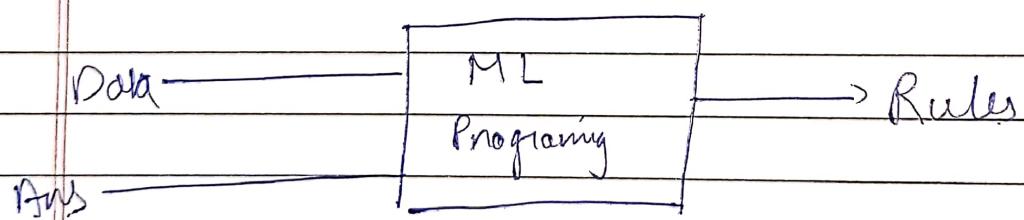
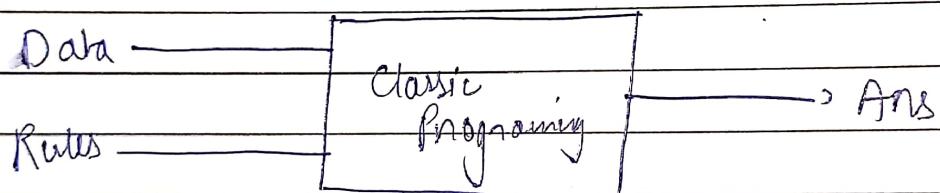
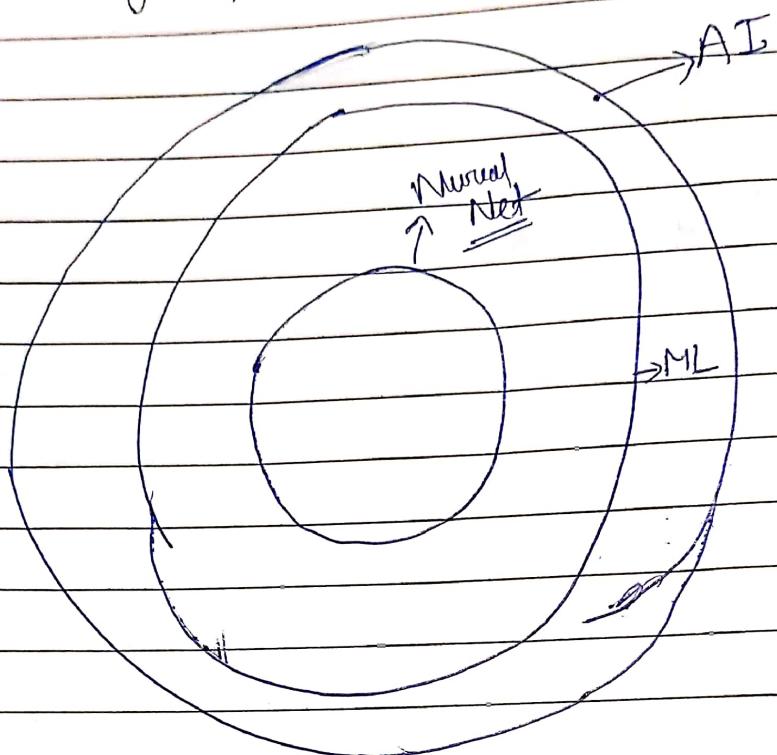


# Artificial Intelligence (Personal Study)

Date  
Page

Agent: Anything that can be viewed as perceiving its environment through the sensors and acting upon that environment through actuators



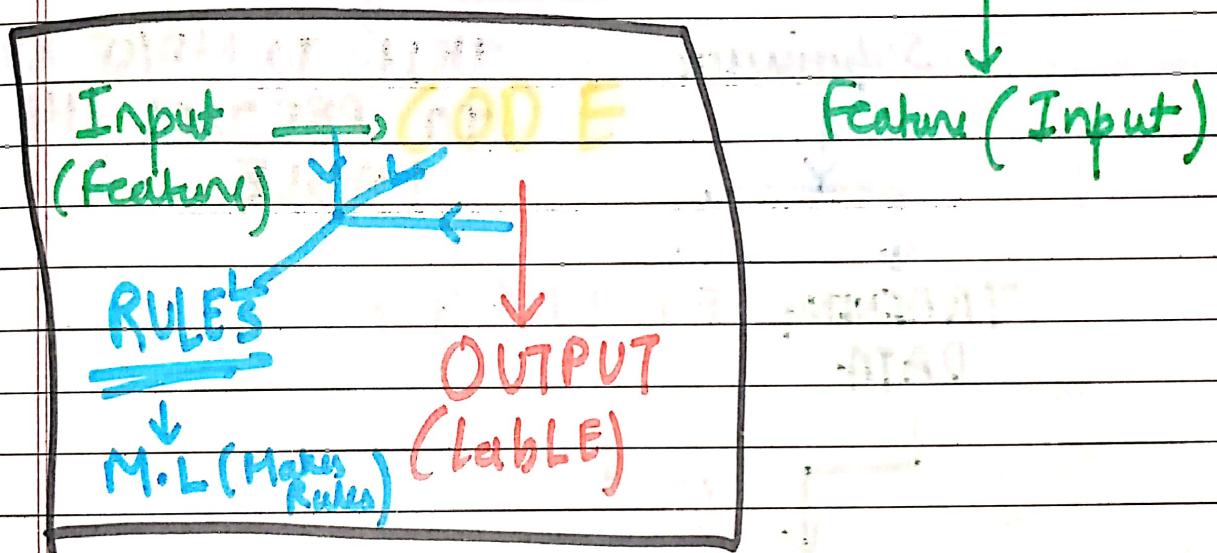
Neural Net → layered

Machine learning → figures out the rules for a given input and output

## Data

Ex Table. (for detection of Mid Term grade)

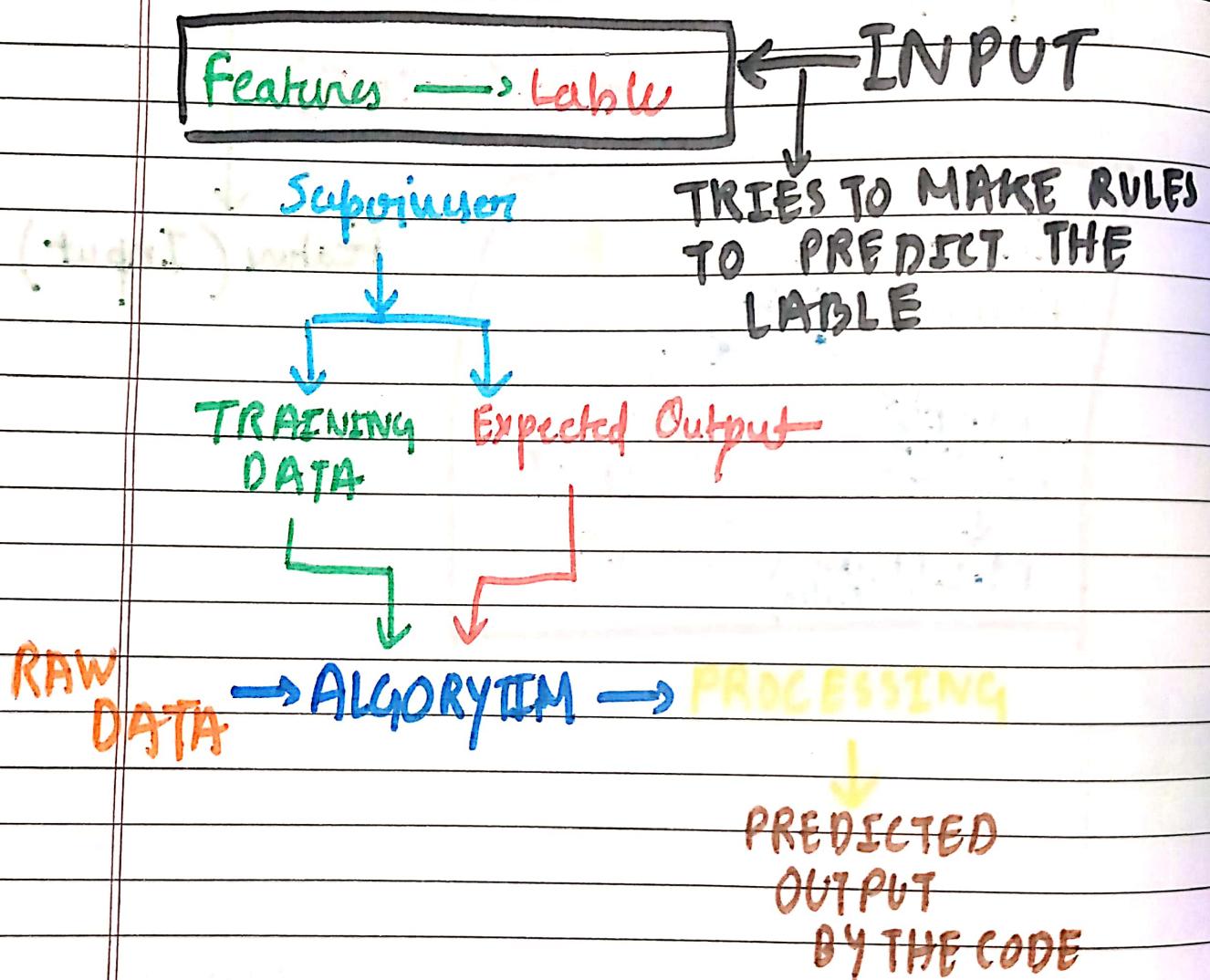
Feature	Mid form 1	Mid form 2	Final	Feature
1.	70	80	77	
2.	60	90	84	
3.	40	50	38	



