DEEP LEARNING

We need to provide a good amount of data to create a model (ML model)
ex : identifying a dog - length of legs, nose size, etc

We need to populate all Features and create the model. Even after this, there is a chance for failure

→ Rule based system : (Around 1970)

Rules + Data] → MODEL $\xrightarrow{\text{gives}}$ output.

→ ML system : (1970 - 2010)

Input data + Output date] → MODEL $\xrightarrow{\text{creates}}$ Rules

Need to provide structured / tabular data (rows & columns)

→ DL Model :

- No need to provide structured data
- Finds out features on itself
- It is a ML model which works on unstructured data and derives the features on itself.

Image → Finds basic primitive features → uses these features to form more complex features

→ | / ^ → ○ □ △ → ... → Final dog picture

$\underbrace{\hspace{6cm}}_{\text{many layers are present}}$ - SEQUENTIAL LAYER ARCHITECTURE

(Good inputs, Good outputs)

We need to provide the data as accurately as possible - otherwise system gives bad outputs (GIGO - Garbage In Garbage Out

Learning happens in a sequential manner.

Every layer is responsible for different type of features. More layers means more deeper architecture.

DL model learns the representations from raw data.

- Supervised Learning : labelled data.

- Unsupervised Learning (clustered approach) : Unlabelled
Find similarity in data and cluster it

- DL model : Supervised learning
- DL is mostly used in unstructured data
- It does not have much applications in structured data (not required).
- DL is mostly designed for unstructured data. Incase of structured data, the features are already present - does not make sense to use DL. DL model can be used, but not required - other simpler models can be used

Low level Features →	Mid level Features →	High level Features
(Edges, dark spots)	(Eyes, ears, nose).	(Facial structure)
Conv Layer 1	Conv. Layer 2	Conv. Layer 3

Mostly in DL, the first few layers (lowe level features) are the same. As we go deeper the features will be different.

More layers / deeper architecture → Better accuracy

→ Difference b/w DL and ML:

① Data dependency : DL requires more data.

② Hardware dependency : requires speacialised hardwares like GPUs.

③ Training Time : higher for DL.

④ Feature selection : Features are automatically extracted by layers.

⑤ Interpretability : Very less or not present in DL model.

TENSOR FLOW

Sklearn helps use numpy functions more efficiently

numpy → Sklearn → used in CPUs.

numpy → TensorFlow → used in GPUs.

The background of Sklearn and TensorFlow is numpy functions (arrays).
TensorFlow functions are supported by both CPUs and GPUs. Sklearn supports only CPUs.

Mostly in DL, the First Few layers (lower level Features) are the same. As we go deeper the Features will be different.

More Layers / deeper architecture → Better accuracy

→ Difference b/w DL and ML:

1. Data dependency : DL requires more data
2. Hardware dependency : requires speacialised hardwares like GPUs.
3. Training Time : higher for DL.
4. Feature selection : Features are automatically extracted by layers.
5. Interpretability : Very Less or not present in DL model.

TENSOR FLOW //

Sklearn helps use numpy functions more efficiently

numpy → Sklearn → used in CPUs.

numpy → TensorFlow → used in GPUs.

The background of Sklearn and TensorFlow is numpy functions (arrays).
TensorFlow Functions are supported by both CPUS and GPUs. Sklearn supports only CPUs.

Keras - can run on top of TensorFlow.

TensorFlow consists of tensors (similar to arrays)

Constant :(cannot be changed) and variable tensors

Arithmetic operations : Tensors

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + 20$$

gets converted to

$$\begin{bmatrix} 20 & 20 & 20 \\ 20 & 20 & 20 \end{bmatrix}$$

$(2, 3, 2) + (2, 3)$

Need to change dimension of $(2, 3)$

(can be added on either side)

$$\begin{cases} (1, 2, 3) \\ \text{or} \\ (2, 3, 1) \end{cases}$$

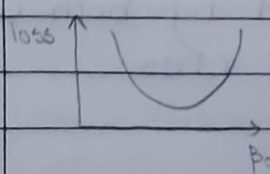
Simple Linear Regression

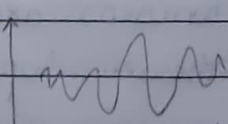
$$y = \beta_1 x + \beta_0$$

$\beta_0, \beta_1 \rightarrow$ weights

Loss Function : $mse = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$

Loss Functions helps get best fit line - From parabola of curve





For diff values of β_0, β_1 compute β_{new}
at that β_{new}, check loss.

