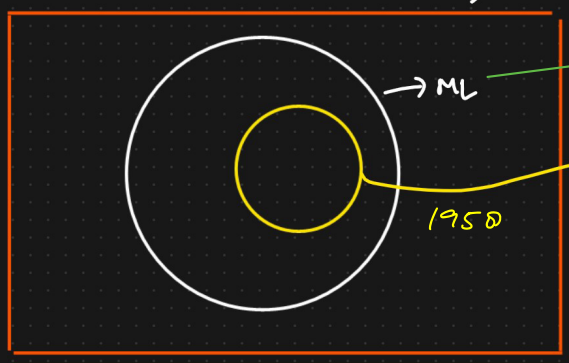
Deep Learning → recognizes patterns in the data and then makes predictions based on Multilayered Neural Network

Imp !! (mimics human brain)



→ Stats tools to Analyse Data.

Multilayered ⇒ Neural Networks

⇓

Mimic the Human Brain

ML - CPU
DL - GPU

Deep Learning → huge amounts of Data

① ANN → Artificial Neural N/W → Classification, Regression

② CNN → Convolutional Neural N/W → I/P : Images, Video frames → RCNN, MASKED RCNN, Detection, YOLO V5, V6, V7

Computer Vision
Objection Detection

③ RNN → Recurrent Neural N/W → NLP → NLP, Time Series

Sequential Data → I/P: Text, Time Series

⇓

{ Word Embedding, LSTM RNN, GRU RNN, Bidirectional LSTM RNN, Encoder Decoder, Transformers, BERT }

FRAMEWORK

TENSORFLOW KERAS

End to End Project

LSTM = Long Short Term Memory → precise & slow

GRU = Gated Recurrent Unit → Simple & Fast

Tensorflow → Deep learning library by Google

↳ Keras → API on top of Tensorflow

PyTorch → Deep learning library by Meta

② Why Deep Learning Is Becoming popular ?

2005 ⟶ Facebook, Youtube, Whatsapp, Linked, Twitter } ⇒ Social Media platform

2011-2012

⇓

DATA WAS GENERATED Exponentially

Image, Text, Documents, Videos

↳ Big DATA ⟶ HADOOP → unstructured DATA

⇓

2011 ⇒ Cloudera, Hortonworks

Netflix → movie streaming ⇓ platform

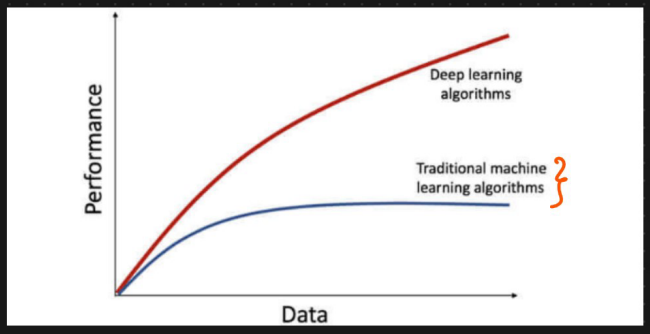
Recommendation System ⇓ Increase Revenue

2011-2012 :

① Hardware Requirement ⟶ GPU's cost↓↓↓ → NVidia GPU Priced↓↓

TRAIN MODELS

② Huge amount of data is getting generated → Deep Learning Models Perform well



③ Deep learning is been used in Many Domains

① Medical
② ECommerce
③ Retails
④ Marketing . . . . .

```
Artificial Neural Network (ANN)
    ├── Perceptron (single-layer)
    ├── Multi-Layer Perceptron (MLP)
    ├── Convolutional Neural Network (CNN)
    ├── Recurrent Neural Network (RNN)
    └── etc.
```

④ Frameworks Open source

Community size ↑↑

Tensorflow → Google

Pytorch → Facebook

More Research

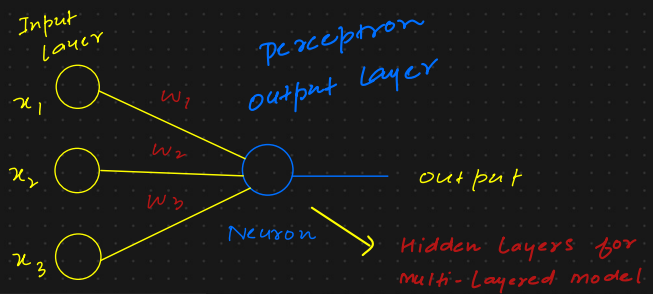
PERCEPTRON ; Simplest Model that Mimics Brain

Simplest form of Neural Network Unit.