## Different types of M.L.

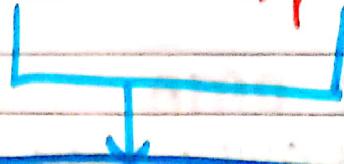
- Unsupervised learning
- Supervised learning
- Reinforcement learning

## Supervised LEARNING



IF,

PREDICTED OUTPUT != Expected OUTPUT

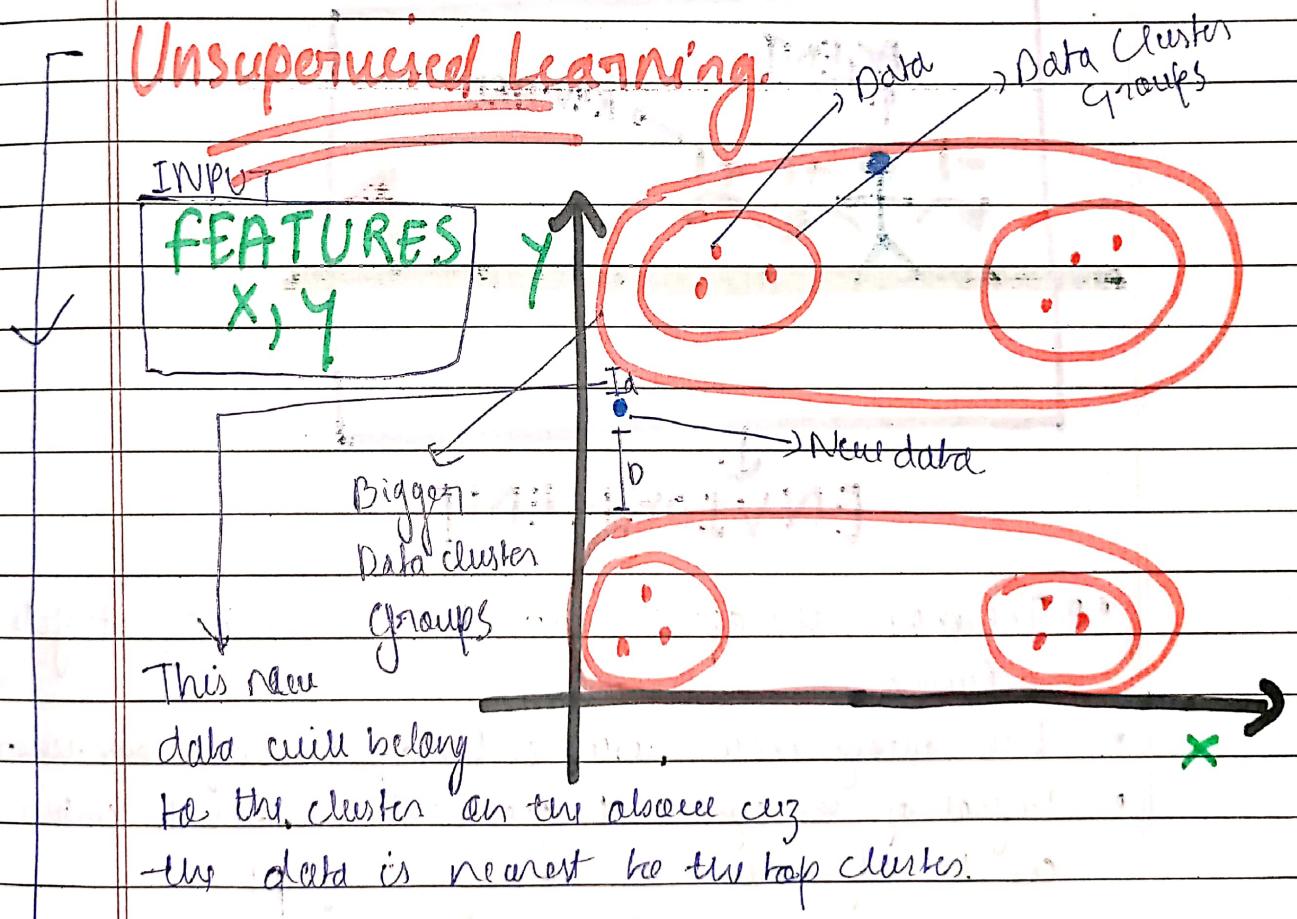


Tweaks Made by the Supervisor to get the correct output

THIS IS REPEATED UNTIL,

PREDICTED OUTPUT == Expected OUTPUT

## Unsupervised Learning.

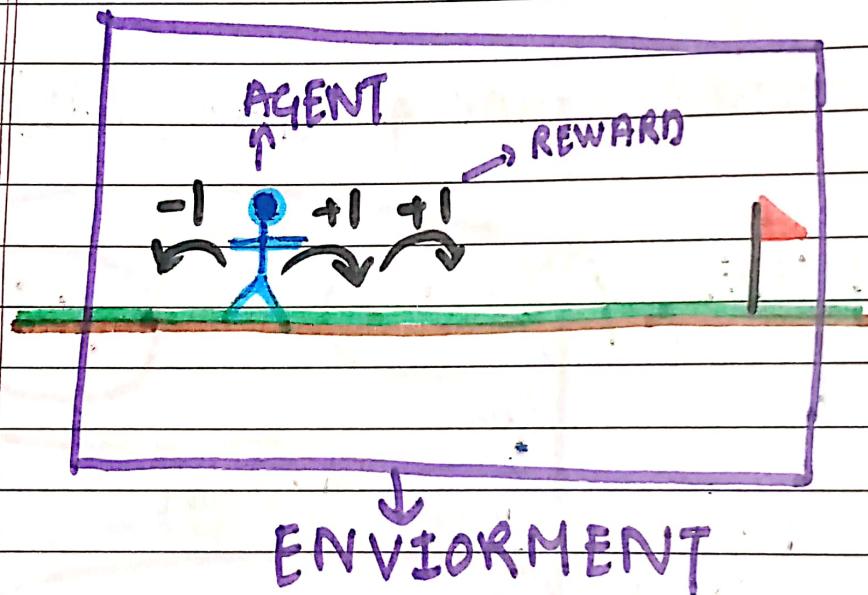


Essentially groups data and data clusters and figures out where new data will go

## REINFORCEMENT LEARNING

- NO DATA
- you ONLY have a AGENT
- An Environment

For example (goal is to reach the flag)



- The more the agent moves towards the flag it gets a reward
- Each wrong move results in the agent losing some reward
- The goal of the agent is to get maximum reward which can be only done by reaching the goal.

\* GOAL OF THE AGENT IS TO  
GET MAXIMUM REWARD

Import Tensorflow as tf

# INTRODUCTION TO TENSOR Flow

>>> pip install TENSORFlow

A collection of Data  
for our ai  
Making Math easy.

Tensor :- Vector generalised in higher dimensions

Data types

Represent dimensions  
of the data

float32, int32, string etc.

How to create tensors

Example

String = tf.Variable("this is a string", tf.String)

Number = tf.Variable(1234, tf.int16)

Floating = tf.Variable(3.567, tf.float64)

Rank and Degree of tensor.

## RANK / DEGREE

## DIMENTION

1D ← rank0-tensor = tf.Variable("str", tf.string)  
or scalar

2D ← rank1-tensor = tf.Variable(["Hey", "Hello"], tf.string)

3D ← rank2-tensor = tf.Variable([["test"]], tf.string)

Dimensions can be known by counting the number  
of nested list.

