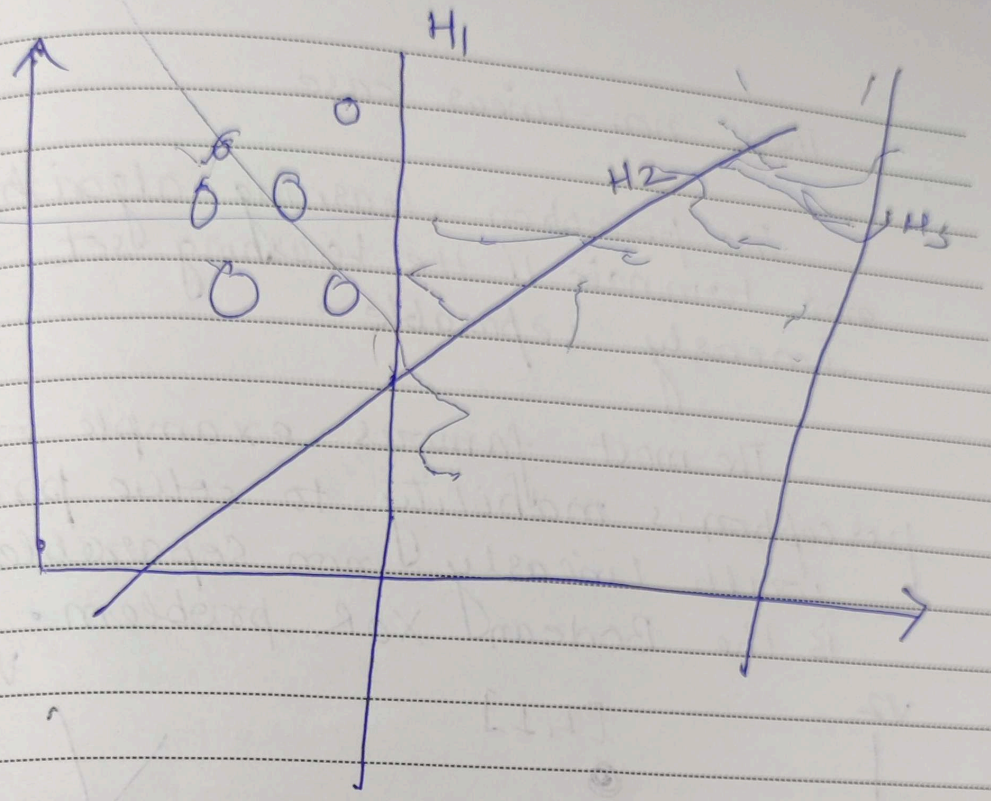
A perceptron is a linear classifier that tries to find a hyperplane in the space spanned by its input

The adjustments of weights corresponds to changes in orientation of hyper plane.

The adjustments of bias corresponds to changes in the hyper plan intercepts with axes of the space

For a 2 dimensional space the perceptron corresponds a line in the plane, the orientation of which is decided by the weights and the intercepts of which are decided by the bias

An optimal perceptron finds a hyperplane (eg line) that best separates the data items.



Which problems can a perceptron solve?

= Binary Classification for linearly separable instance spaces.

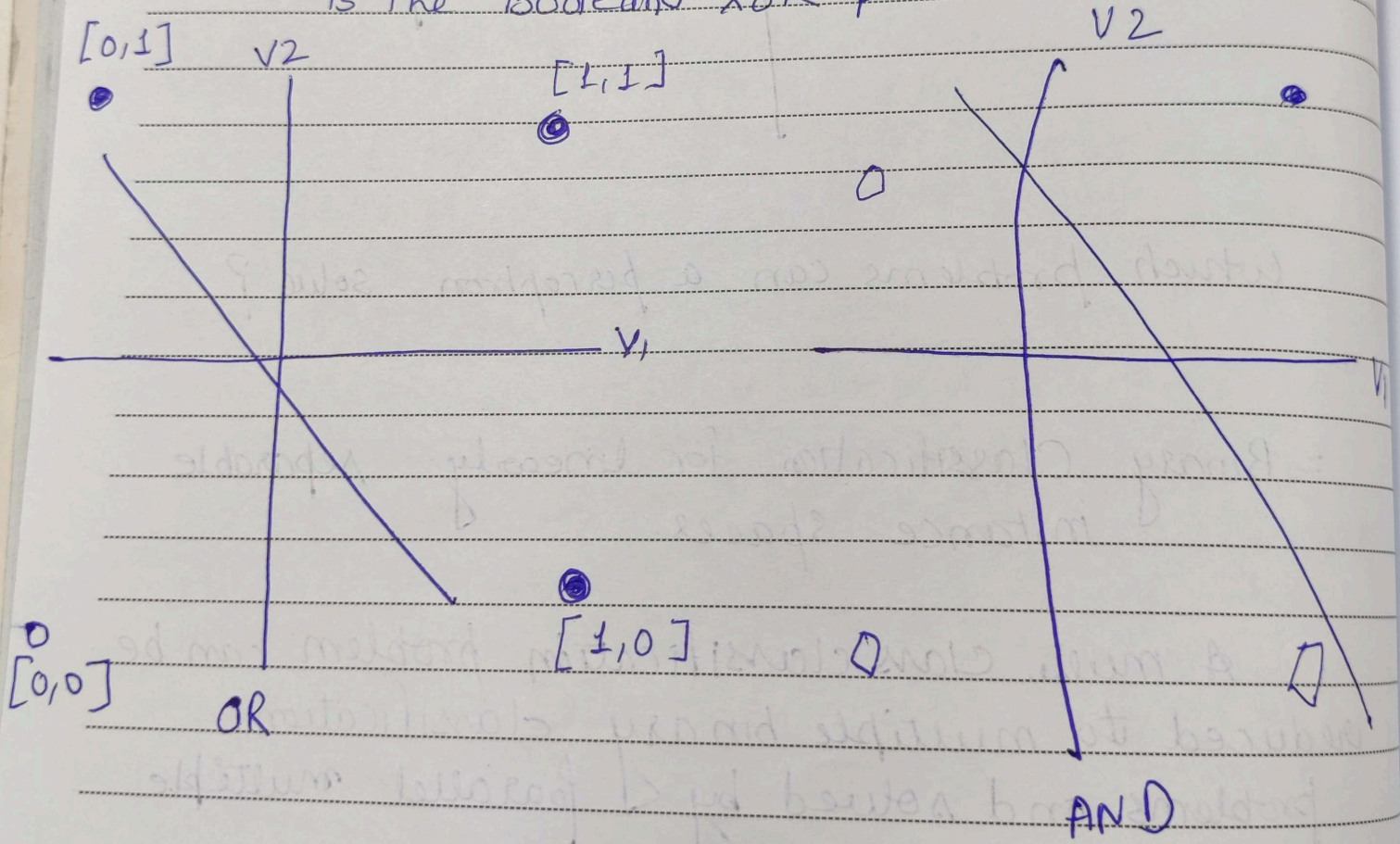
A multi class classification problem can be reduced to multiple binary classification problems and solved by parallel multiple perceptrons.

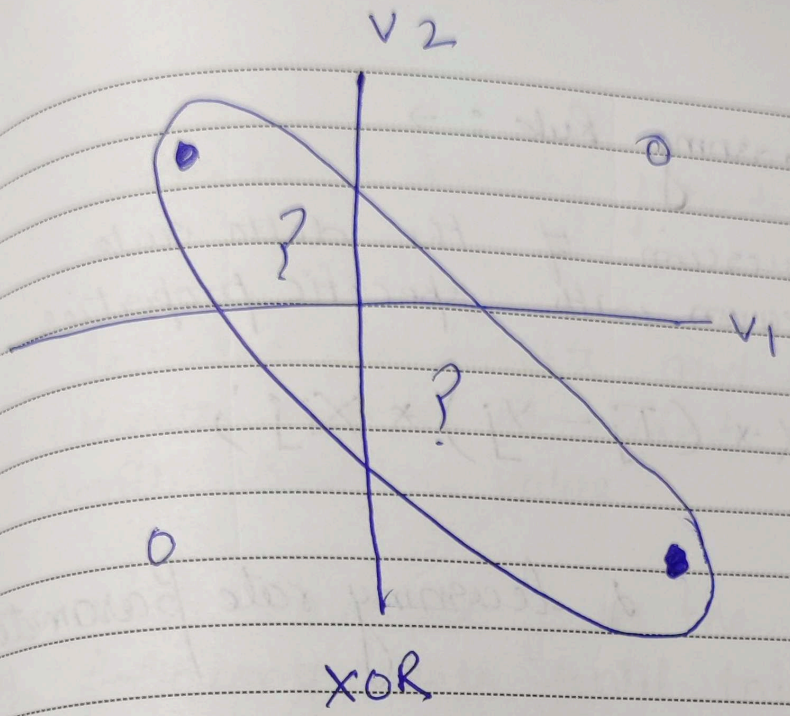
As a perceptron only has binary output, regression cannot be performed.

Then non-Linear case

The perceptron learning algorithm does not terminate if the learning set is not linearly separable.

The most famous example of the perceptron's inability to solve problems with linearly non separable vectors is the Boolean XOR problem.





XOR

Perceptron learning Rule :→

A simpler version of the delta rule for ONE neuron with specific properties

$$W_{ij+1} = W_{ij} + \alpha * (T_j - Y_j) * X_{ij},$$

α learning rate parameter

$Y_j = X_{ij} W_{ij}$, i = input, j = iteration.

T = target

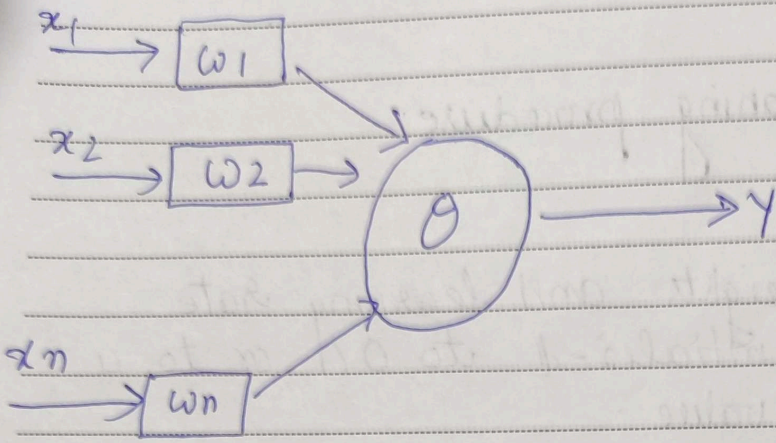
The threshold can also be learned by transforming it to a bias = − threshold and completing the data items with an extra input $X_{0j} = 1$ and corresponding weight W_{0j} = bias

steps in a learning procedure:-

Initialize the weights and learning rate
Weights may be initialized to 0 or to a small random value.

For each example j in the dataset, perform the following steps until total training set error ceases to improve.