How It Works:

1. **Inputs** → Like features (e.g., height, weight).
2. **Weights** → Importance of each input.
3. **Summation** → Adds the weighted inputs.
4. **Activation Function** → Decides the output (e.g., 0 or 1).

Output = Activation($(Input_1 \times Weight_1) + (Input_2 \times Weight_2) + \dots + Bias$)



Imp.

Quick Recap:

Type	Hidden Layer?	Example
Single-layer NN	❌ No	Perceptron
Multi-layer NN (MLP)	✅ Yes	Deep Learning models

Imp

Steps

Input → weight → Summation → Activation functions → output

$(Input_1 \times w_1 + I_2 \times w_2 + Bias)$

V.V. Imp.

- Neurons don't "think" — **weights decide how much each input matters**
- Weights are adjusted through **training** (using backpropagation)
- This is how deep learning models **learn patterns**

So, the entire process is called Forward Propagation

Actual Equation

$y = \sum x_i w_i + bias$ ⇒ $mx + c$ (think of it as bias.)

Important so we start from some default value and not just zero

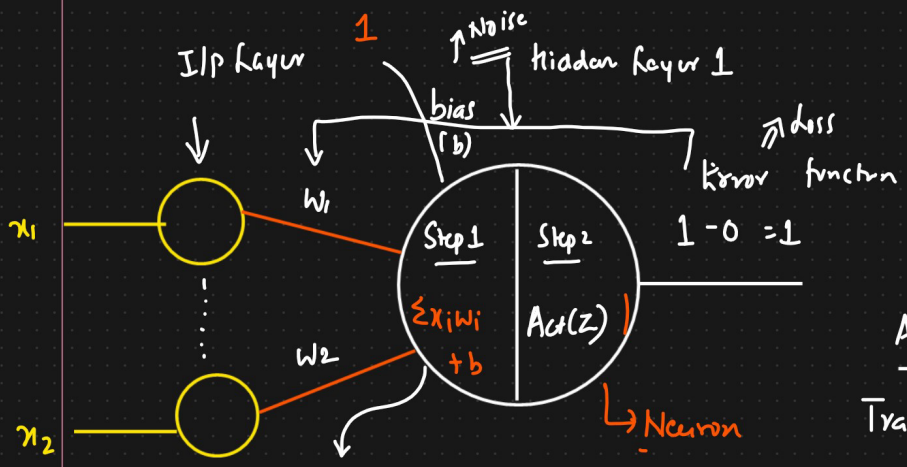
③ Perceptron [Artificial Neuron or Neural Network Unit]

① Input layer ✓
→ No hidden layer in perceptron
② Hidden layer ✓
③ Weights ✓
④ Activation function ✓

[Single layered NN]
⇓
Binary classifier

DATASET

	x_1 Id	x_2 No. of study hours	o/p Pass/Fail
→	95	3	0
→	110	4	1
→	100	5	1



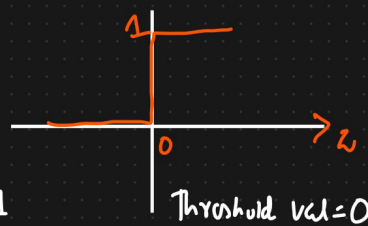
Activation function

Transform the output between 0 to 1 / -1 to 1

Imp $Z = w_1x_1 + w_2x_2 + b \rightarrow \beta_0 x_1 + \beta_1 x_1$ similar

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

Step Function

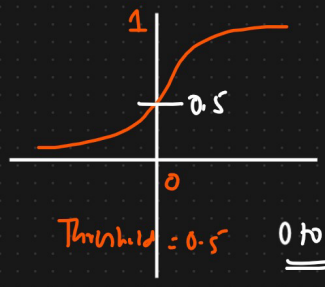


0 to 1

Threshold val = 0

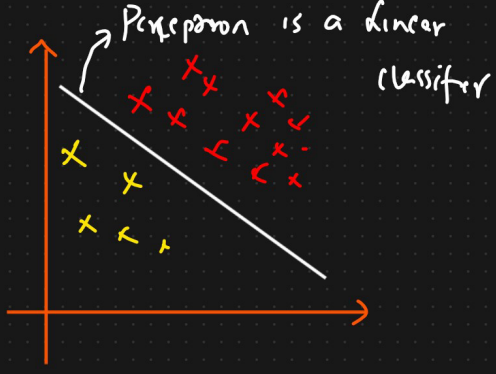
$$\begin{cases} 0 & Z \le 0 \\ 1 & Z > 0 \end{cases}$$