To find the Rank we can use :-

tf.rank(rank2-tensor)

OUTPUT

<tf.Tensor: Shape=(), dtype=int32, numpy=2>

Rank ←

**Example** = `tf.Variable([["A", "B"], ["C", "D"]])`

## Shape of a tensor

- Simply no. elements in each dimension

To get shape we use:

`Example. Shape`

### OUTPUT

Tensor shape `(2, 2)`

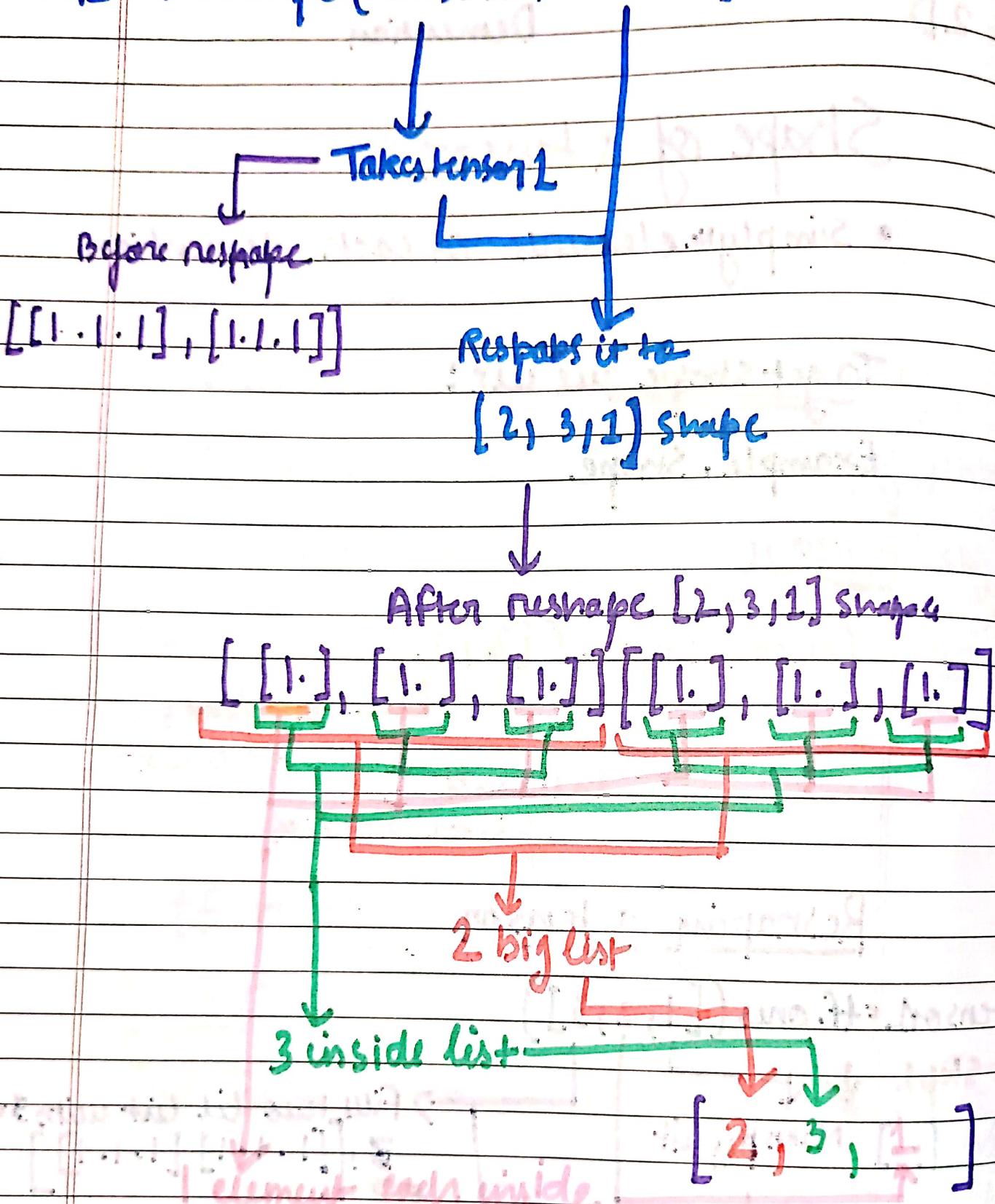
↓  
2 elements  
in first Dimension  
↓  
2 elements  
in 2<sup>nd</sup> dimension  
→ 2 elements  
in 2<sup>nd</sup> dimension

## Reshaping a tensor

`tensor1 = tf.ones([1, 2, 3])`

Step 1 [ ] Makes big list → Fill two list with 3 ones  
Step 2 ↓ Makes two list inside big list  
Step 3 [ [ ] , [ ] ]

tensor2 = tf.reshape(tensor1, [2, 3, 1])



No first value will be taken as 1

↑

Date \_\_\_\_\_  
Page \_\_\_\_\_

tensor3 = tf.reshape(tensor2, [3, -1])

↓

- It tells tensor to calculate the data  
Dimension

↓

$$[3, -1] = [3, 3-1] = [3, 2]$$

Tensor2 before modification

$[[[1.], [1.], [1.]] [[1.], [1.], [1.]]]$

New Tensor3

$\begin{bmatrix} [1. 1.] & [1. 1.] & [1. 1.] \end{bmatrix}$

## Types of Tensor:-

- Variable }  $\rightarrow$  Mutable
  - Constant
  - Placeholder }  $\rightarrow$  Immutable
  - SparseTensor

## Evaluating a tensor

get value of a tensor, (uses a graph)

use next scalars to eval tensor

With tf.Session() as sess:

`tensor.eval()`

↓

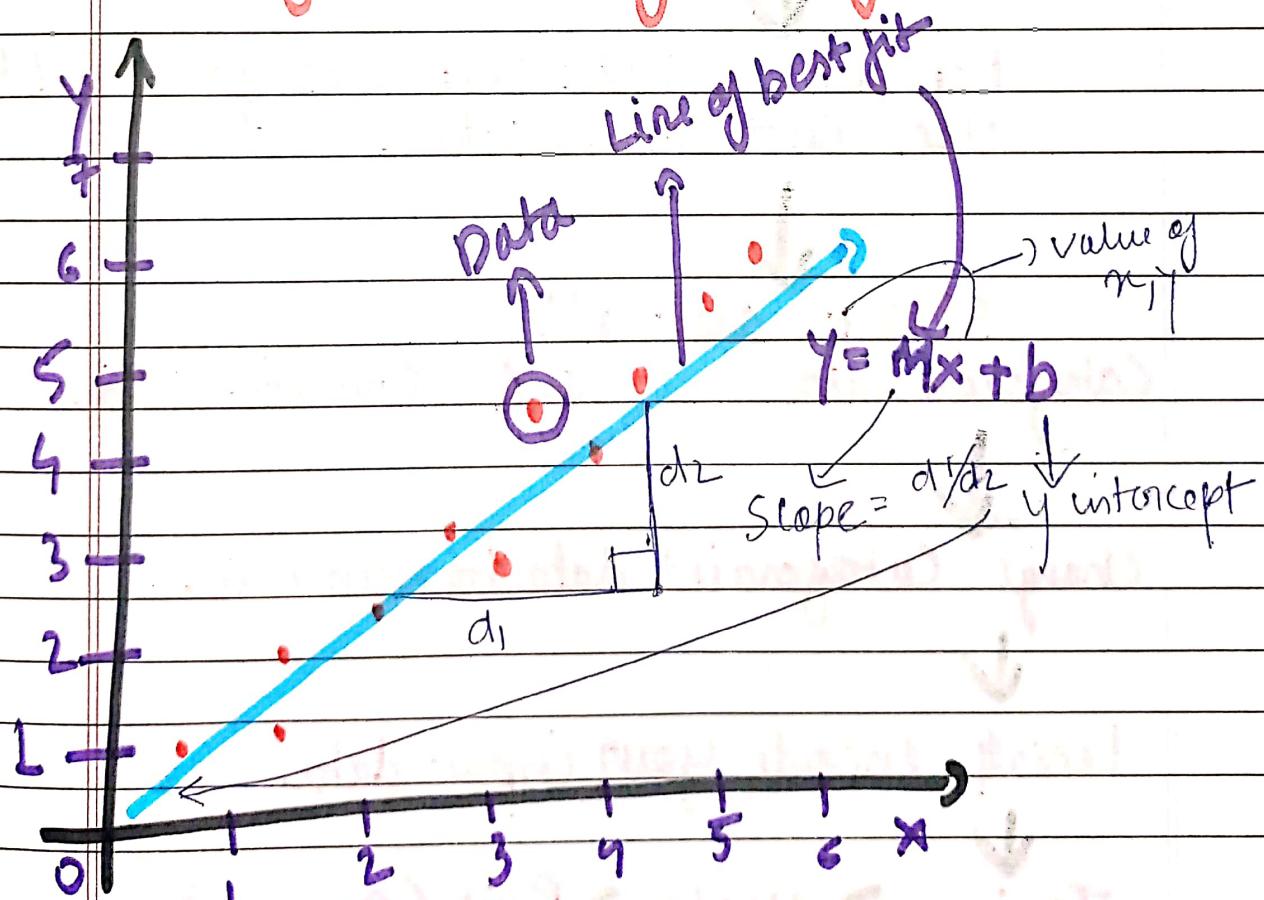
Name of tensor.