→ Calculate the output
→ Calculate the new weights



$Y_j = 1$ if $W * X + B = \left(\text{Sum } w_{ij} * x_{ij}\right) > \theta$

$i = 1 \text{---} n$

Otherwise $Y = 0$

$W_{ij+1} = w_{ij} + \alpha * (T_j - Y_j) * X_{ij}$ $\alpha = 0.2$

delta weight

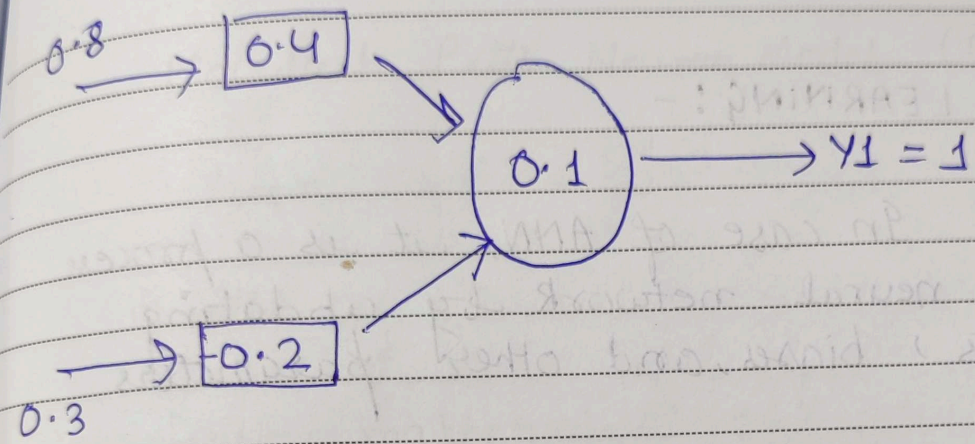
$X11 =$

$\alpha = 0.2$

$X_{11} = 0.8$ $T_1 = 0$

$X_{21} = 0.3$ $w_{11} = 0.4$

$w_{21} = -0.2$



$.4 \times .8 + (-0.2 \times 0.3)$

$.32 - .06$

$= 0.26 > .1 \longrightarrow Y_1 = 1$

$w_{12} = .0.4 + 0.2 * (0-1) * .0.8$

$= 0.24$

$w_{22} = -0.2 + .2 * (0-1) * 0.3 = -0.26$

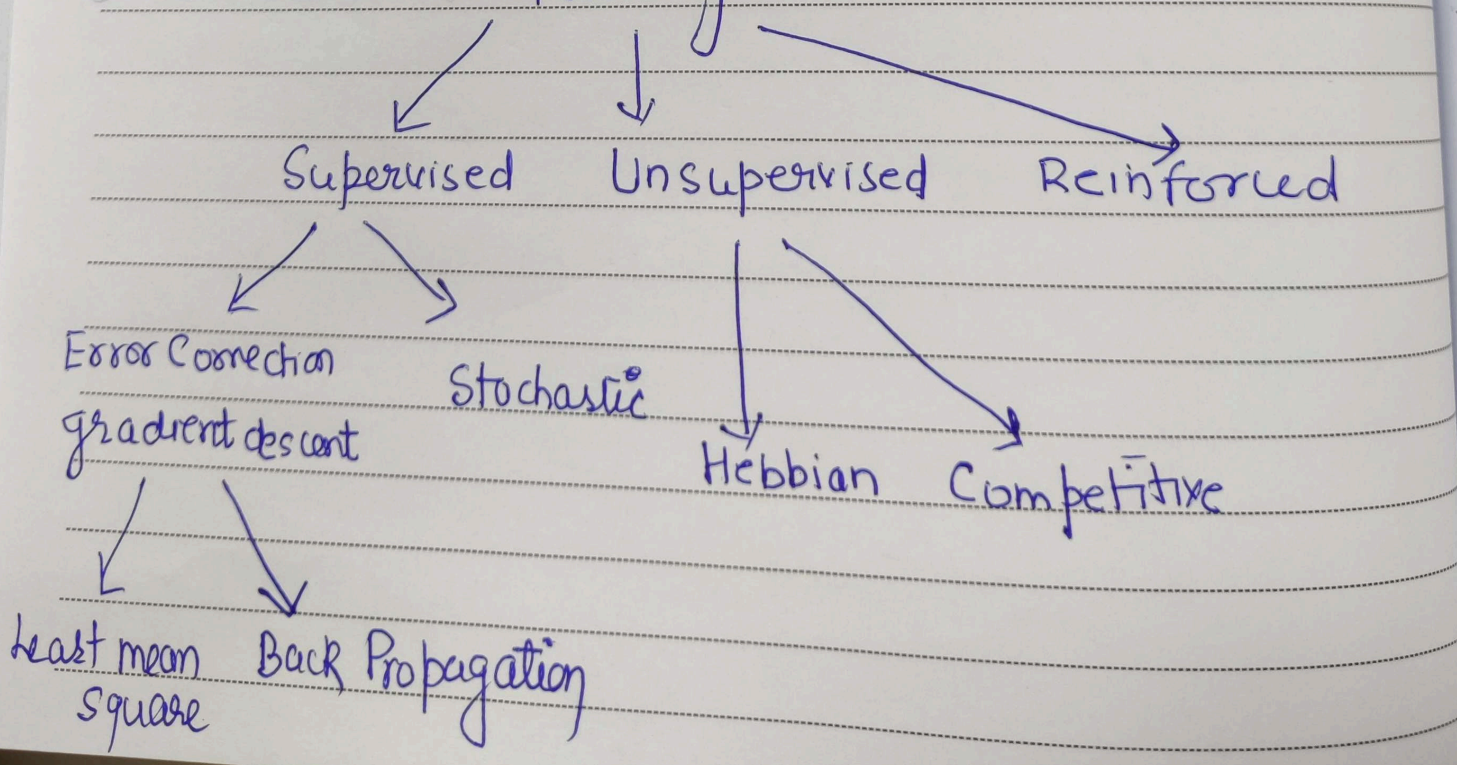
CONCEPT OF LEARNING :-

In case of ANN, it is a process of modifying neural network by updating its weights, biases, and other parameters if any.

During the learning, the parameters of the networks are optimized and as a result process of curve fitting.

It is then said that the network has passed through a learning phase.

Types of learning



McCulloch Pitts Neuron Model (MCP)

Linear threshold Gate Model

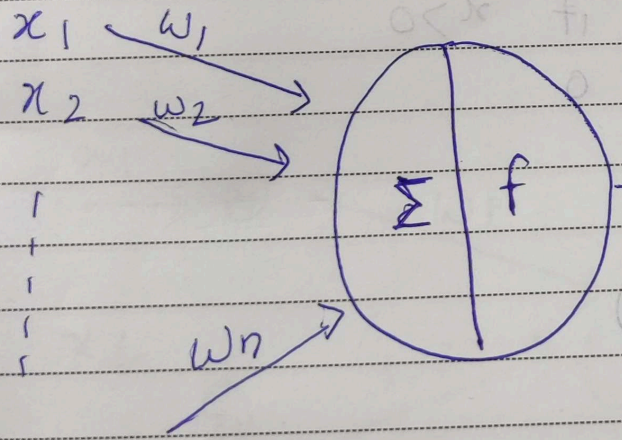
in 1943 by Warren McCulloch and Walter Pitts

First mathematical model of biological neuron

Basic building block of neural n/w

Directed (Graph) - weight graph is used for connecting neurons

Two possible states of neuron → Active (1)
→ Silent (0)



Σ it aggregates the weighted input into single numeric value.

$\Sigma x_1 w_1 + x_2 w_2 \text{ --- } + x_n w_n$] X

f produces the o/p using threshold (T)

o/p 1 if $X > T$

0 if $X < T$

BIAS / Threshold:

Minimum value of weighted active input for a neuron to fire.

if effective i/p is larger than T, then o/p → 1 Else 0

o/p

f(a)

$a \cdot \Sigma \omega_i x_i \rightarrow T$

$Q(x) = 1$ if $x > 0$

0

excitatory (+ve)

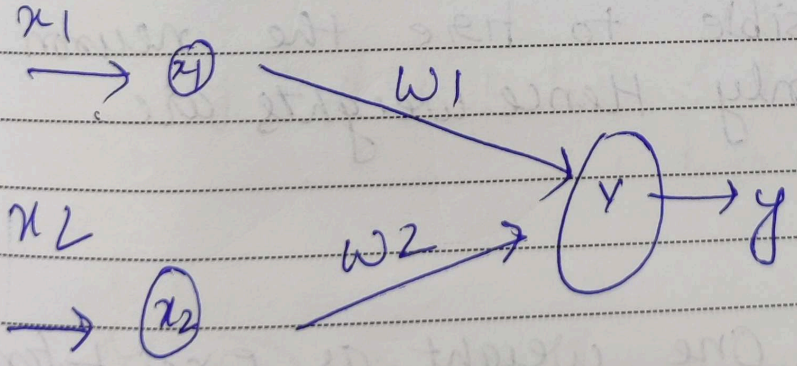
inhibitory (-ve)

$\theta \geq nw - p$

Implement AND NOT function using McCulloch-Pitts neuron (Binary data)

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

AND	NOT
0	0 0
0	1 0
1	0 0
1	1 1



Case 1: Assume both weights are excitatory

$w_1 = w_2 = 1$

$\theta \geq nw - p$

$\geq 2 \times 1 - 0$

$\theta \geq 2$

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

for inputs

(0, 0) $y_{in} = 0 \times 1 + 0 \times 1 = 0$

(0, 1) $y_{in} = 0 \times 1 + 1 \times 1 = 1$

(1, 0) $y_{in} = 1 \times 1 + 0 \times 1 = 1$

(1, 1) $y_{in} = 1 \times 1 + 1 \times 1 = 2$

$$f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geq 2 \\ 0 & y_{in} < 2 \end{cases}$$

It is not possible to fire the neuron with i/p (1,0) only. Hence weights are not suitable

Case 2 : Assume one weight as excitatory and other as inhibitory

$w_1 = 1$, $w_2 = -1$

(0,0) $y_{in} = 0 \times 1 + 0 \times -1 = 0$

(0,1) $= 0 \times 1 + 1 \times -1 = -1$

(1,0) $= 1 \times 1 + 0 \times -1 = 1$

(1,1) $1 \times 1 + 1 \times -1 = 0$

It is possible to fire the neuron with i/p (1,0) only

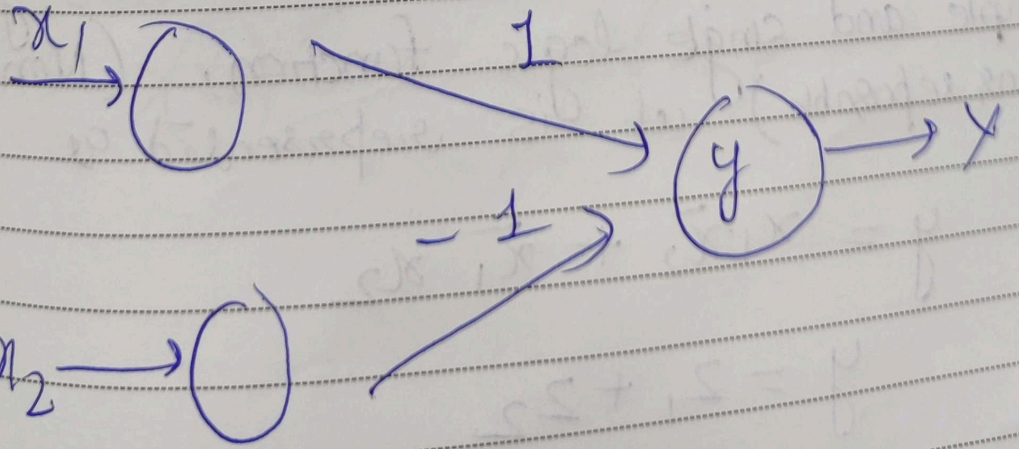
$\theta \geq nw - p$

$2 \times 1 - 1$

$\theta \geq 1$

o/p of Neuron

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



PN-1

Implement XOR function using McCulloch - Pitts Neuron

x_1	x_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Consider the truth table for XOR function

The MP neuron has no particular training algo.