Sigmoid Function



Threshold = 0.5 — 0 to 1

$$\begin{cases} 1 & Z > 0.5 \\ 0 & Z \le 0.5 \end{cases}$$



Step 1

$$Z = \sum_{i=1}^{n} w_i x_i + b \rightarrow \text{Bias}$$

$$Z = b + w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n$$

$$y = mx + c$$

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 \; - - - + \beta_n x_n$$

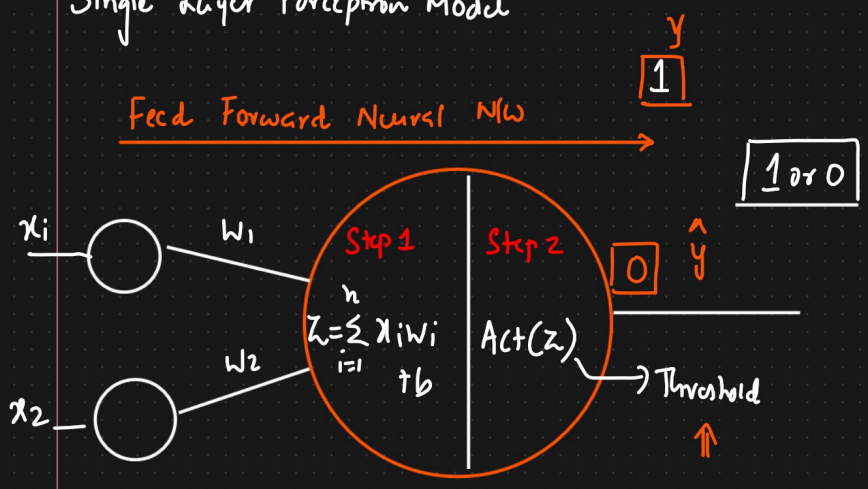
linear problem Statement

Perceptron Models

- Single Layer Perceptron Model
- Multi Layered Perceptron Model [ANN]

Single Layer Perceptron Model

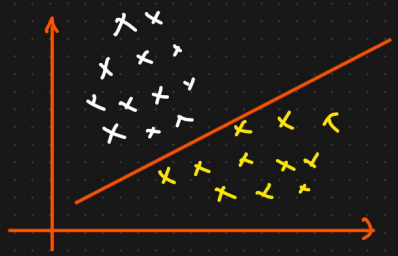
Feed Forward Neural NW



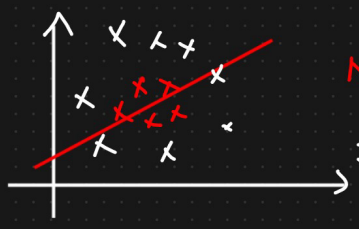
Multi Layered Perceptron Model

1. Forward Propagation
2. Backward "
3. Loss functions
4. Activation functions
5. Optimizers

Linear Seperable problem



Binary classification ⇓ linearly separable



Non linear seperable ⇒ Multi Layered NN

Activation function

$$\frac{1}{e^{-y}} = \frac{1}{e^{-(x_i w_i + b)}}$$ → gives output in 0 to 1.

$\geq 0.5 = 1$

$< 0.5 = 0$

Summary

Deep learning → mimic human brain with the help of Neural Networks

working ;

Input Data → Calculate weights → Add weights + Bias (Sum)

$\sum x_i w_i + b$

Activation functions to get probability → Output

pros / cons →

More Accurate
Solves complex problems
Work on huge Data
- Requires GPU's
- More Time
- Blackbox (Hidden Layers)

Neural Networks

Linear classification
Perceptron

LSTM

Single layered Neural Network	Multi layered Neural Network
• Single Layer	• Multiple Layers
• No hidden layers	• Hidden Layers
• only Linear Tasks	• Complex & Non Linears
• And / OR Gate	• Image Classifier / NLP
• Layer = 1	• Layers ≥ 2

Q1. What is Non-Linearity Concept ? Usually Activation functions

ability of a model to learn complex patterns, like curves, shapes etc. not just simple line relationships.

imp !!

In short:

> **Non-linearity** gives neural networks the **power to learn complex, real-world relationships.**

Without it, even a deep network would just behave like a **linear regression** model.

Q11. what do you understand by activation functions ?

decides whether neuron should be activated.