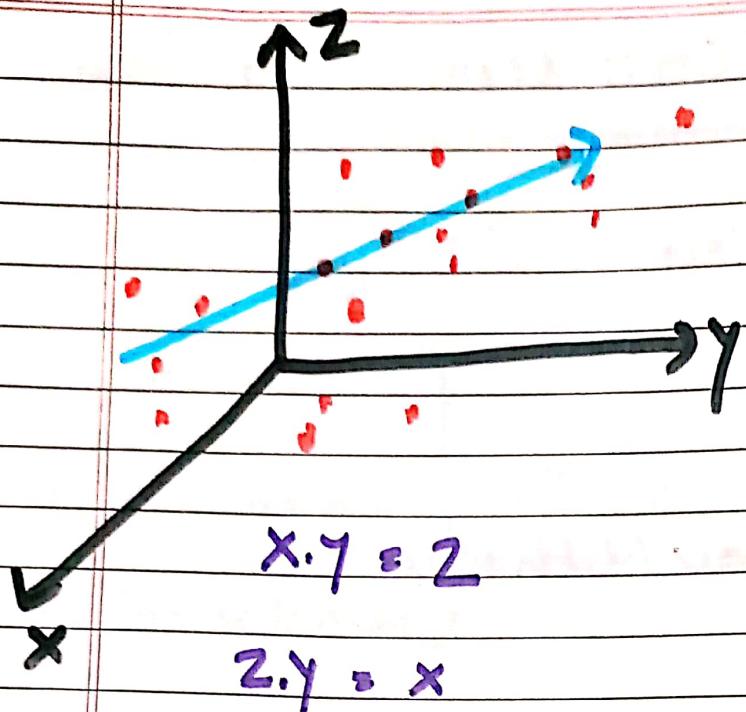
# Tensorflow CORE learning algorithms

- ) Linear Regression
- ) Classification
- ) Clustering
- ) Hidden Markov Methods

## Linear Regression

Takes linearly correlated Data to find the line of best fit and predict using the line of best fit





## In CODE

Take a .csv (datasheet)

↓  
Pop out all the elements you need to use into a variable

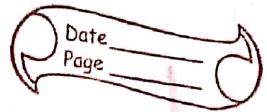
↓  
Short out all Categorical Data and Numeric data?

↓  
Change Categorical data to numeric.

↓  
Lastly, encode your input data.

↓  
Train → eval → Print (Accuracy)

# CLASSIFICATION



CLASSIFIES DATA INTO DIFFERENT DATA SET (CLASSES)

< TAKE IN CODE >

→ CLASSIFIER TYPE

- DNN CLASSIFIER (DEEP NEURAL NET)
- LINEAR CLASSIFIER

Take the CSV file

↓  
specify the class types

↓  
pop out the columns you need.

↓  
Make an input function

↓  
Make feature columns

↓  
Make the model

Classifier = tf.estimator.DNNClassifier(  
feature\_columns=feature\_my\_feature\_columns,  
hidden\_units=[30, 10],  
n\_classes=3)

Number of  
classes

Number of Nodes in 2<sup>nd</sup> layer  
Number of Nodes in 1<sup>st</sup> layer  
of Neural Net



## Train Model

`classifier.train (`

`input_fn = lambda: input_fn (train, train_y),`  
`Training = True, steps = 5000)`

Number of times  
to be runned

`lambda` defines a single line `map` function  
in python



`equal`

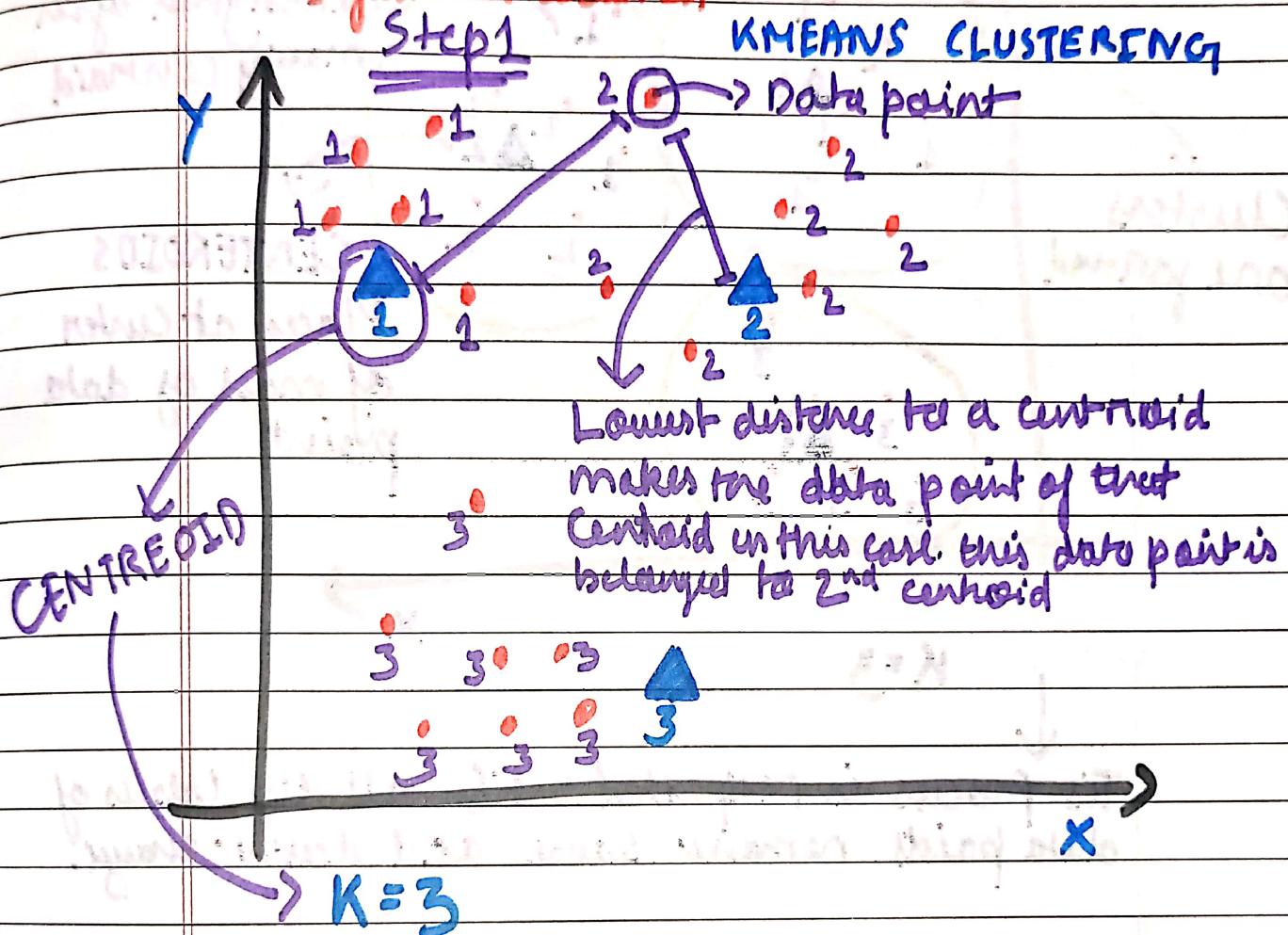


`Predict / Accuracy`

# CLUSTERING

Date \_\_\_\_\_  
Page \_\_\_\_\_

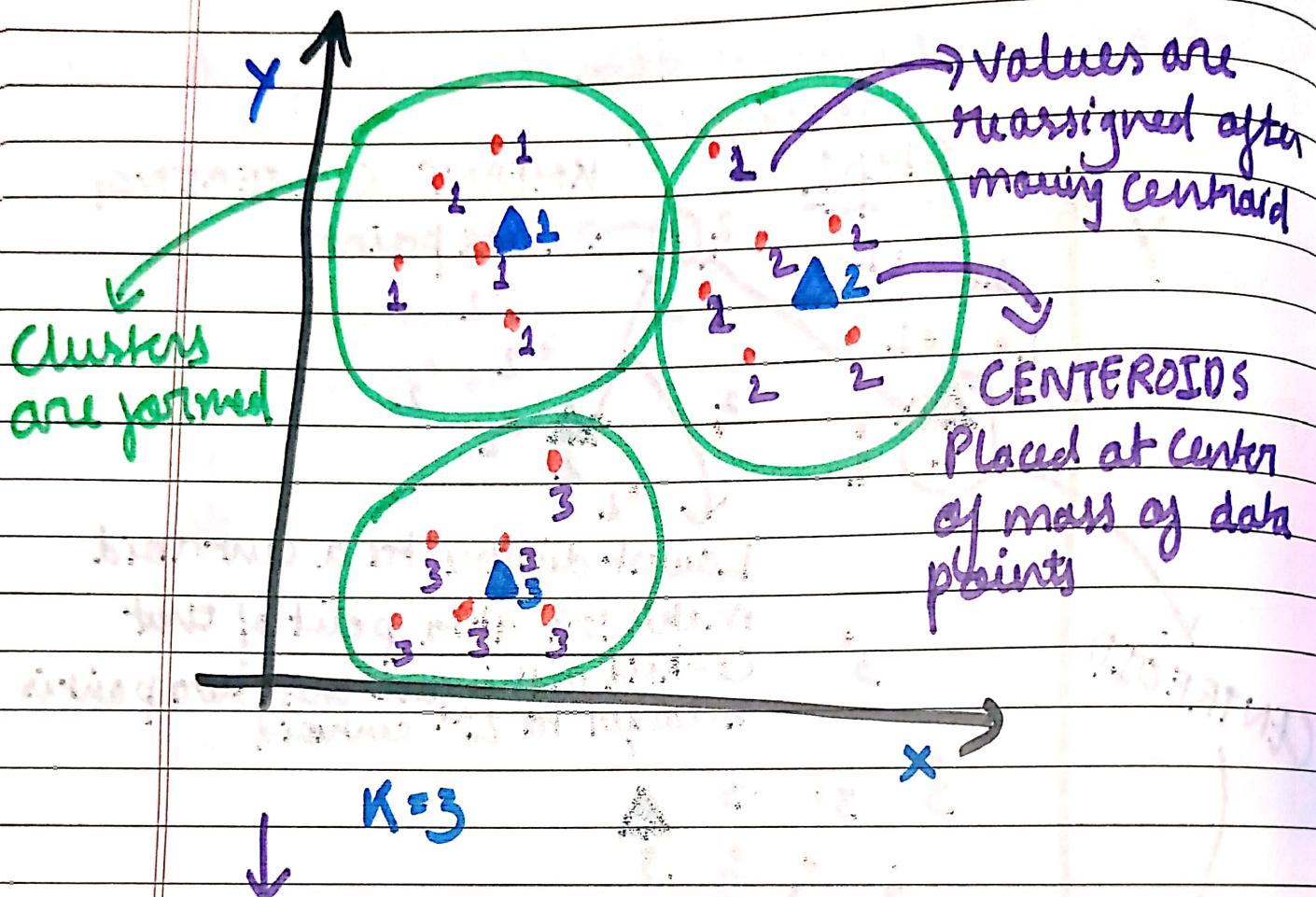
- • Unsupervised learning algorytm.
- • Used when you only have input info and no labels (output) info.
- • Finds cluster of data like data points and tells you its location