In MP neuron, only analysis is being performed

XOR function cannot be represented by simple and single logic function, (Non Linear separable) it is represented as

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

$$y = z_1 + z_2$$

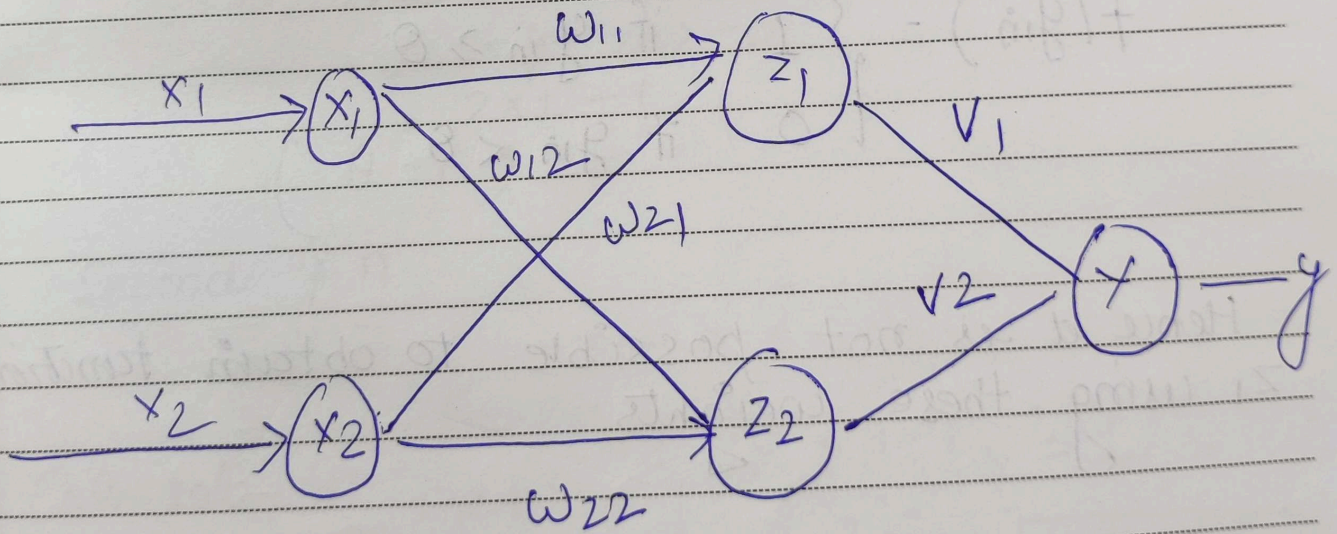
where

$z_1 = x_1 \bar{x}_2$ function 1

$z_2 \bar{x}_1 x_2$ function 2

$y = z_1 \text{ OR } z_2$ function 3

A single layer net is not sufficient to represent the XOR function. We need to add an intermediate layer is necessary



③

First function

$z_1 = x_1 \bar{x}_2$

truth table for function z_1

	1	0	1

$w_{11} = w_{21} = 1$

(0,0) z_{1in} $0 \times 1 + 0 \times 1 = 0$

(0,1) $0 \times 1 + 1 \times 1 = 1$

(1,0) $1 \times 1 + 0 \times 1 = 1$

(1,1) $1 \times 1 + 1 \times 1 = 2$

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

Hence it is not possible to obtain function z_1 using these weights

Assume the weights are initialized to

$w_{11} = 1, \quad w_{21} = -1$

Calculate the net inputs,

(0,0) $z_{1in} = 0 \times 1 + 0 \times -1 = 0$

(0,1) $z_{1in} = 0 \times 1 + 1 \times -1 = -1$

(1,0) $z_{1in} = 1 \times 1 + 0 \times -1 = 1$

(1,1) $z_{1in} = 1 \times 1 + 1 \times -1 = 0$

What should be threshold value

$\theta = np \cdot nw - p$

$2 \times 1 - 1$

$(\theta = 1)$

Second fn

$z_2 = \bar{x}_1 x_2$

Assume $w_{12} = w_{22} = 1$

Calculate net inputs

(0,0)	0	x
(0,1)	1	✓
(1,0)	1	✓
(1,1)	2	✓

} fire in 3 cases

0	1	1

(5)

$w_{12} = -1$, $w_{22} = 1$

Calculate the net inputs

(0,0) $= 0 \times -1 + 0 \times 1 = 0$

(0,1) $= 0 \times -1 + 1 \times 1 = 1$

(1,0) $= 1 \times -1 + 0 \times 1 = -1$

(1,1) $= 1 \times -1 + 1 \times 1 = 0$

$\theta = 2 - 1$

$= 1$

$\theta = 1$ then neuron fires

Third function

$y = z_1 \text{ (OR) } z_2$

x_1	x_2	y	z_1	z_2
0	0	0	0	0
0	1	1	0	1
1	0	1	1	0
1	1	0	0	0

truth table

$y_{in} = z_1 v_1 + z_2 v_2$

Assume the weights are initialed to

$v_1 = v_2 = 1$

Calculate the net inputs

(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	0

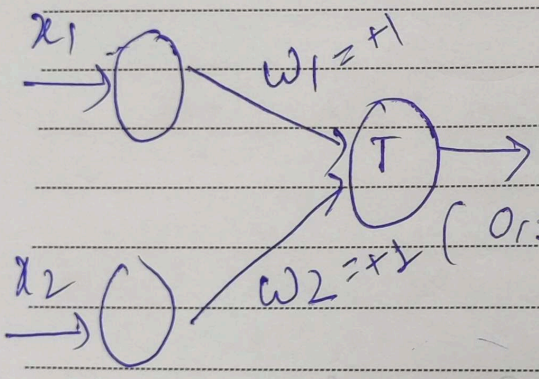
$$f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geq \theta \\ 0 & y_{in} < \theta \end{cases}$$

$\theta = 1$, then the neuron fires

(7)

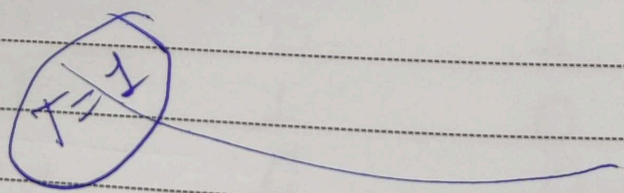
OR Gate

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



$w_1 = +1$, $w_2 = +1$

$(0,0) = 0 \times 1 + 0 \times 1 = 0$
$(0,1) = 0 \times 0 + 1 \times 1 = 1$
$(1,0) = 1 \times 1 + 0 \times 0 = 1$
$(1,1) = 1 \times 1 + 1 \times 1 = 2$



Can we put threshold = 1 and get final answer

AND gate

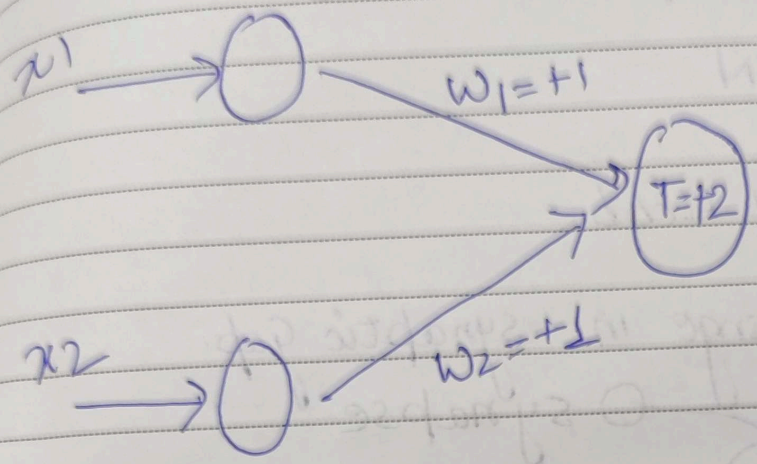
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

$w_1 = +1$, $w_2 = +1$

we should set threshold ? so that above condetions are satisfied.

so

$T \geq 2$

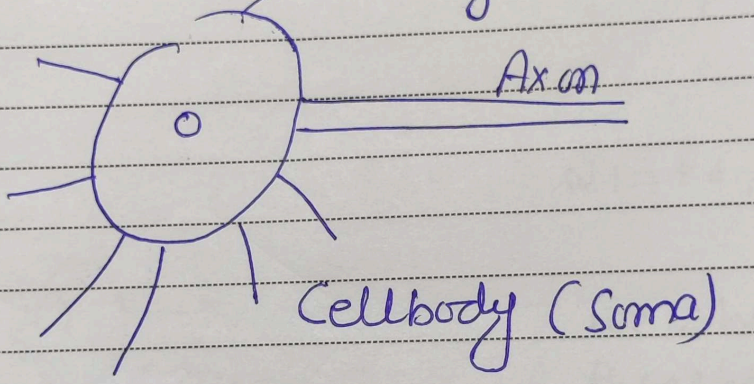

x1
w1=+1
T=+2
x2
w2=+1

Hebb Network / Hebbian rule

ANN

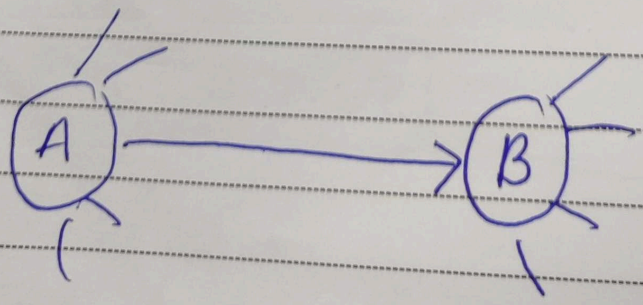
In 1949, Donald Hebb

⇒ learning ⇒ change in synaptic gap.



Hebb rule:

When an axon of cell A is near enough to excite cell B repeatedly / permanently firing it, then growth / metabolic change takes place in both of the cells.



- unsupervised algo
Pattern association, pattern categorization, and pattern classification

Hebb law

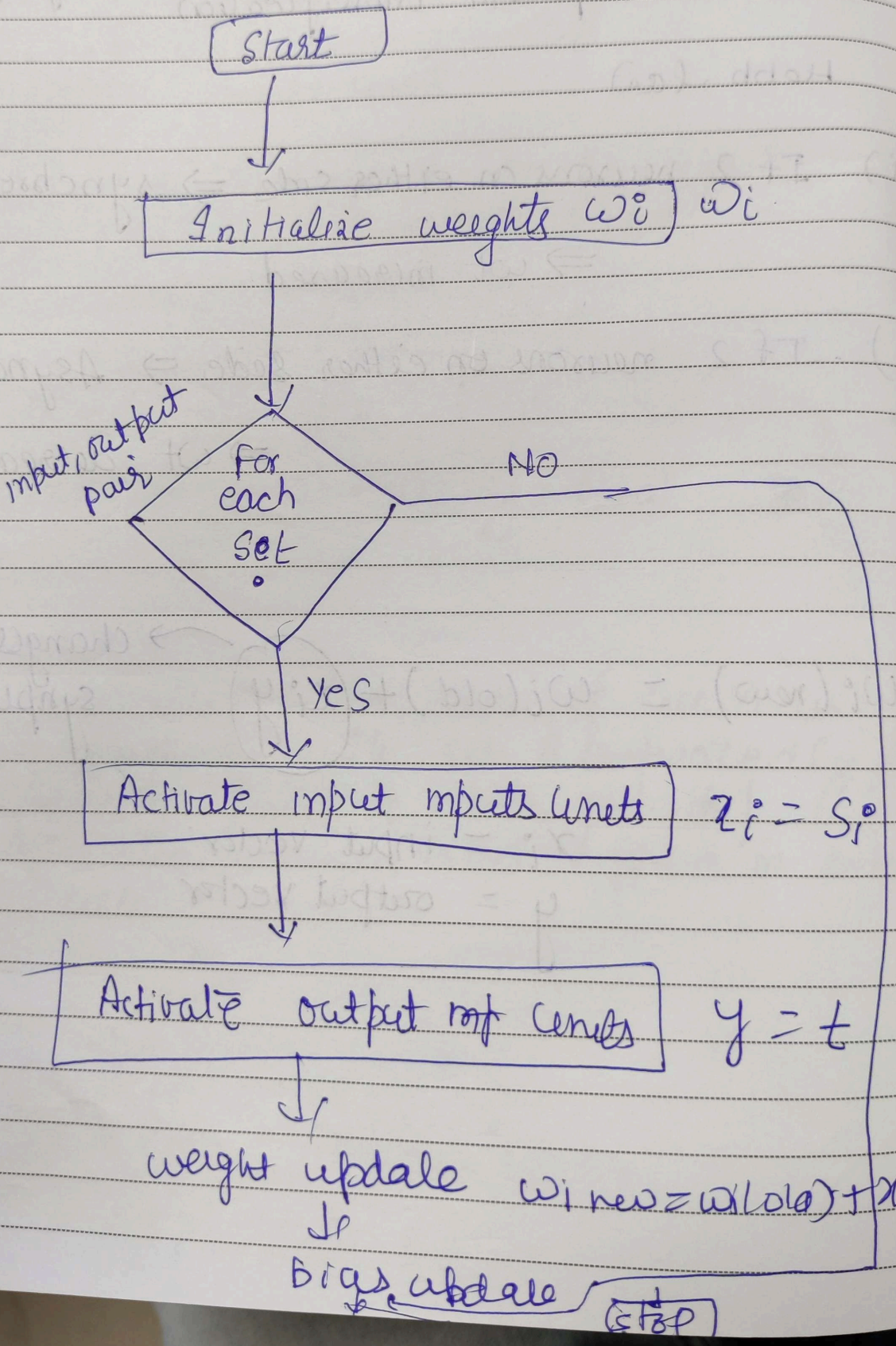
i) If 2 neurons on either side ⇒ synchronously
⇒ wt increased

ii) If 2 neurons on either side ⇒ Asynchronously
⇒ wt decreased

$$w_i(new) = w_i(old) + (x_i y)$$ → changes synaptic gap

x_i = input vector
y = output vector

Flowchart


Start
Initialize weights wi
wi
input, output pair
for each set
NO
YES
Activate input inputs units
xi = si
Activate output net units
y = t
weight update
wi new = wi(old) + xiy
Bias update
STOP