- introduces probability & non linearity
- helps model to learn beyond linear data

✅ Activation Function Comparison Cheat Sheet

Activation	Used For	Non-Linearity	Probability Output	Common Use Case
ReLU	Hidden Layers (default)	✅ Yes	❌ No	Enables deep networks to learn patterns
Leaky ReLU	Hidden Layers (ReLU alternative)	✅ Yes	❌ No	Solves "dead ReLU" problem
Sigmoid	Output Layer (Binary Classification)	✅ Yes	✅ Yes	Converts output to probability (0–1)
Softmax	Output Layer (Multi-class Classification)	✅ Yes	✅ Yes	Distributes output as class probabilities
Tanh	Sometimes in Hidden Layers	✅ Yes	❌ No	Like Sigmoid but zero-centered

Forward & Backward Propagation

🔵 Forward Propagation (Prediction Phase)

🔹 **Purpose**	**To compute the output (prediction) of the model.**
🔹 Direction	From **input → hidden layers → output**

🔹 What happens?

- Input data is passed through the network.
- At each layer:
  - Weighted sum is calculated: `Z = W·X + b`
  - Activation function is applied: `A = activation(Z)`
- Final output (ŷ) is produced.

| 🔹 Example | Predicting whether an email is spam or not. |

🔴 Backward Propagation (Learning Phase)

🔹 **Purpose**	**To update the weights to reduce error/loss.**
🔹 Direction	From **output → hidden layers → input** (reverse)

🔹 What happens?

- Compute the **loss** between predicted output and actual output.
- Apply **chain rule** of calculus (using gradients):
  - Compute how much each weight contributed to the error.
- Calculate gradients (`dW`, `db`) and update weights using **gradient descent**.

| 🔹 Example | Adjusting model so it better predicts spam in future.|

I/p → weights → Σ xiwi + b → O/p → O/p → Loss fx → Activators → min. loss

Activation

Gradient Descent update

→ What is ReLU Rectified Linear Unit

✅ ReLU Meaning (In Simple Terms)

- **If the input is positive**, keep it as it is.
- **If the input is negative**, change it to zero.

👉 In short:

> **ReLU(x) = x if x > 0, else 0**

⚠️ Limitation: → Important

- **"Dead Neurons"**: If a neuron always receives negative inputs, it outputs 0 forever and stops learning.
  - To solve this, we use **Leaky ReLU** sometimes.

Summary:

> **ReLU = Let the positive values pass, block the negative ones.**
> It's simple, fast, and the default choice for hidden layers in deep learning.

Q1 what is Leaky ReLU ? Leaks negative values

Keeps neurons alive

🔍 What is Leaky ReLU?

Leaky ReLU is a modified version of **ReLU** that fixes its main problem:

> It **allows a small, non-zero output** when the input is negative.

⚠️ Why do we need it?

Problem with ReLU:

- ReLU outputs **0 for all negative inputs**.
- Some neurons may get stuck and **never activate or learn** — called the **"dead ReLU" problem**.

✅ What Leaky ReLU does:

- For **positive inputs**: same as ReLU → output = input.
- For **negative inputs**: output a **small negative value** (like 0.01 × input) instead of 0.

🧠 Intuition:

Input (x)	ReLU(x)	Leaky ReLU(x)
-3	0	-0.03
0	0	0
2	2	2

✅ Benefits of Leaky ReLU:

Feature
Keeps neurons alive
Allows small gradients
Works better for deep networks

Q1 what is Tanh hyperbolic Tangent

🔍 What is tanh in Deep Learning?

tanh, short for **hyperbolic tangent**, is an **activation function** used in neural networks — especially in hidden layers.

-1 to 1

unlike Softmax

0 to 1

✅ Key Characteristics:

Property	Description
Output Range	-1 to 1
Shape	S-shaped (like sigmoid, but zero-centered)
Non-linearity	✅ Yes – allows the network to learn complex patterns
Used In	Hidden layers (especially in RNNs or older networks)

🧠 Why use tanh?

- Unlike sigmoid (which outputs between 0 and 1), **tanh is centered around 0**.
- This makes optimization **easier and faster**, especially for data with negative values.

Conclusion;

1. I/p layers
2. Weight
3. $\Sigma x_i w_i$ + Bias
4. Activation function

} Forward propagation

5. Loss function $(\hat{y} - y)$
6. Minimize Loss with Optimizers
7. Update weights

} Backward propagation

Imp.

Machine learning = White Box (except Random Forest).

Deep learning = Black Box Models

```
Artificial Neural Network (ANN)
    ├── Perceptron (single-layer)
    ├── Multi-Layer Perceptron (MLP)
    ├── Convolutional Neural Network (CNN)
    ├── Recurrent Neural Network (RNN)
    └── etc.
```