K = Number of clusters

## Step 2

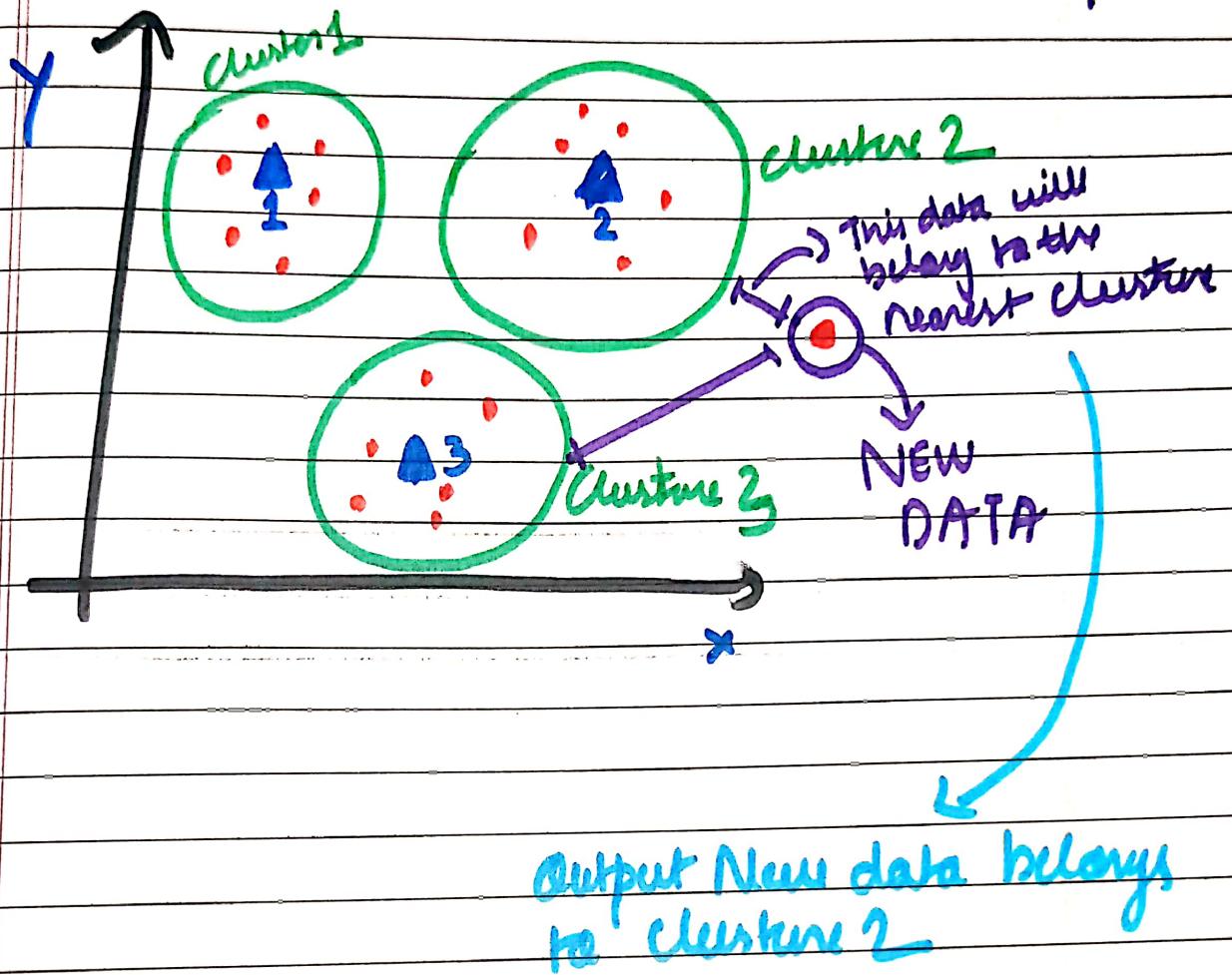
Finds center of mass of Data points  
and places centroids in the middle of them



This process is repeated until all the labels of data points remain same and doesn't change

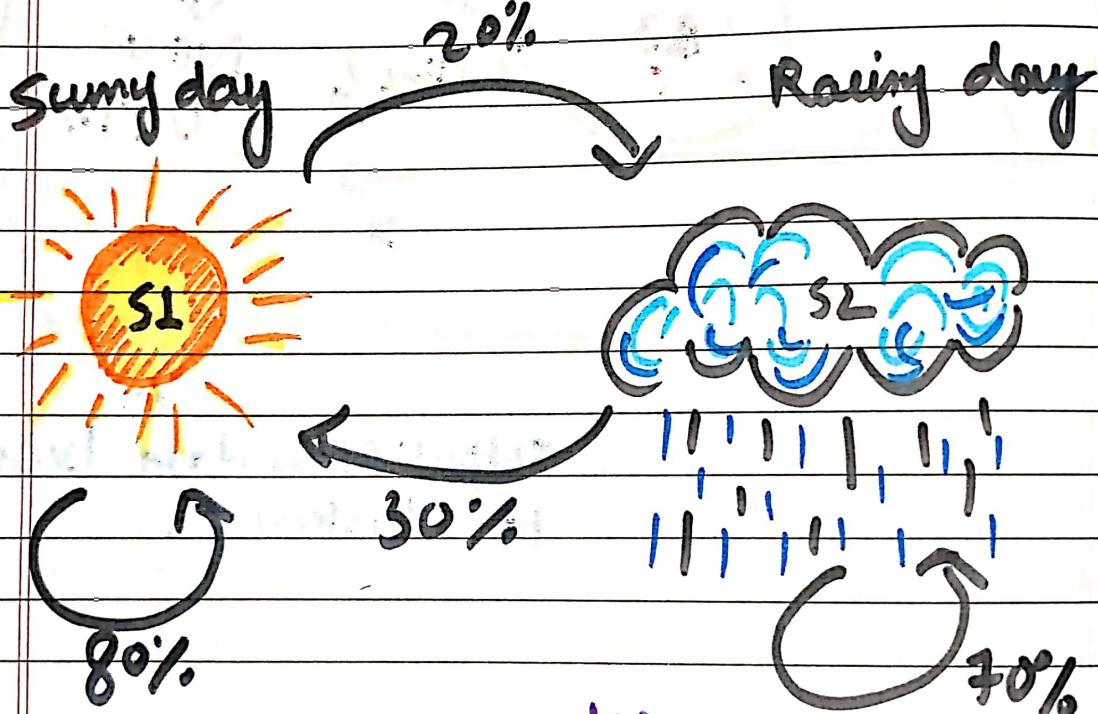
### Step 3

Once clusters are formed the new data are assigned to the nearest cluster and labelled the label will be the output



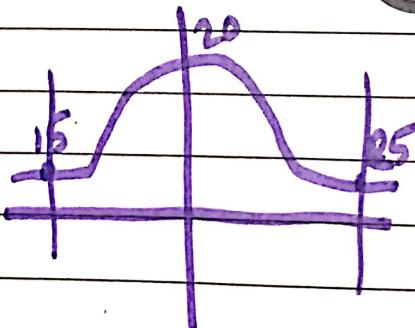
# HIDDEN MARKOV MODEL

- ↳ • finds probability of something happening based on observations
- ↓
  - States
  - ↓
    - Observation Distribution
    - ↓
      - Transition Distribution



Observation:

Mean: 20  
Min: 15  
Max: 25



Observation:  
Mean: 5  
Min: -5  
Max: 15

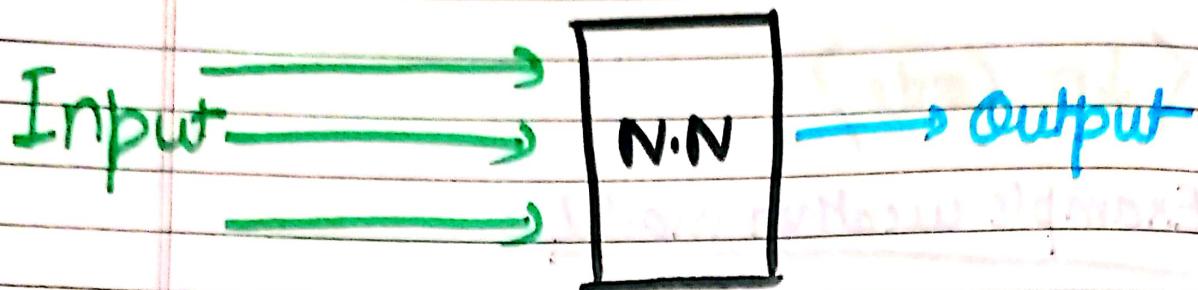
Purpose: Predict future event based on passed events

### < In Code >

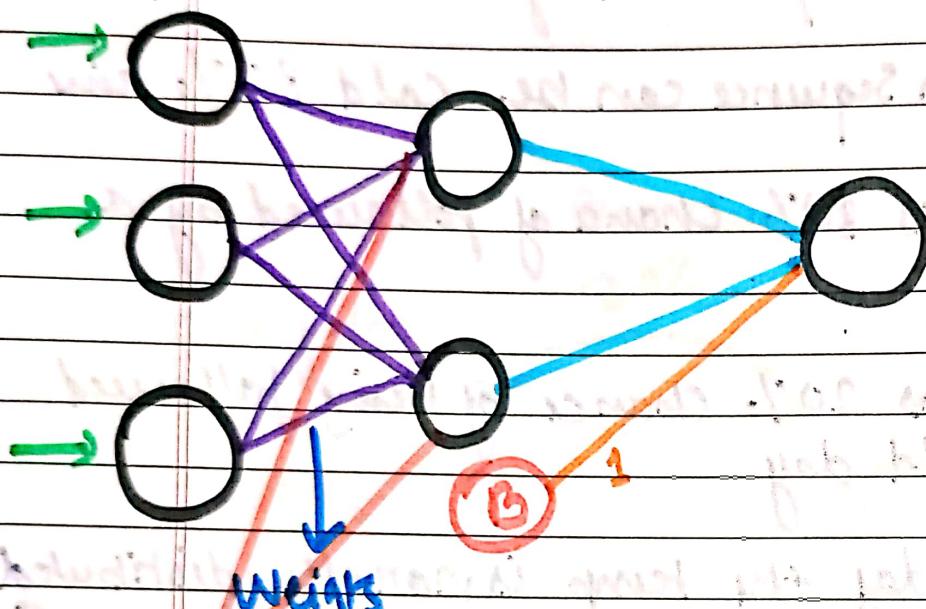
Example weather model.

1. Cold days = 0 / hot days = 1
2. First day in sequence can be cold 80% time
3. Cold day has 30% chance of followed by a hot day
4. Hot day has 20% chance of being followed by a cold day
5. On each day the temp is normally distributed with mean 0 and standard deviation 0 and 5 on a cold day and mean and standard deviation 15 and 10 on a hot day