Training Algo

Step 0: Initialize the weights and bias

Step 1 For each training i/p and o/p pair perform step 2 to 4

Step 2 $x_i = s_i$ (activation)

Step 3 $y = t$ (activation)

Step 4 wt updation

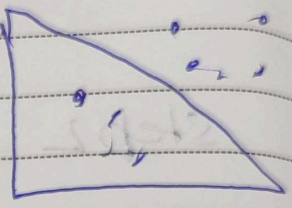
$$w_i(new) = w_i(old) + x_i y$$

$$b(new) = b(old) + y$$

Linear Separability

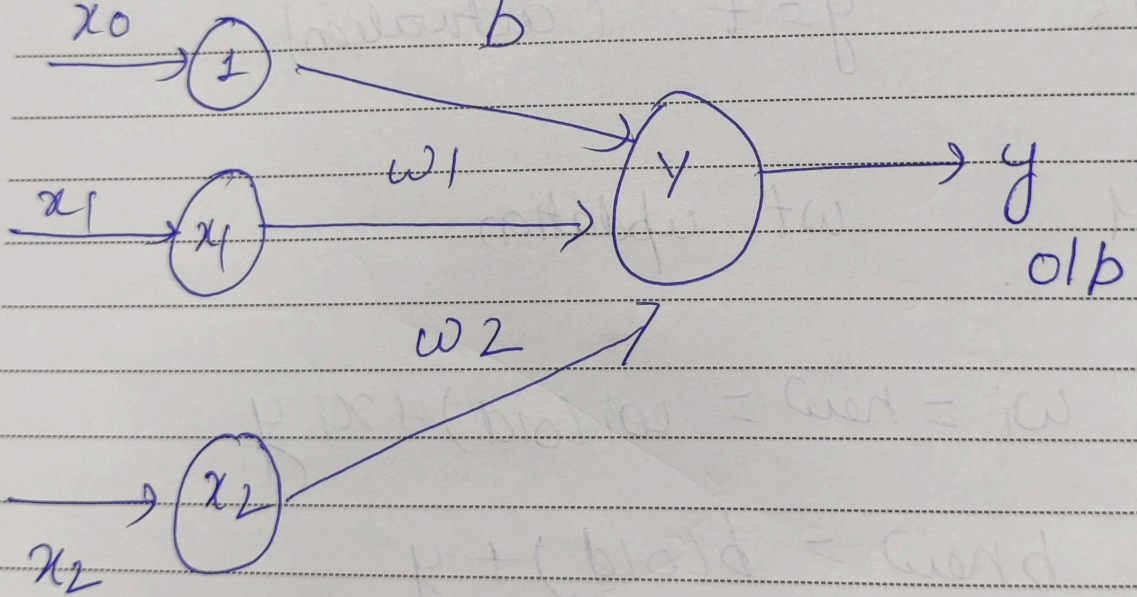
A decision line is drawn to separate positive and negative responses.

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



Requirement for the response of the net is

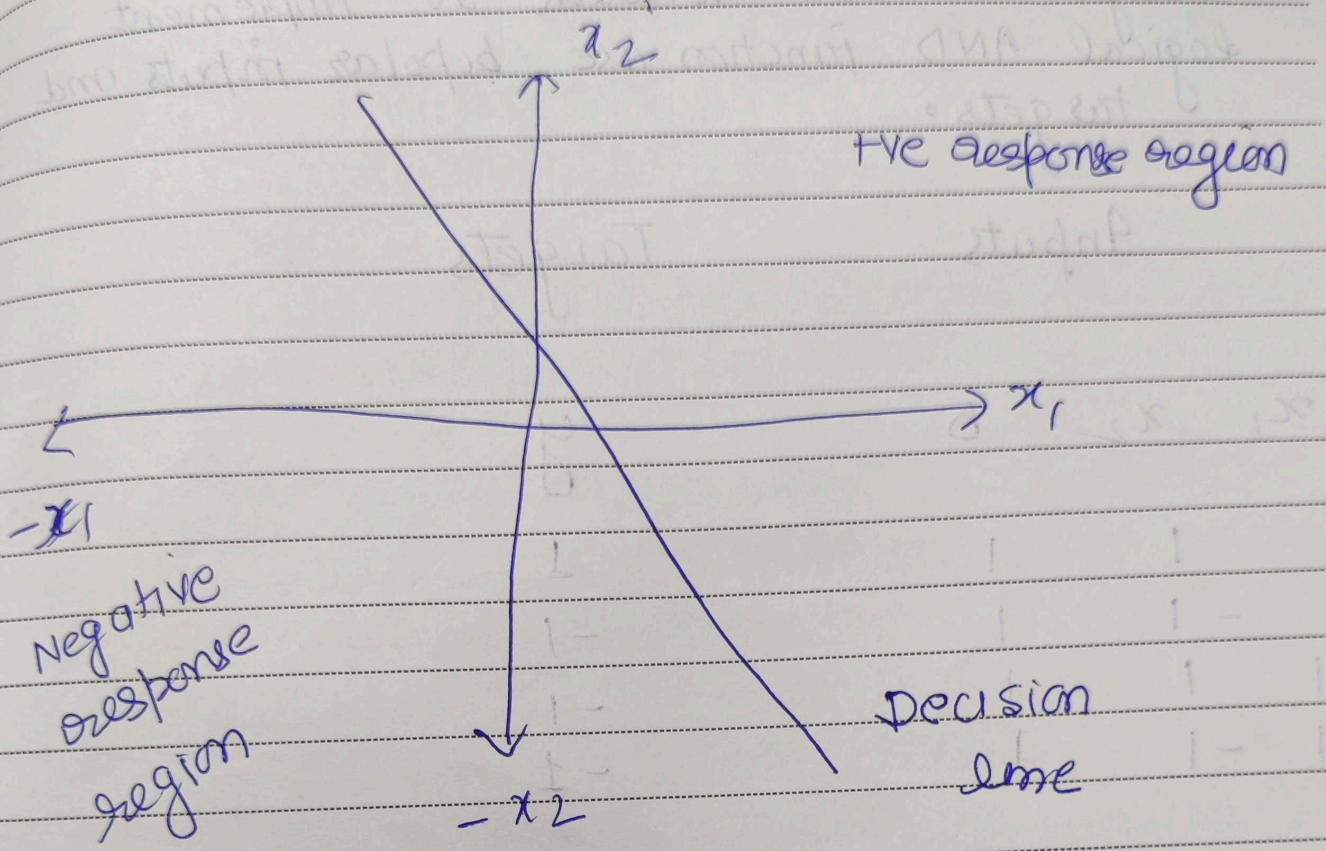
$$b + \sum_{i=1}^{n} x_i w_i = 0$$



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

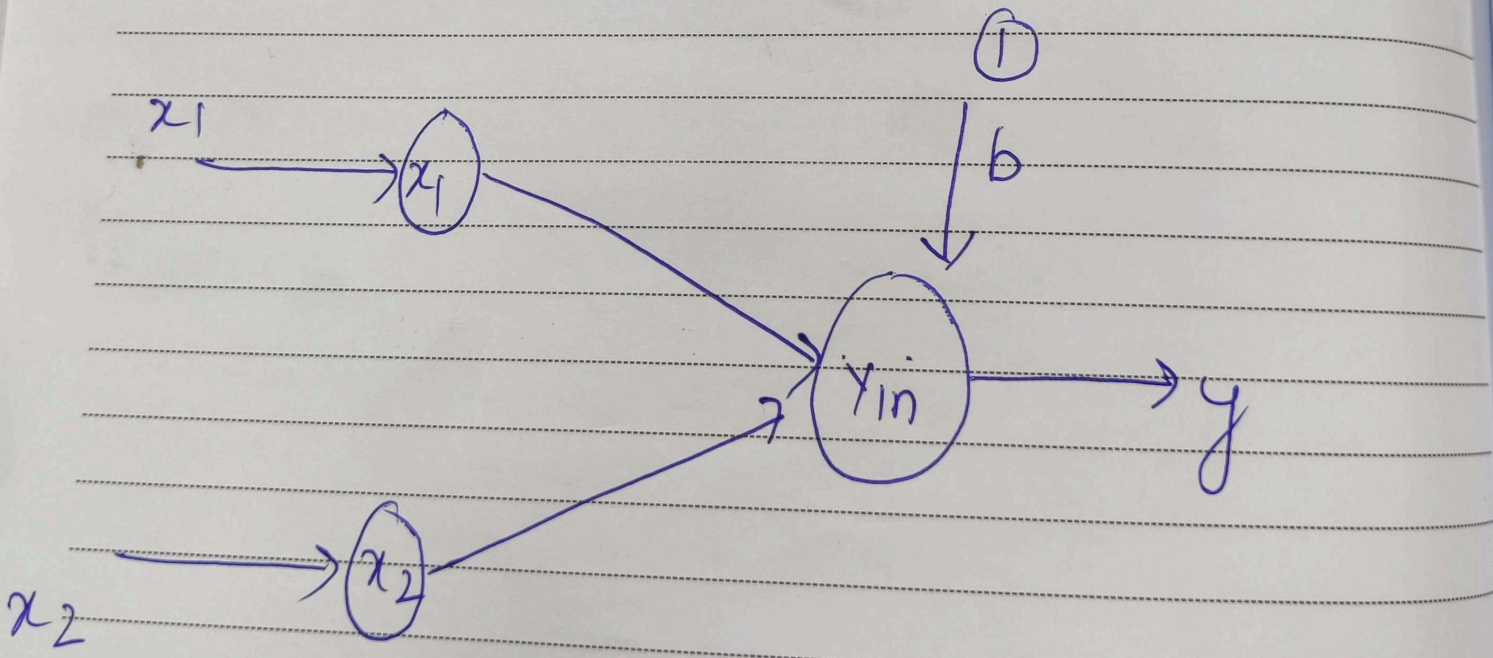
$$x_2 = -\frac{w_1}{w_2}$$

$$x_1 = -\frac{b}{w_2}$$


x_2
+ve response region
x_1
$-x_1$
Negative response region
Decision line
$-x_2$

Q. Design a Hebb network to implement logical AND function use bipolar inputs and targets.

Inputs			Targets
x_1	x_2	b	y
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1



$w_i(new) = w_i(old) + \Delta w$

$x_i y$

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

$$w_1 = w_2 = b = 0$$

We have to train

Inputs			y	weight changes			New weight (0 (0) (0)		
				Δw_1	Δw_2	Δb	$w_1(new)$	$w_2(new)$	$b(new)$
x_1	x_2	b	y	$x_1 y$	$x_2 y$	y	$w_1(old) + \Delta w_1$	$w_2(old) + \Delta w_2$	$b(old) + y$
1	1	1	1	1	1	1	0+1 = 1	0+1 = 1	0+1 = 1
1	-1	1	-1	-1	1	-1	1+(-1) = 0	1+1 = 2	1+(-1) = 0
-1	1	1	-1	1	-1	-1	0+1 = 1	2+(-1) = 1	0+(-1) = -1
-1	-1	1	-1	1	1	-1	1+1 = 2	1+1 = 2	-1+(-1) = -2