$\frac{\partial Loss}{\partial \beta_0, \beta_1}$ where slope = 0 → that is the global minimum point

$$\beta_{0\,new} = \beta_{0\,old} - \eta \frac{\delta Loss}{\delta \beta_{0\,old}}$$

why minus? we need to minimise $\beta_{0\,new}$

① Vanilla gradient descent - batch processing
- slow process
- For ex:

β_0, β_1 Loss
:
:

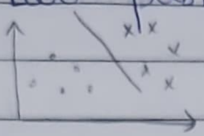
update β_0, β_1 ← Take avg
↓
Calculate loss again - slow process

② Stochastic gradient descent
In 1 epoch: updating 1000 times

There fore we apply vanilla GD in stochastic GD
- mini batch gradient descent (randomly picks batch size)
Perform stochastic - form batches → take avg
↓
Update β_0, β_1

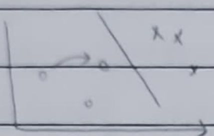
There should not be many batches

Linear Binary Classifier : when you can easily draw a line between two Classes / if you can easily differentiate two classes into two parts



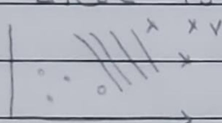
LOGISTIC REGRESSION ⇒

marks	GA	Pass/Fail
20	40	1
20	30	1
10	10	0
10	20	0



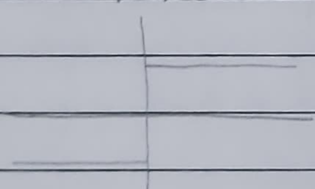
suppose one point is marked incorrectly, it shifts to new position → line also shifts

Suppose there is a huge gap in between

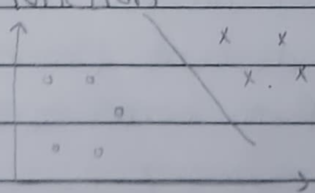


we can draw multiple lines in between
- difficult to determine
→ this is why we need to use optimisation process

we get step Function

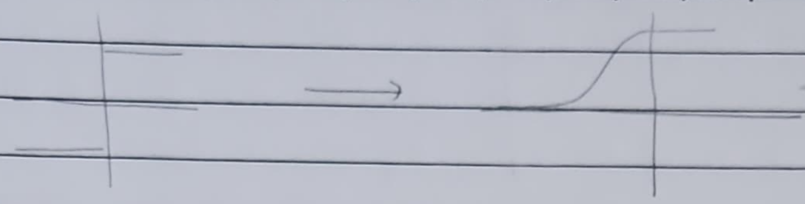


This step function needs to be changed into sigmoid function



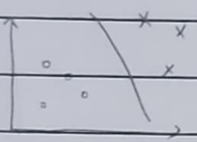
Take a line in between and calculate error
distance b/w point and line
→ depending upon that taken as 0 or 1 class

Likelihood is always taken for class 1
Best line is the one with the best likelihood

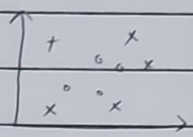


(sigmoid function)

Perceptron with Sigmoid Function = Logistic regression

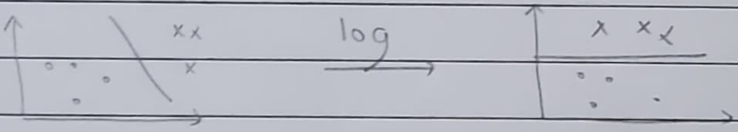


Linear



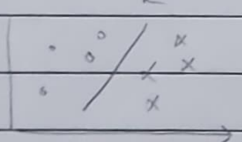
Non linear

Using multiple perceptrons to get the output – Multilayer Perceptron (MLP)

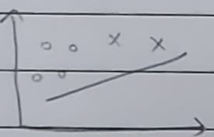


The backstory of Logistic regression – Perceptron

same point



P = 0.8



P = 0.6

$$P = 0.8 + 0.6 = 1.4$$

Cannot be 1.4 as Σ Prob = 1

↓ pass through sigmoid function

o/p