# NEURAL NETWORKS



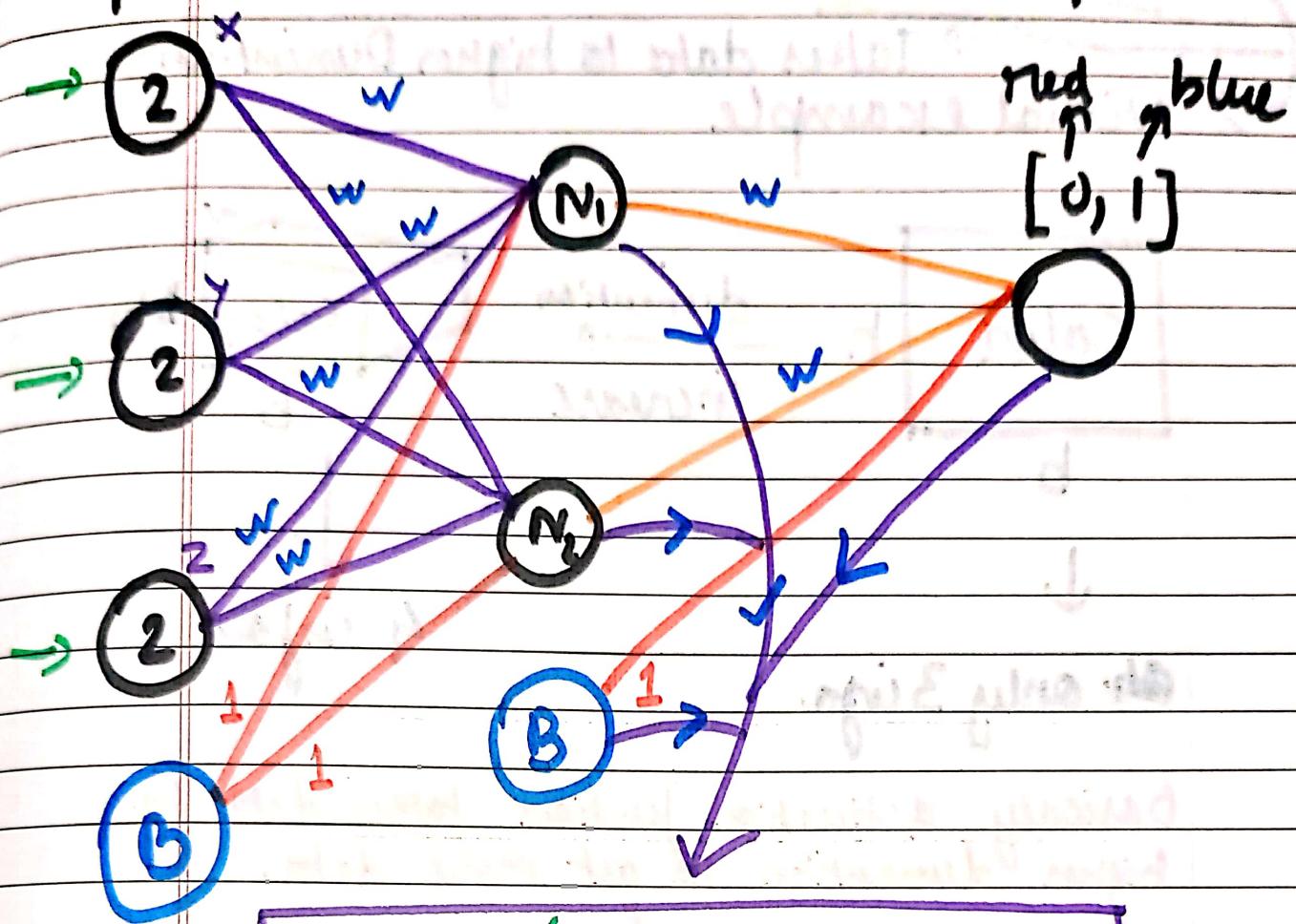
Input      Hidden      Output



Biases: Constant value

Example:  $(x, y, z) \rightarrow [\text{red}, \text{blue}]$

Input      Hidden      Output



activation function

$$N_x^F = \left( \sum_{i=0}^n w_i x_i + b \right)$$

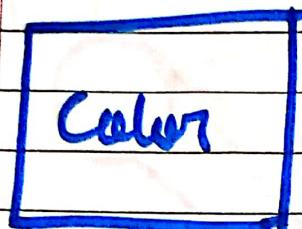
$$= w_x(2) + w_y(2) + w_z(2)$$

= v

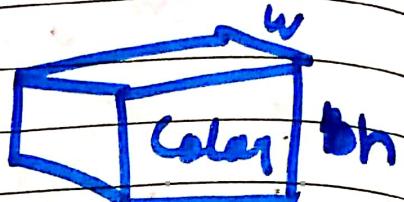
→ Next page

## Activation function

→ Takes data to higher Dimension  
Visual example



h dimension  
increase



4 info

But only 3 info

Basically activation function takes data to higher dimension to get more data

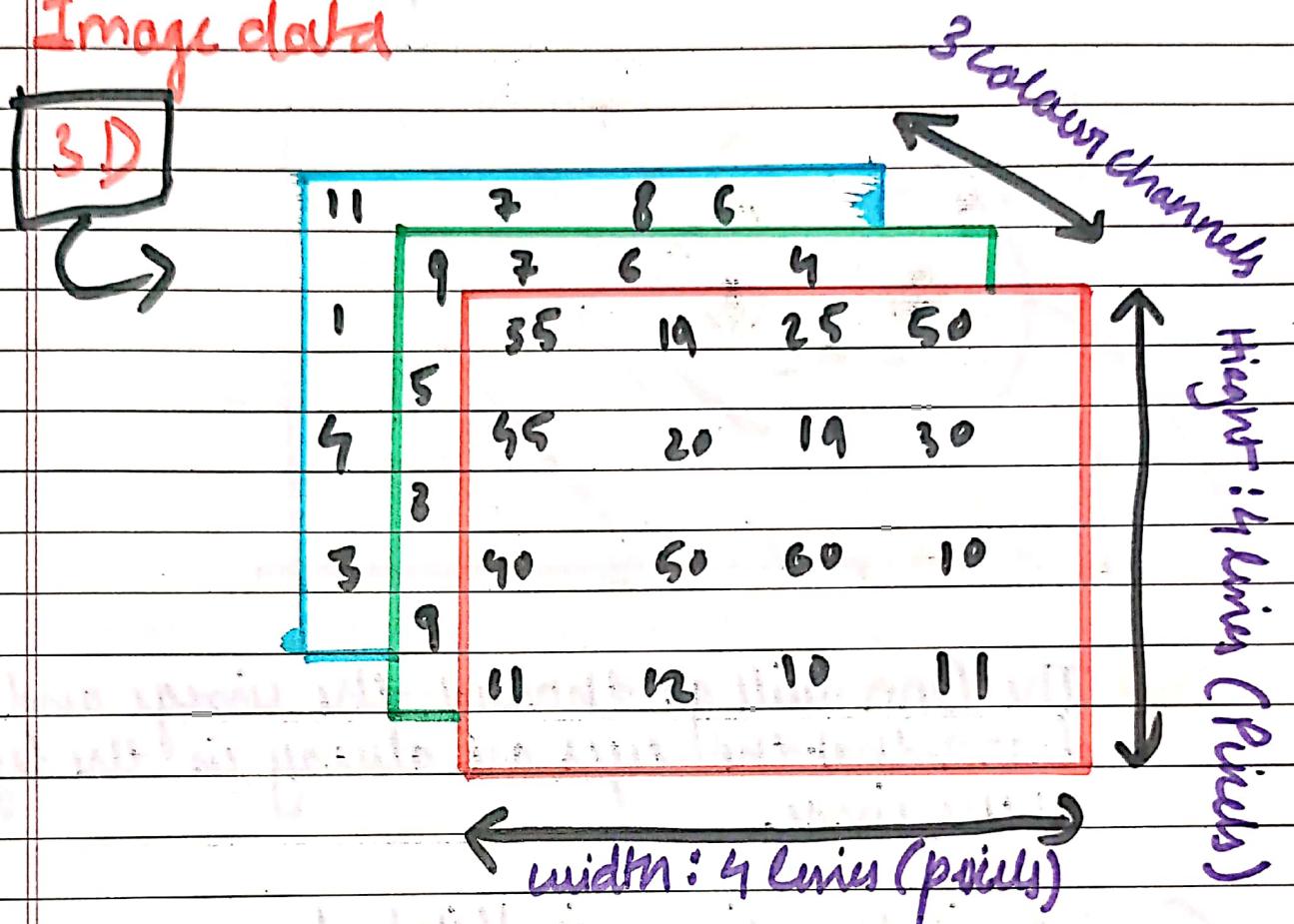
## Loss function

Calculates how far output is from expected output and suggest changes

$$(S)_{\text{actual}} + (S)_{\text{expected}} - (S)_{\text{actual}}$$

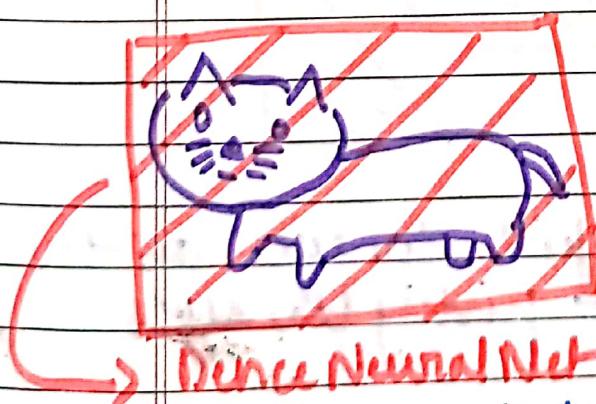
# DEEP Computer Vision

## • Image data

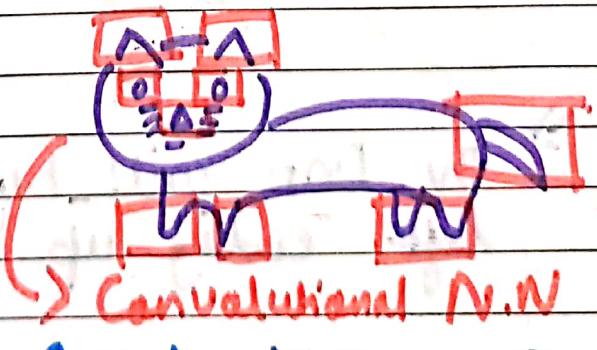


## Convolutional Neural Network

Normal Neural Network sees everything on a global scale  
But Convolutional NN focuses on small parts too

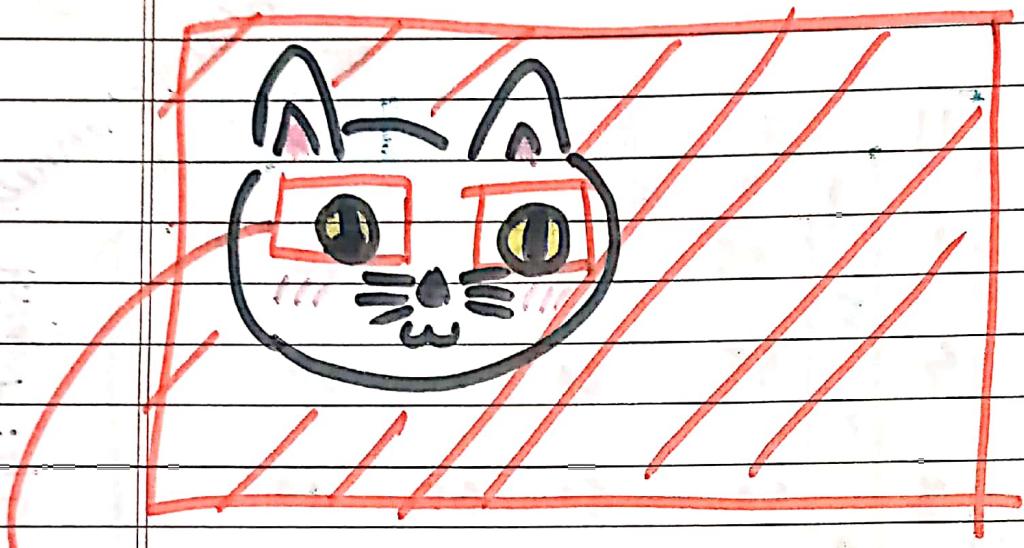


DNN  
Seeing as a whole.



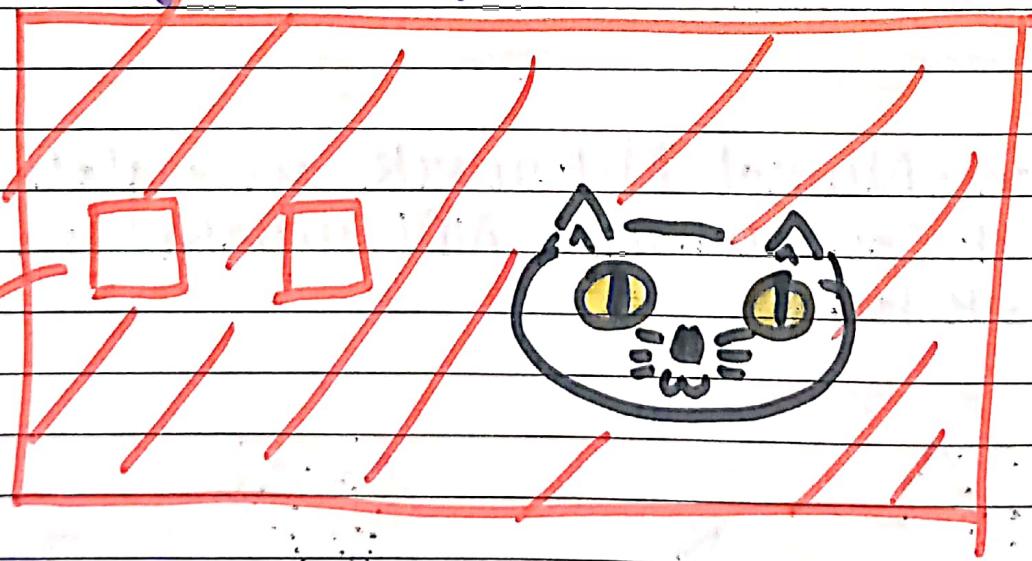
CNN  
Seeing in specific parts

## Dense neural net working



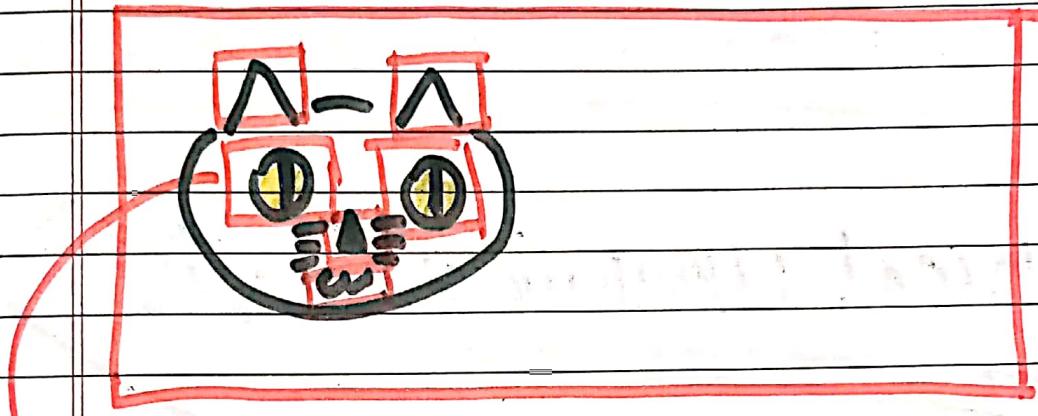
→ The Dnn will go through the image and learn that the eyes are always on the left of the image.