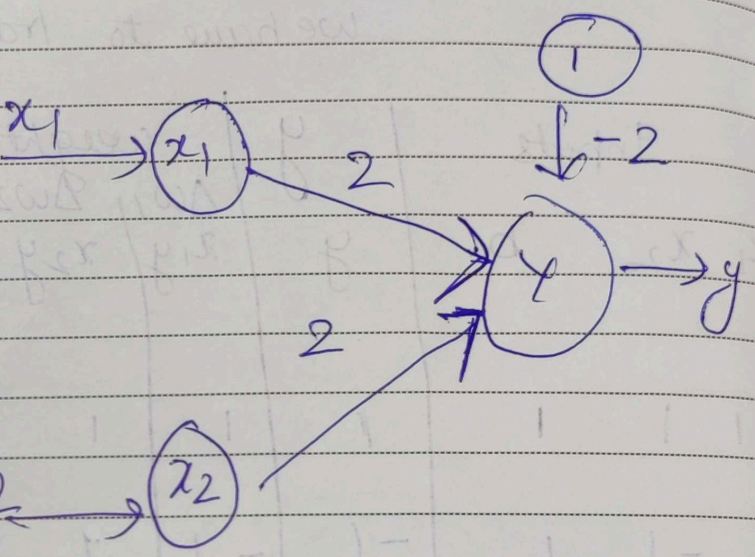
Final weights are

$w_1 = 2$

$w_2 = 2$

$b = -2$

x_1	x_2	y
1	1	1
1	-1	-1
-1	1	-1
-1	-1	1



(1, 1)

$y = b + x_1 w_1 + x_2 w_2$

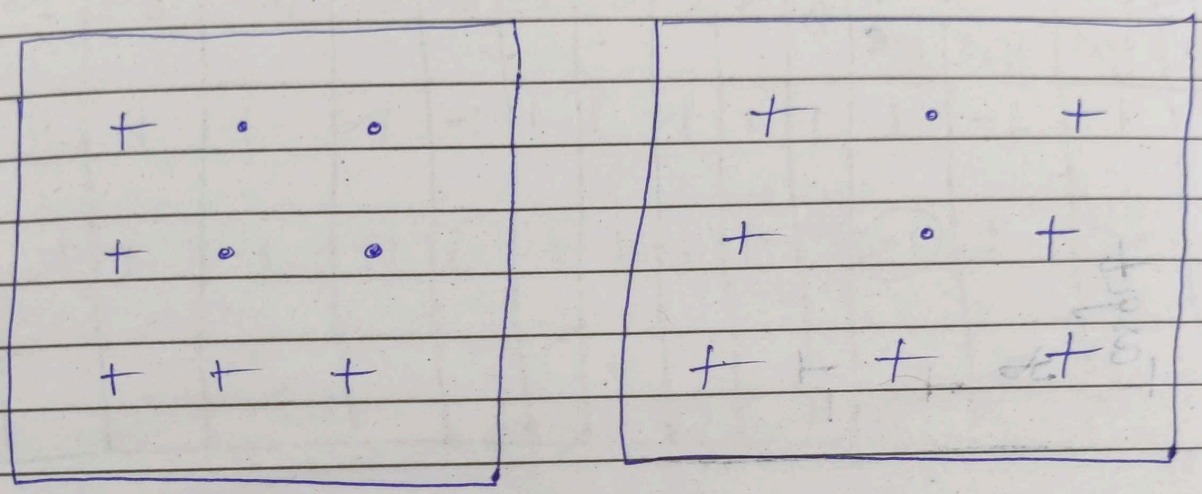
$= -2 + (1)(2) + 1(2)$

$= -2 + 2 + 2$

$y = 2$

Pair

Design a Hebb Network, find the weights required to perform the classification of the given input pattern.



'L'

'L' = 1

'U'

'U' = 0)

The pattern is shown as 3×3 matrix form in the squares. The "+" symbol represent the value "1" and empty squares indicates "-1". Consider "L" belongs to class so target value is 1 and "U" does not belong to the members of class (so target value is 0)

Page No.
Date:

Pattern	Inputs									Target
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	y
L	1	-1	-1	1	-1	-1	1	-1	-1	1
U	1	-1	1	1	-1	1	1	1	1	-1

Step 1

w_1

x_1	x_2	x
1	-1	
1	-1	

w_1
(0)
$w_1(0+0) + \Delta w_1$
$0+1=1$
$1+-1=0$

Step 1

$w_1 = w_2 = w_3 = w_4 = w_5 = w_6 = w_7 = w_8 = w_9 = 0$

$b = 0$

Inputs											weight changes			
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	b	y	Δw_1	Δw_2	Δw_3	Δw_4
											$x_1 y$	$x_2 y$	$x_3 y$	$x_4 y$
1	-1	-1	1	-1	-1	1	1	1	1	1	1	-1	-1	1
1	-1	1	1	-1	1	1	1	1	1	-1	-1	1	-1	-1

Δw_5	Δw_6	Δw_7	Δw_8	Δw_9	Δb
$x_5 y$	$x_6 y$	$x_7 y$	$x_8 y$	$x_9 y$	
-1	-1	1	1	1	1
1	-1	-1	-1	-1	-1

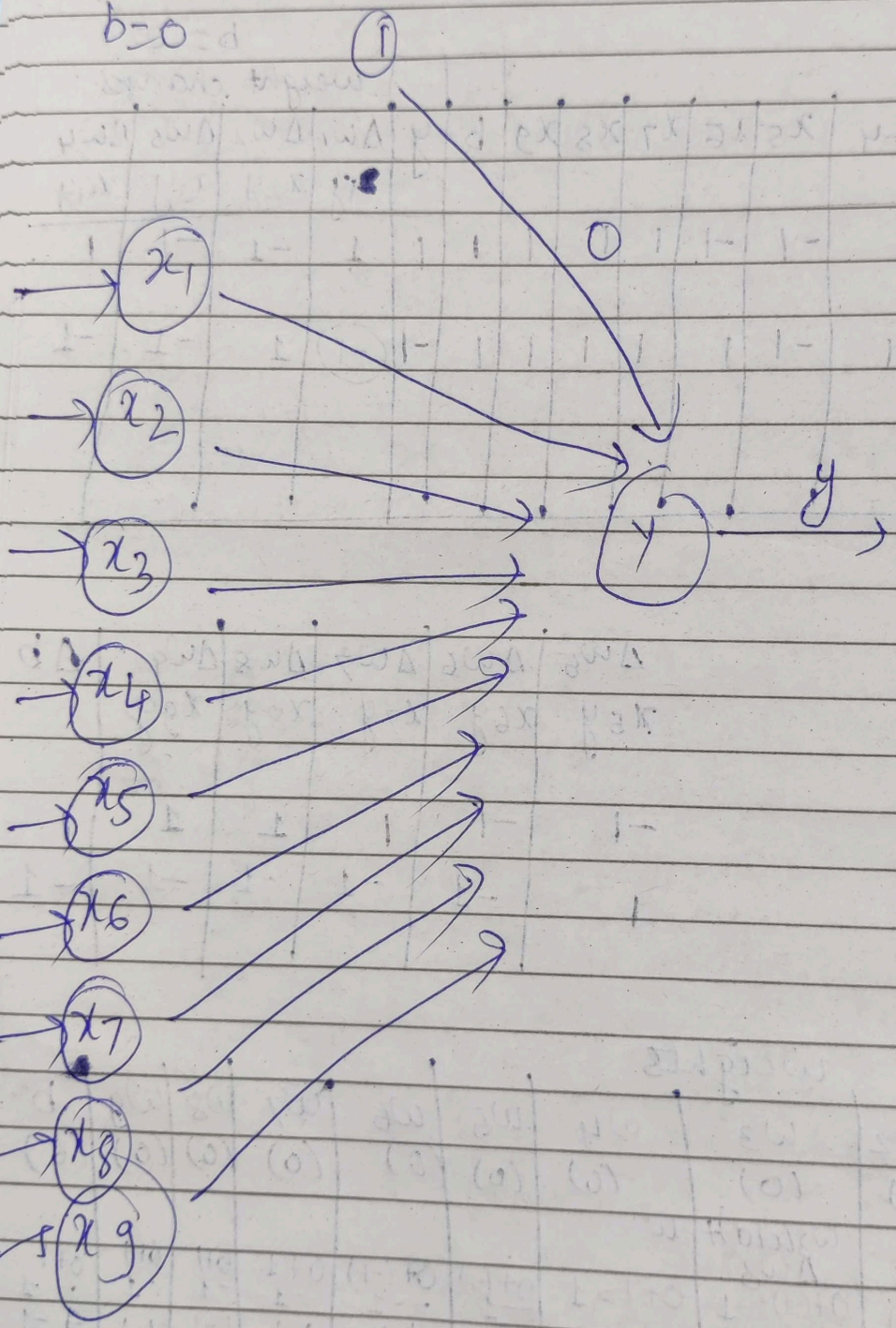
New weights

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	b
(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)
$w_1(old) + \Delta w_1$	$w_2 old + \Delta w_2$	$w_3(old) + \Delta w_3$	w						
$0+1=1$	$0+(-1)=-1$	$0+(-1)=-1$	$0+1=1$	$0+(-1) = -1$	$0+(-1) = -1$	$0+1 = 1$	$0+1 = 1$	$0+1 = 1$	$0+1 = 1$
$1+-1=0$	$-1+1=0$	$-1-1 = -2$	$1-1=0$	$-1+1 = 0$	$-1-1 = -2$	$1-1 = 0$	$1-1 = 0$	$1-1 = 0$	$\exists -1$

Page No.
Date:

wnew $\begin{bmatrix} 0 & 0 & -2 & 0 & 0 & -2 & 0 & 0 & 0 & 0 \end{bmatrix}$

$b = 0$



Date:

1 -1 -1 1 -1 -1 1 1 1 1 b=0

1x0 + -1x0 + -1x-2 + 1x0 + -1x0 + -1x-2
+ 1x0 + 1x0 + 1x0 + 0

y = (4) ✓

[1 -1 1 1 -1 1 1 1 1 1] second set
bias=0

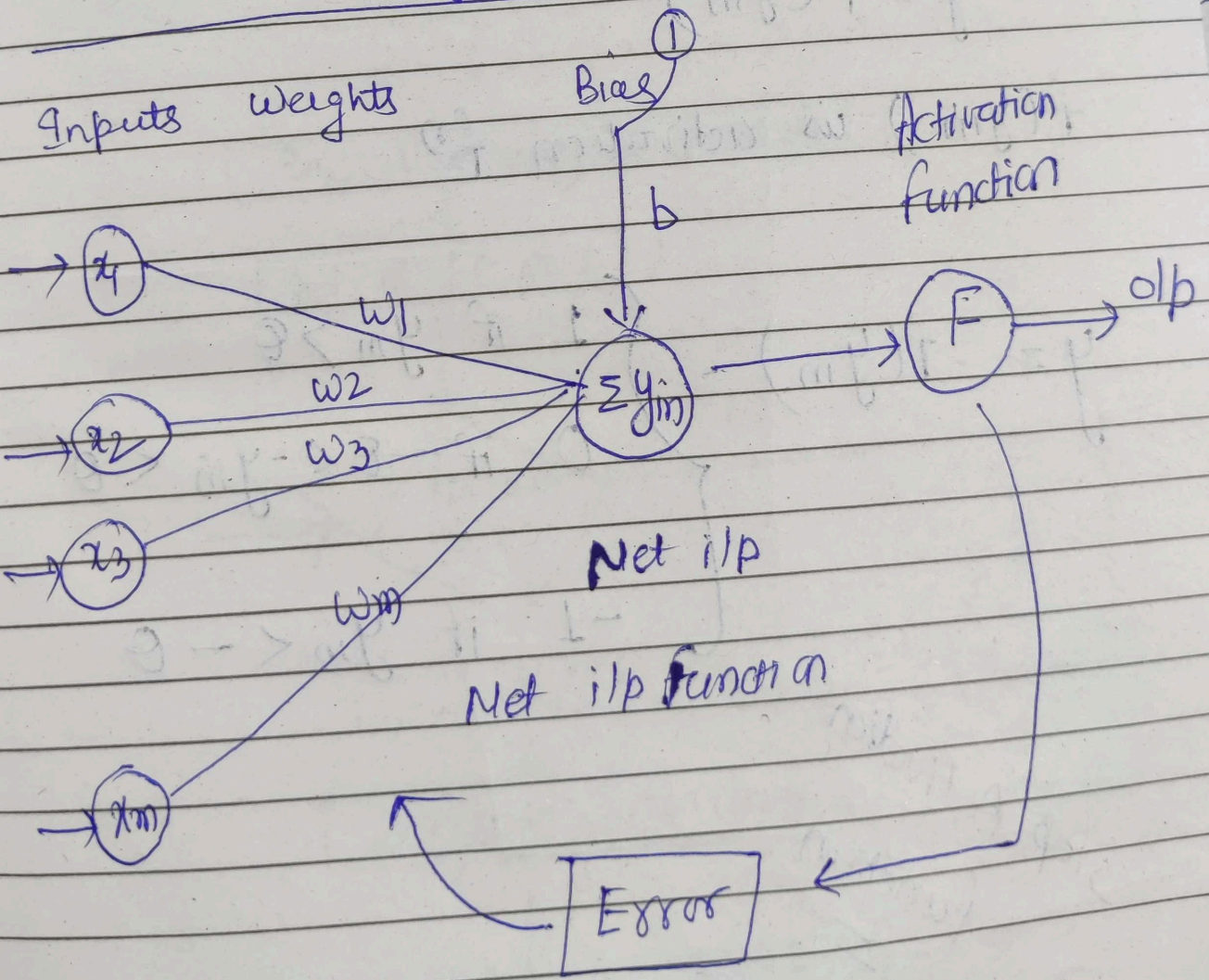
y = -2-2 = -4 (-ive value

Supervised learning Network

→ Perceptron networks come under single layer feed forward networks →
simple Perceptrons

→ Simplest form of Neural Network & is able to classify data into two classes.

→ Introduced by Frank Rosenblatt in 1957



Page No.
Date:

2 types of Perceptrons

i) Single layer
ii) Multi Layer — 2 or more layers

$$y_{in} = x_1 w_1 + x_2 w_2 \; - - \; x_n w_n + b$$

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

$f(y_{in})$ is activation fn

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

Step - Activation function

Page No.
Date:

the weight updation

if $y \neq t$ then

$w(new) = w(old) + \alpha t x$

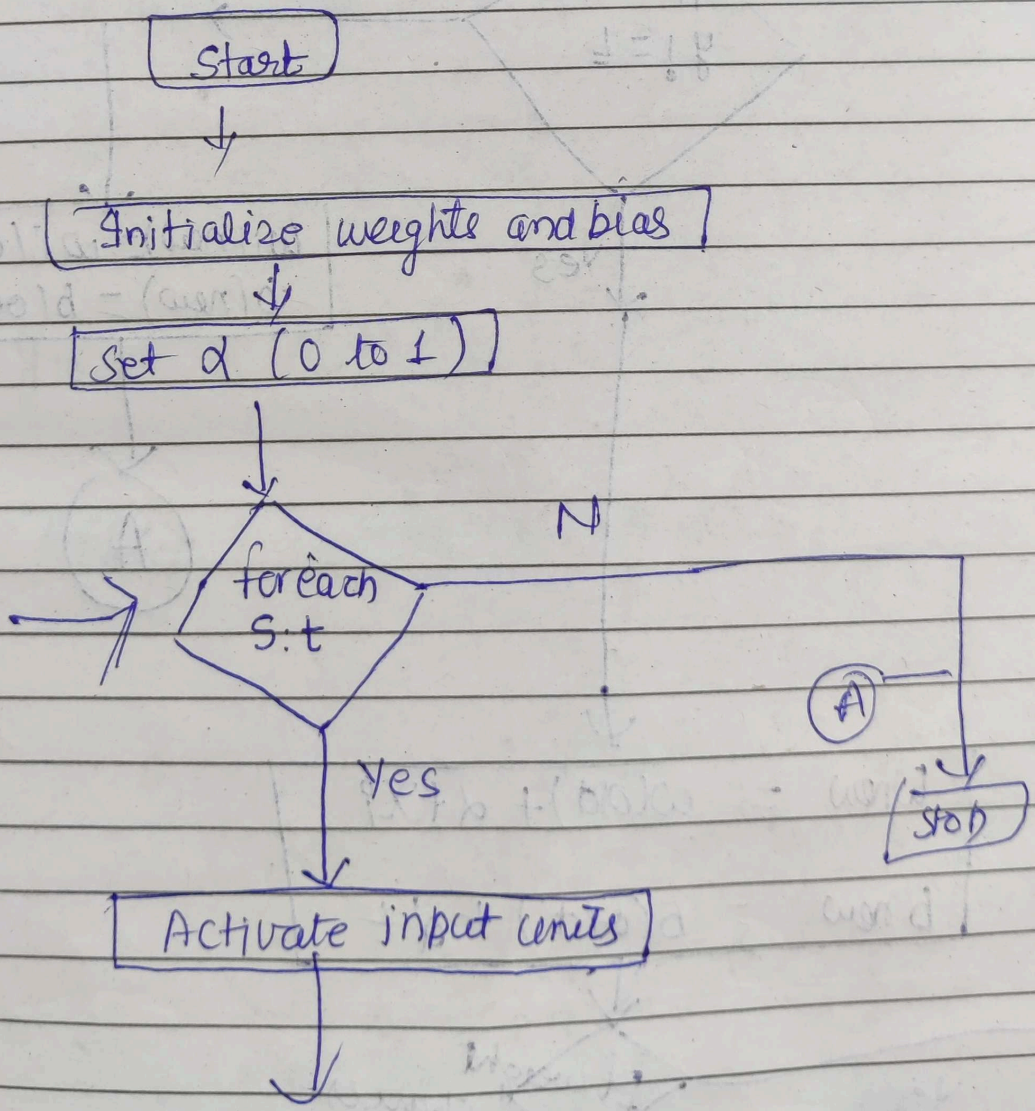
α → learning rate

x → input

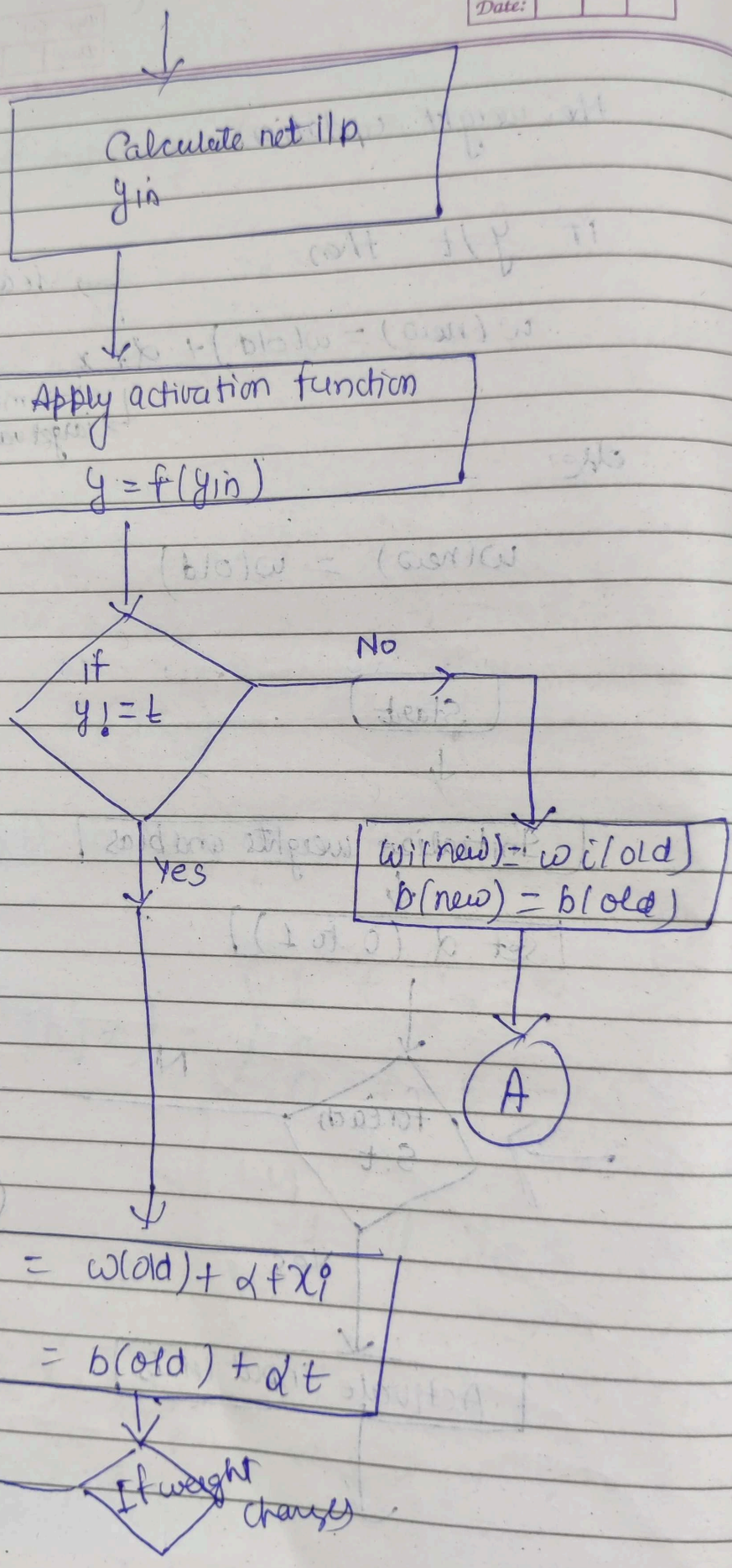
t → target value

else

$w(new) = w(old)$



Page No.
Date:



Step 0

Step 1

Step 2

Step 3

(Training Algorithm)

Step 0 : Initialize weights and bias

$q = 1$

Step 1 Perform step 2-5 until final stopping condition is false

Step 2 Perform step 3-4 for each training pair

Step 3 Calculate the o/p

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

Apply activation function

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

Page No.
Date:

Step 4 : weight and bias adjustment

if $y \neq t$ then

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

$$b(new) = b(old) + \alpha t (1)$$

else

$$w_i(new) = w_i(old)$$
$$b(new) = b(old)$$

Step 5

Page No.
Date:

Testing Algorithm

step 0:

The initial weights to be used here are taken from training algo

step 1: For each training i/p vector x to be classified

perform steps 2–3

step 2 set activations

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

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

Date:

Problem:

Implement AND function using Perceptron networks for bipolar inputs and targets

Epoch 1

Inputs x_1	Inputs x_2	Target t	Net Input y_{in} (Initially $w_1=0$, $w_2=0$, $b=0$)	Calculated o/p	Weight changes ($\alpha=1$) Δw_1	Δw_2	Δb
Epoch 1							
1	1	1	$y_{in}=b+x_1w_1+x_2w_2$ $=0+1\times0+1\times0=0$	0 w^0	$0+x_1$ 1×1 $=1$	$0+x_2$ 1×1 $=1$	$0+t$ 1×1 $=1$
1	-1	-1	$=1+1\times1+-1\times1\neq1$	-1 w^1	$1+x_1$ 1×1 $=2$	$1+x_2$ 1×-1 $=0$	$1+t$ 1×-1 $=0$
-1	1	-1	$0+-1\times0+1\times2$ $=2$	1 w^1	$2+x_1$ $-1+-1$ $=-2$	$0+x_2$ 1×1 $=1$	$0+t$ 1×-1 $=-1$
-1	-1	-1	$=-1+(-1)(-1)+$ -1×1 $=-1-1=-3$	-1 equal			

New weights

New weights w_1	w_2	b			
w_1	w_2	b			
$w_1 + \Delta w_1$	$w_2 + \Delta w_2$	$b + \Delta b$			
$0 + 1 = 1$	$0 + 1 = 1$	$0 + 1 = 1$			
$1 + 1 = 0$	$1 + 1 = 2$	$1 - 1 = 0$			
$0 + 1 = 1$	$2 - 1 = 1$	$0 - 1 = -1$			
1	1	-1			

$$3w_1 + 3w_2 - b = 0$$

$$3w_1 + 3w_2 = b$$

$$3(1) + 3(1) = -1$$

$$3 + 3 = -1$$

$$6 \neq -1$$

Epoch 2

$w_1 = 1,\ w_2 = 1,\ b = -1$

x_1	x_2	1	Target	Net Input y_{in} ($w_1=0, w_2=0, b=0$)	Calculated o/p y	weight changes Δw_1	Δw_2	Δb	w_1	w_2	b
						$\alpha t x_1$	$\alpha t x_2$	αt	old $+\Delta w$	old Δw_2	old $+\Delta b$
1	1	1	1	$-1+1\times1+1\times1 = 1$	equal 1	0	0	0	1	1	-1
1	-1	1	-1	$-1+(1\times1)+(-1\times1) = -1$	equal -1	0	0	0	1	1	-1
-1	1	1	-1	$-1+-1\times1+1\times1 = -1$	equal -1	0	0	0	1	1	-1
-1	-1	1	-1	$-1+-1\times1+-1\times1 = -3$	equal -1	0	0	0	1	1	-1