So, if the image is flipped



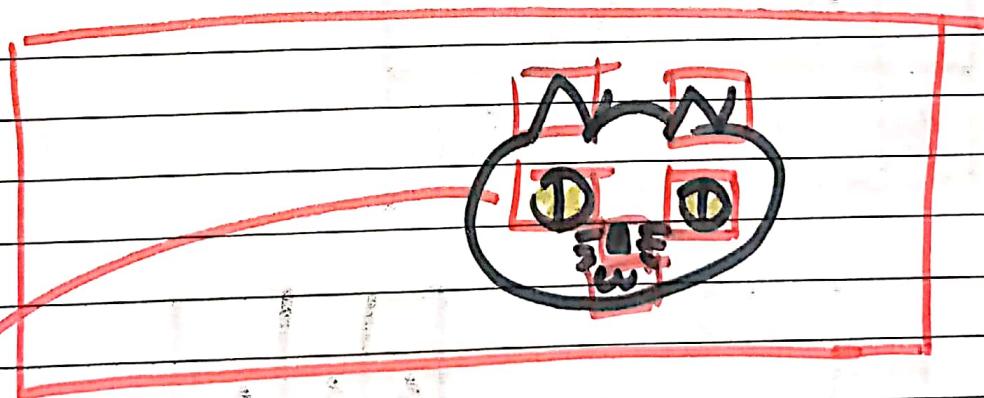
→ The Dnn will try to find the eyes on the left ending up not finding the eyes

## Convolutional N.N working



The convolutional N.N learns that the pattern of eyes are like this and pattern of Mouth/ Ears/ Nose are like this.

So, if eye img is flipped

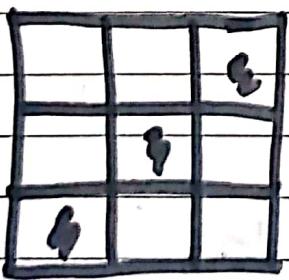
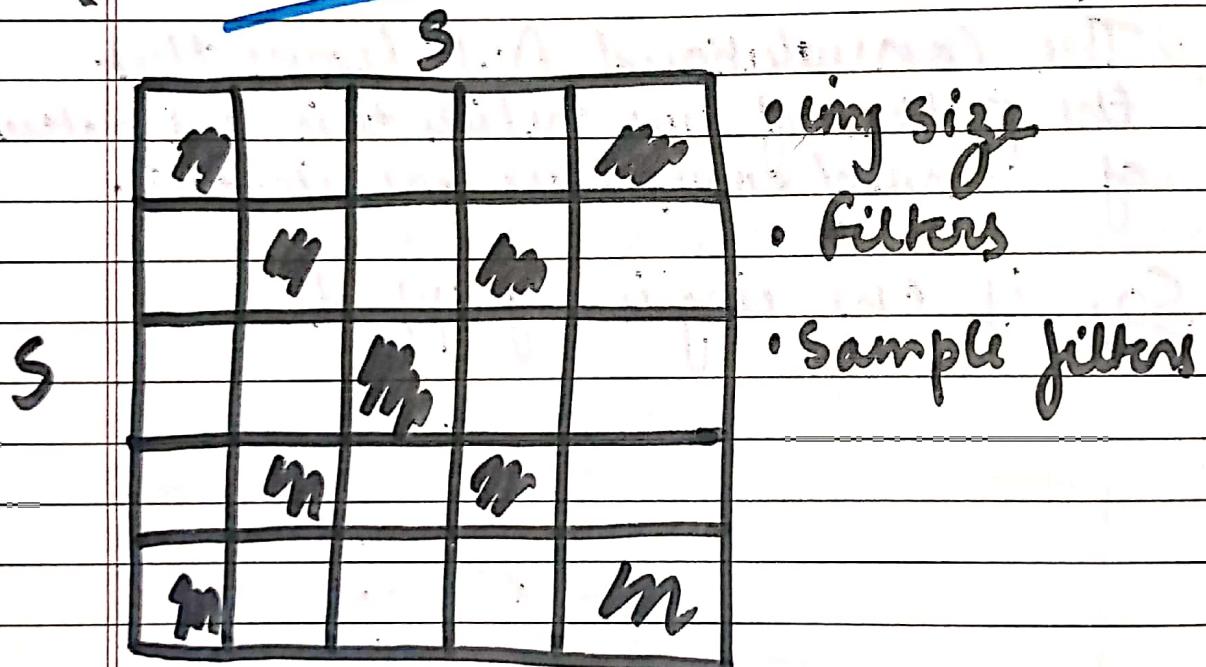


So, it will still be able to find the eyes as it has learned how eyes look

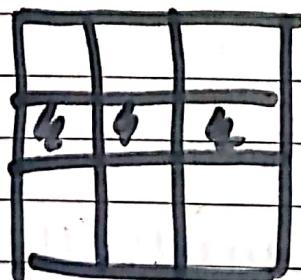
- #gives output a feature map that quantifies the presence of the filter's pattern at different locations

filter = Something we are looking for

## Technical Working of CNN



Filter 1

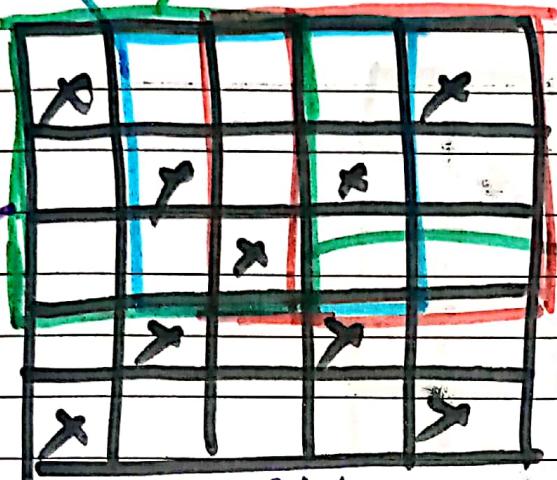


Filter 2

Takes a  $3 \times 3$  area and  
uses dot product over pixels

Date \_\_\_\_\_  
Page \_\_\_\_\_

TM  
 $3 \times 3$   
process  
goes on



x	x
x	x

$F_1$

x	x
x	x

$F_2$

few pixel match

0.1	0	1
0.1	0.12	0.1
1	0	0.1

The dot product

0.1	0.12	0.3
0.4	0.5	0
0.1	0.6	0.8

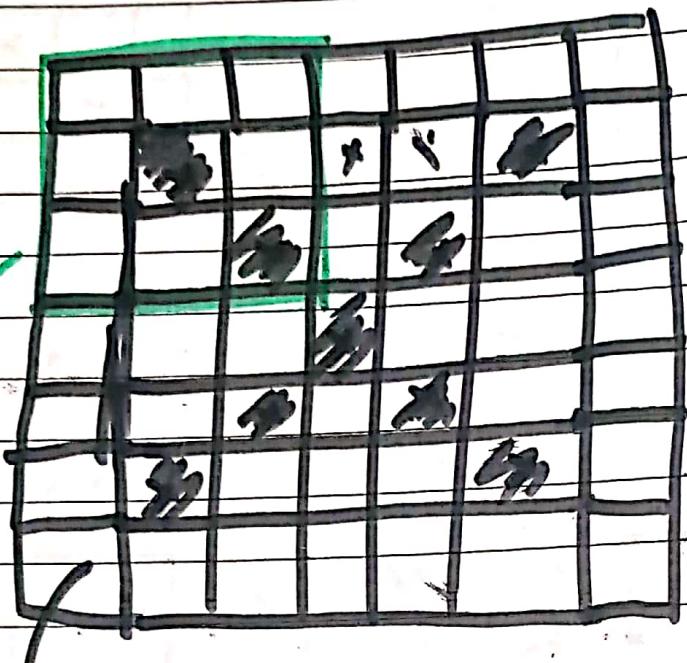
Complete different | Complete match

Output  
feature map

Next it will happen to these  
by taking set of numbers  
and calculating dot products

# Padding

Date \_\_\_\_\_  
Page \_\_\_\_\_



Padding.

→ So that all the pixels can be at the center atleast once even the edge pieces

~~STRIDE~~  
~~STRIDE~~  
~~STRIDE~~

STRIDE1  
stride1  
STRIDE2  
stride2