$y_{in} < 0$

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

Page No.
Date:

Final o/p = $w_1 = 1,\ w_2 = -1$

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

Final o/p = $w_1 = 1$, $w_2 = -1$

Final o/p = $w_1 = w_2 = 1$, $b = -1$

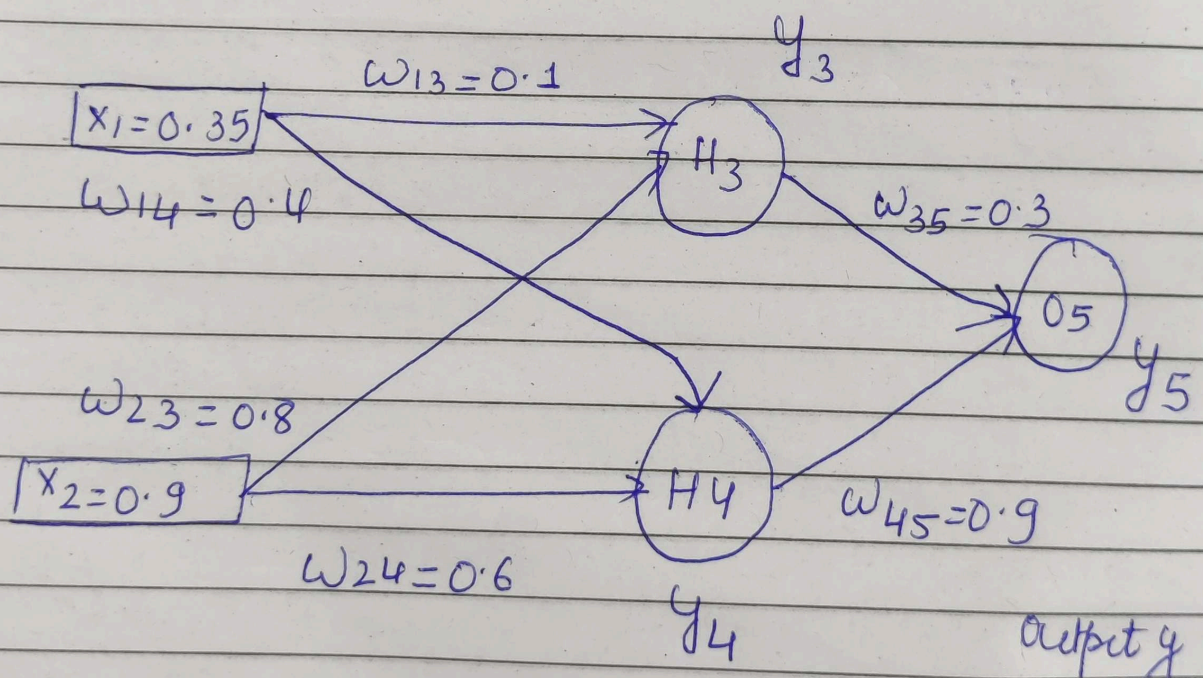
Page No.
Date:

Page No.
Date:

Back Propagation

Assume that the neurons have a Sigmoid activation function, perform a forward pass and a backward pass on the network.

Assume that the actual output of y is 0.5 and learning rate is 1. Perform another forward pass.



→ Forward pass: compute output for y_3, y_4 and y_5

$$a_j = \sum_j w_{ij} * x_i$$

$$y_j = F(a_j) = \frac{1}{1+e^{-a_j}}$$

$$a_1 = (w_{13} x x_1) + (w_{23} * x_2)$$

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

$$Y_3 = f(a_1) = \frac{1}{1+e^{-0.755}} = 0.68$$

$$a_2 = w_{14} * x_1 + w_{24} * x_2$$

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

$$y_4 = f(a_2) = \frac{1}{1+e^{-0.68}}$$

$$= 0.6637$$

$$a_3 = (w_{35} * y_3) + (w_{45} * y_4)$$

$$= (0.3 * 0.68) + (0.9 * 0.6637)$$

$$= 0.801$$

$$y_5 = f(a_3) = \frac{1}{1+e^{-0.801}}$$

$$= 0.69 \text{ (Network output)}$$

Error = $y_{target} - y_5$

$$= 0.5 - 0.69$$

$$= -0.19$$

Try to update weights

- Each weight changed by

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

$$\delta_j = o_j (1 - o_j)(t_j - o_j) \quad \text{if } j \text{ is an output unit}$$

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

where η is a constant called learning rate

t_j is the correct teacher output for unit j

δ_j is the error measure for unit j

Page No.
Date:

Backward pass

Compute δ_3 δ_4 and δ_5

for output unit

$$\delta_5 = y(1-y)(y_{target} - y)$$

$$= 0.69(1-0.69) * (0.5 - 0.69)$$

$$= -0.0406$$

For hidden unit

$$\delta_3 = y_3(1-y_3) w_{35} * \delta_5$$

$$= 0.68(1-0.68) * (0.3 * -.0406)$$

$$= -0.00265$$

$$\delta_4 = y_4(1-y_4) w_{45} * \delta_5$$

$$= -.6637 * (1-0.6637) * (0.9 * (-0.0406))$$

$$= -0.0082$$

Page No.
Date:

Compute New weights

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

$$\Delta w_{45} = \eta \delta_5 y_4 = 1 * -0.0406 * 0.6637 = -0.0269$$

$$w_{45}(new) = \Delta w_{45} + w_{45}(old)$$
$$= -0.0269 + 0.9 = 0.8731$$

$$\Delta w_{14} = \eta \delta_4 x_1 = 1 * -0.0082 * 0.35 = -0.00287$$

$$w_{14}(new) = \Delta w_{14} + w_{14}(old) = -0.00287 + 0.4$$
$$= .3971$$

Evolutionary Algorithms

The algorithms, which follow some biological and physical behaviours:→

Biologic behaviours:

1. Genetic and Evolution → Genetic Algorithms (GA)
2. Behaviour of ant Colony → Ant Colony optimization (ACO)
3. Human nervous system → ~~Artificial~~ Artificial Neural Network (ANN)

In addition to that there are some algorithms inspired by some physical behaviours:-

Physical behaviours:-

1) Annealing process → Simulated Annealing (SA)
2) Swarming of particle → Particle Swarming optimization (PSO)
3) Learning → Fuzzy logic (FL)

Page No.
Date:

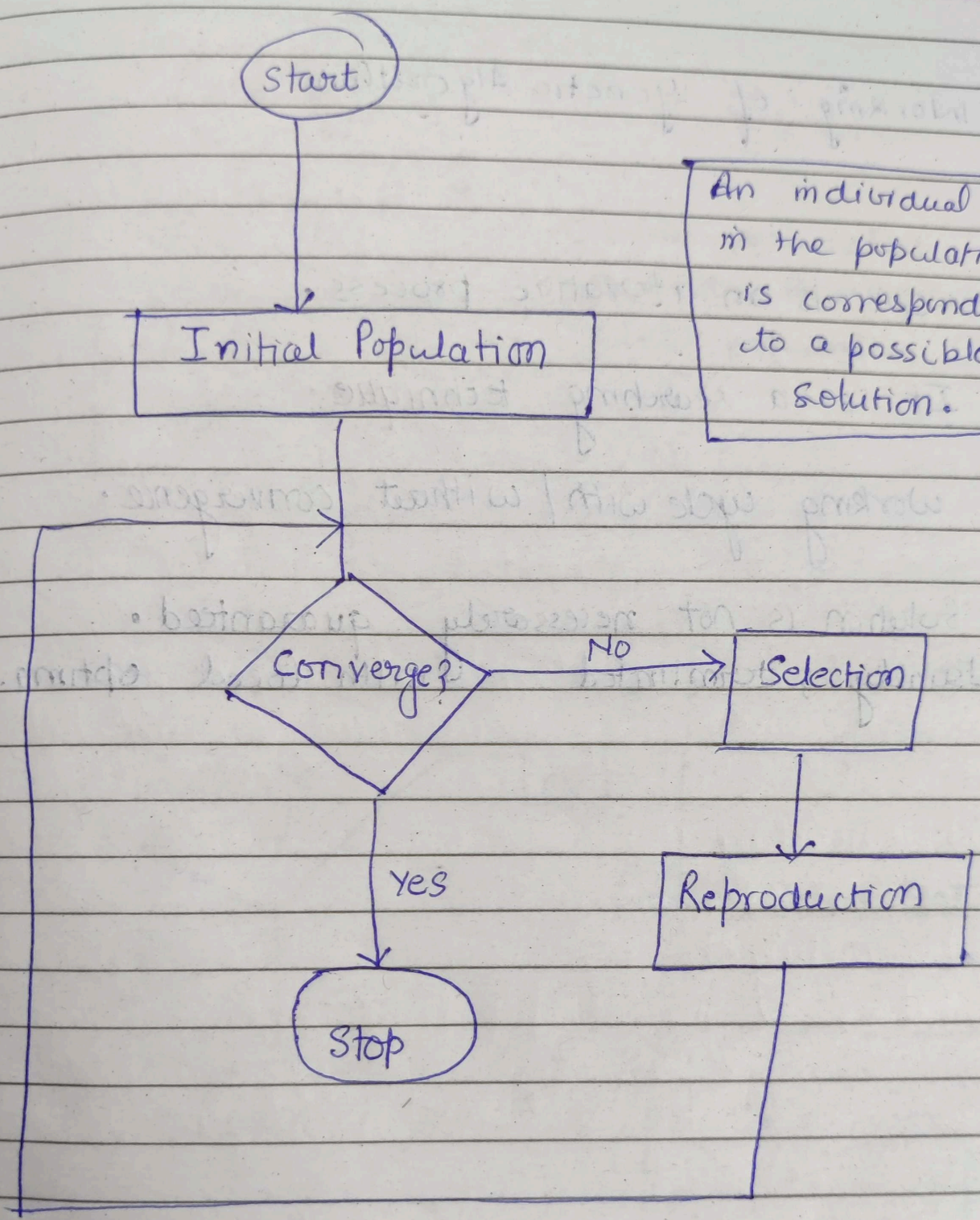
Genetic Algorithm

It is a subset of evolutionary algo-

1) Ant Colony optimization
2) Swarm Particle Optimization

Models biological processes :-

1) Genetics
2) Evolution


Start
Initial Population
An individual in the population is corresponding to a possible solution.
Converge?
No
Selection
Reproduction
Yes
Stop

Page No.
Date:

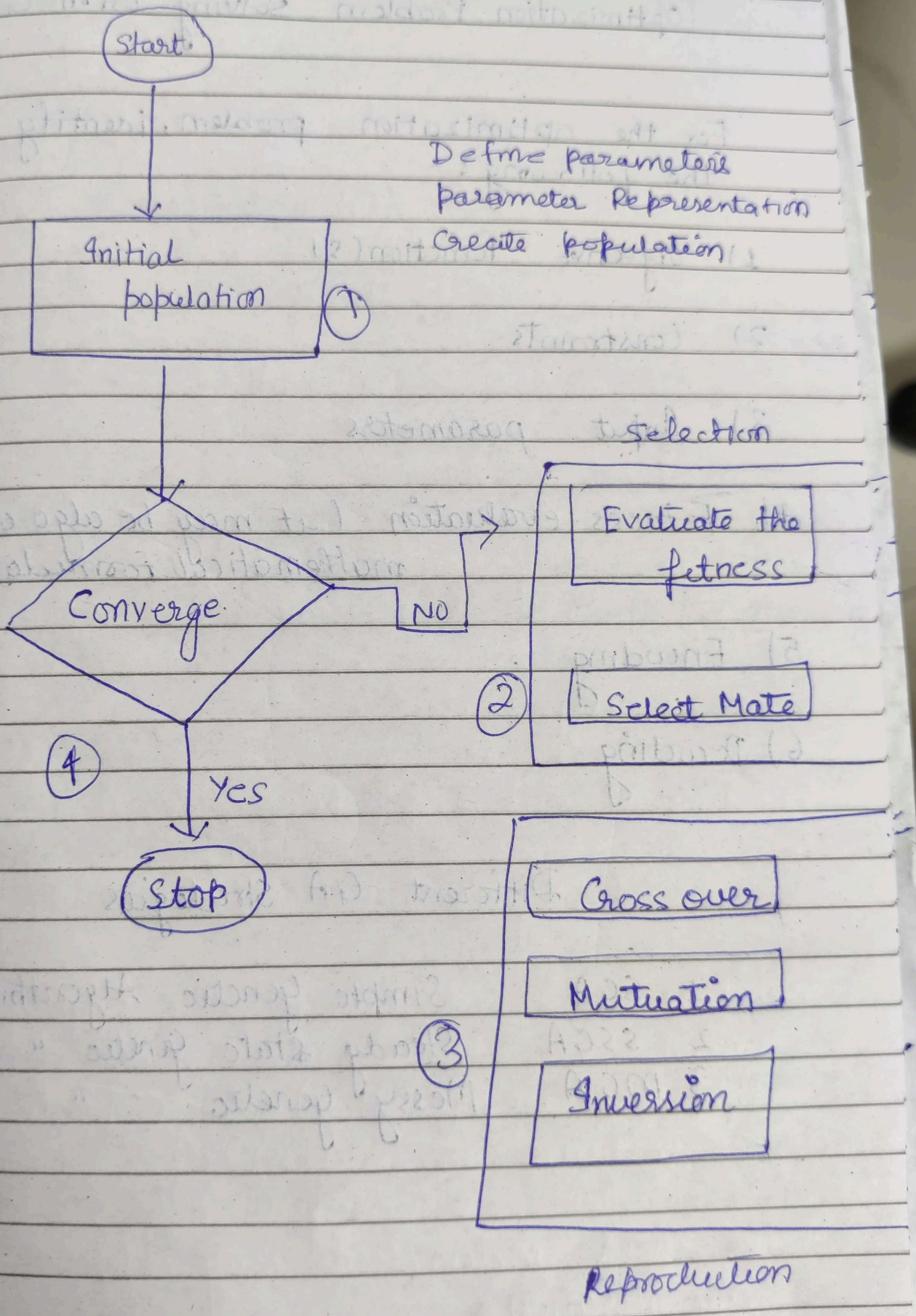
Working of Genetic Algorithm

1) GA is an iterative process.

2) It is a searching technique.

3) Working cycle with / without convergence.

4) Solution is not necessarily guaranteed.
Usually, terminated with local optima.


Start
Initial population
1
Define parameters
parameter Representation
Create population
Converge
NO
Yes
4
Selection
Evaluate the fetness
2
Select Mate
Cross over
Mutation
3
Inversion
Reproduction
Stop

(Optimization Problem solving with GA)

For the optimization problem, identify the following:-

1) Objective function(s)

2) Constraints

3) Input parameters

4) Fitness evaluation (it may be algo or mathematical formula)

5) Encoding

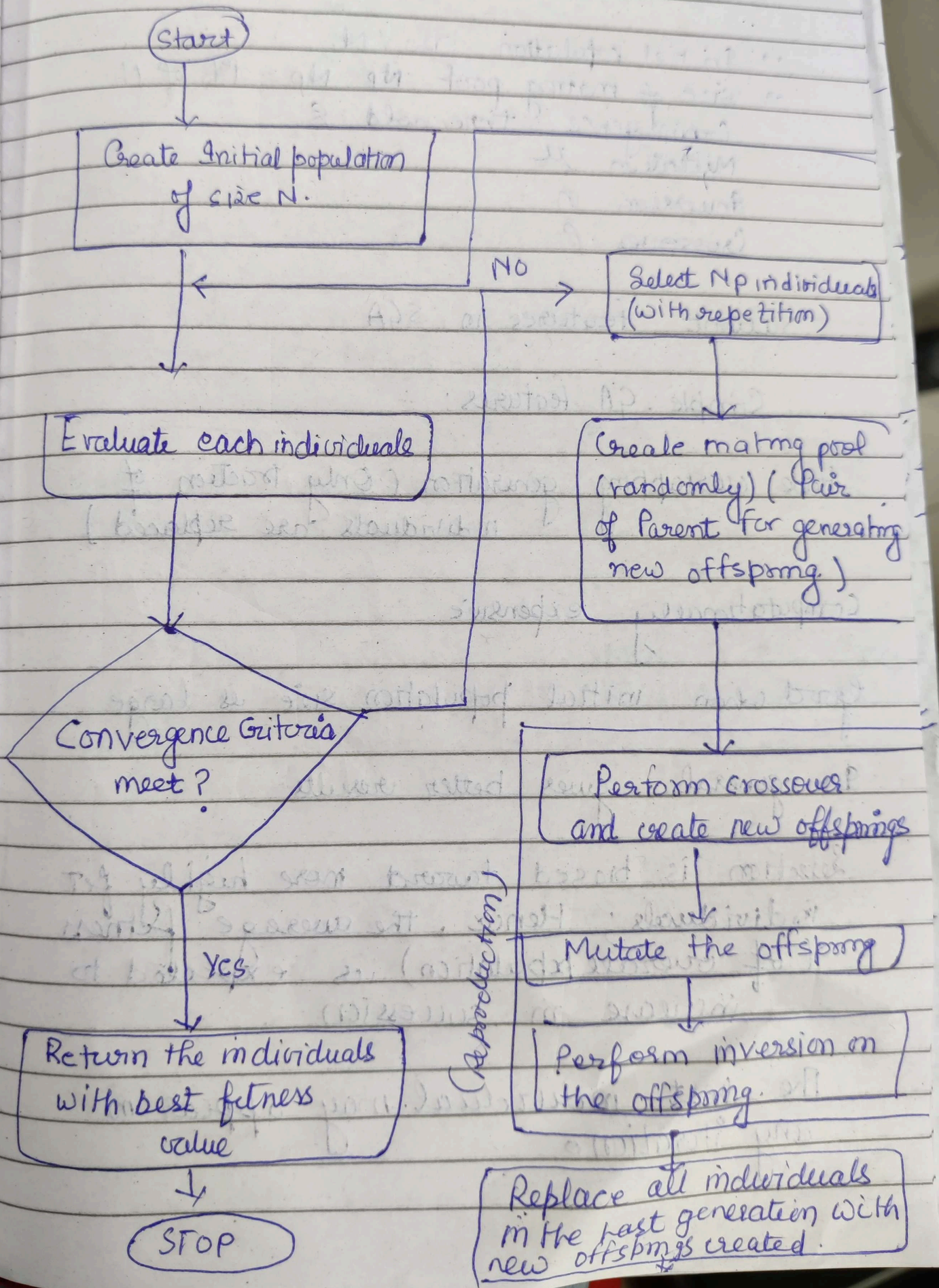
6) Decoding

Different GA Strategies

1. SGA Simple Genetic Algorithm.
2. SSGA Steady State Genetic " .
3. MGA Messy Genetic " .

Page No.
Date:

Simple GA



Genetic Algo:-

Probabistic search; random search;

Page No.
Date:

SGA parameters

→ Initial population size : N
→ size of mating pool N_p : N_p = P% of N
Convergence threshold δ
Mutation μ
Inversion η
Crossover ρ

Salient features in SGA

Simple GA features:

Have overlapping generation (Only fraction of individuals are replaced)

Computationally expensive

Good when initial population size is large

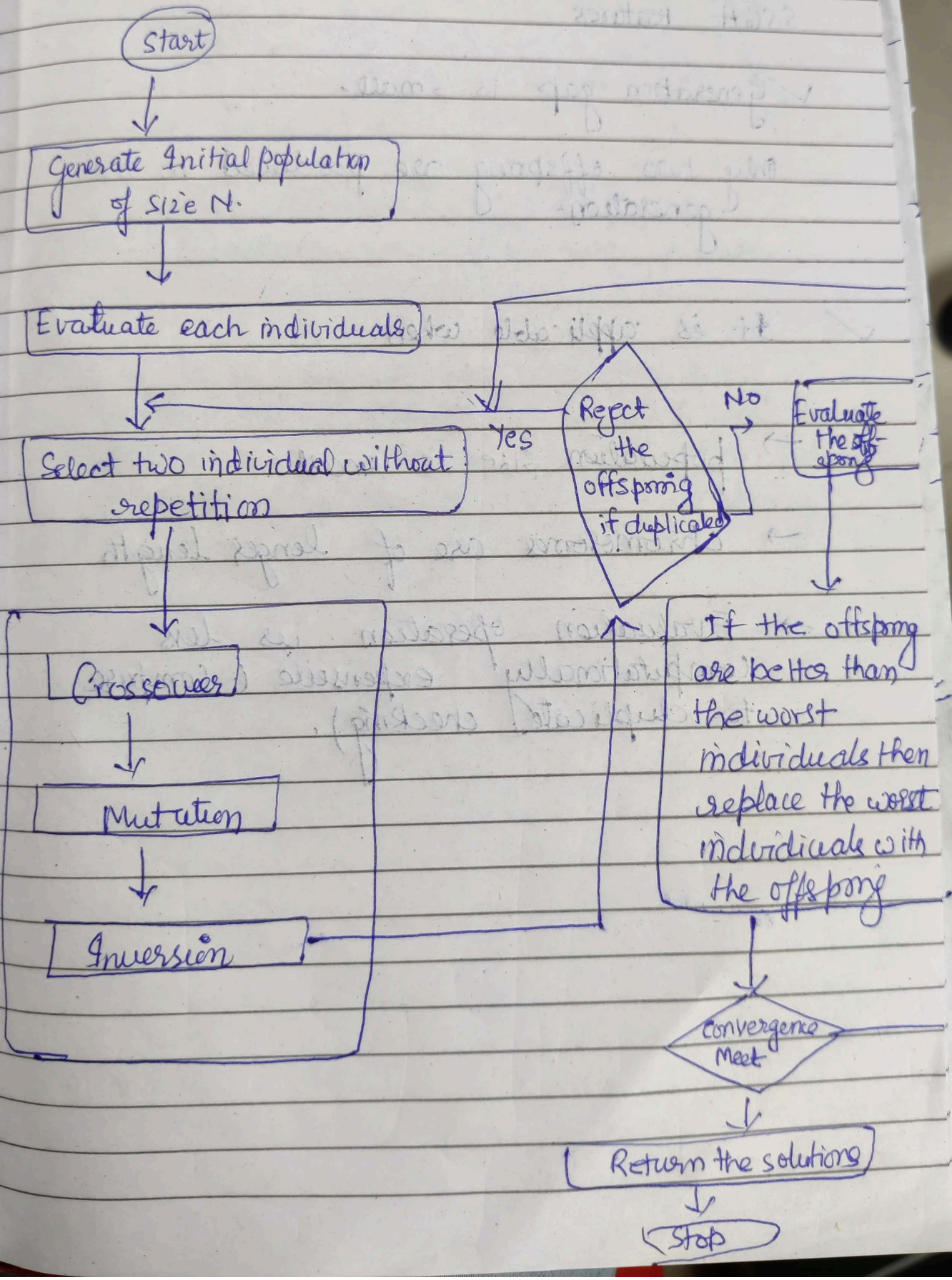
In general, gives better results

Selection is biased toward more highly fit individuals. Hence, the average fitness (of overall population) is expected to increase in succession.

The best individual may appear in any iteration.

Page No.
Date:

Steady State Genetic Algorithm (SSGA)



Date:

Lecture - 16

Following are the GA operators in Genetic Algorithms:-

Encoding
Convergence test
Mating pool
Fitness Evaluation
Crossover
Mutation
Inversion

Page No.
Date:

Encoding Schemes

↳ Binary encoding — Binary GA
↳ Real value encoding
↳ Order encoding — Permuted GA
↳ Tree encoding

(1) Individual — is a single solution
(2) Population — set of individuals