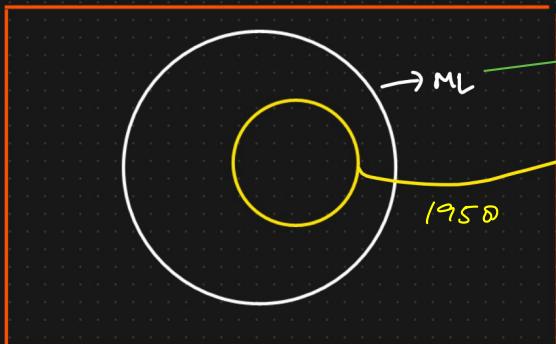


Deep learning → recognizes patterns in the data and then makes predictions based on Multilayered Neural Network

Imp. (mimics human brain)

AI ⇒ Artificial Intelligence



→ 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 }
- ② CNN → Convolutional Neural N/w → I/P : Image, Video → RCNN, MASKED RCNN, frames
Detection, YOLO vs V6, V7
- ③ RNN → Recurrent Neural N/w → NLP ↗ NLP, Time Series

Computer Vision
Object Detection
↑

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, LinkedIn, Twitter } ⇒ Social Media platform
↓
2011-2012

DATA WAS GENERATED

Images, Text, Documents,
Videos ↳ **Big DATA** → Exponentially

Videos ↳ **Big DATA** → HADOOP → unstructured
DATA

↓
2011 ⇒ Cloudera, Hortonworks

Netflix → movie streaming
↓ Platform

2011-2012 :

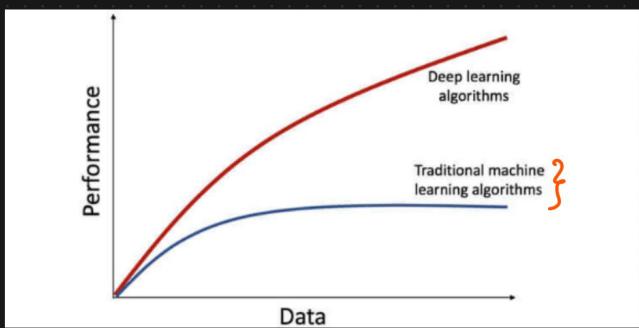
① Hardware Requirement → GPU's cost ↓ ↓ → Nvidia GPU

TRAIN MODELS

Price ↓

Increase Revenue

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 E-commerce
- ③ Retail
- ④ Marketing . . .

④ Frameworks Open Source

Community size ↑

Tensorflow



Google

Pytorch



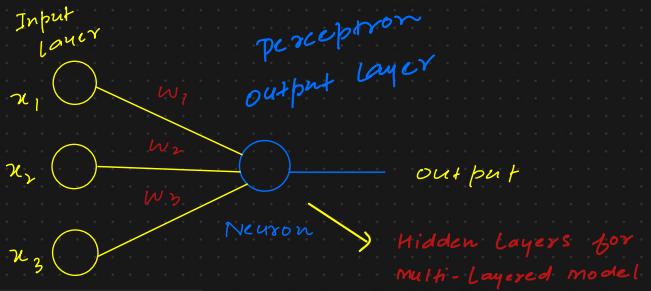
Facebook

More Research

Perceptron; Simplest Model that Mimics Brain
Simplest form of neural network Unit.

🧠 How It Works:

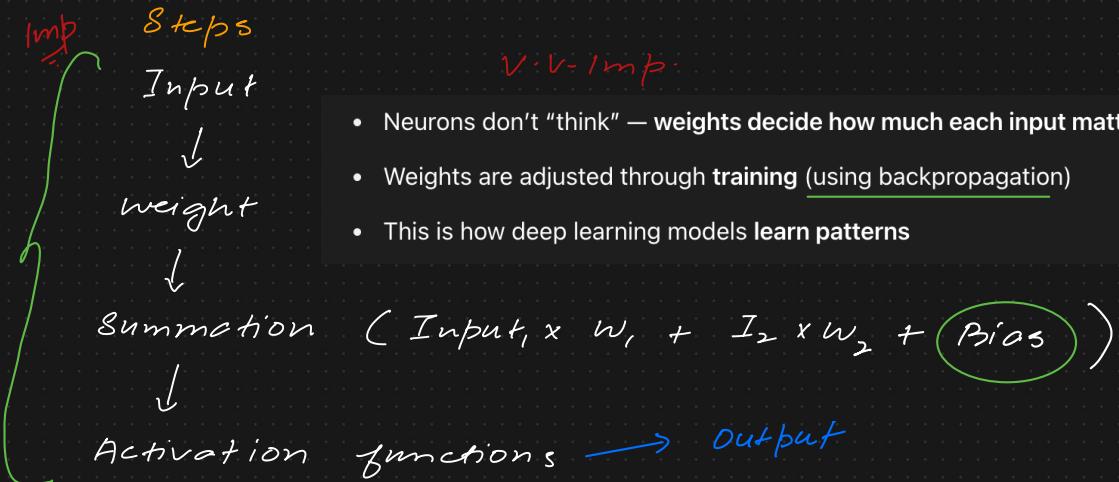
1. Inputs → Like features (e.g., height, weight).
 2. Weights → Importance of each input.
 3. Summation → Adds the weighted inputs.
 4. Activation Function → Decides the output (e.g., 0 or 1).
- 💡 Output = Activation((Input₁ × Weight₁) + (Input₂ × Weight₂) + ... + Bias)



Imp.

🧠 Quick Recap:

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



V: V-1mp.

- 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

$$y = \sum x_i w_i + \text{bias} \Leftrightarrow m x + C \quad \text{think of it as bias.}$$

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

③ Perceptron [Artificial Neuron or Neural Network Unit]

① Input layer ✓ [Single layered NN]

→ no hidden layer

② Hidden layer ✓ in perceptron ↓

③ Weights ✓

Binary classifier

④ Activation function ✓

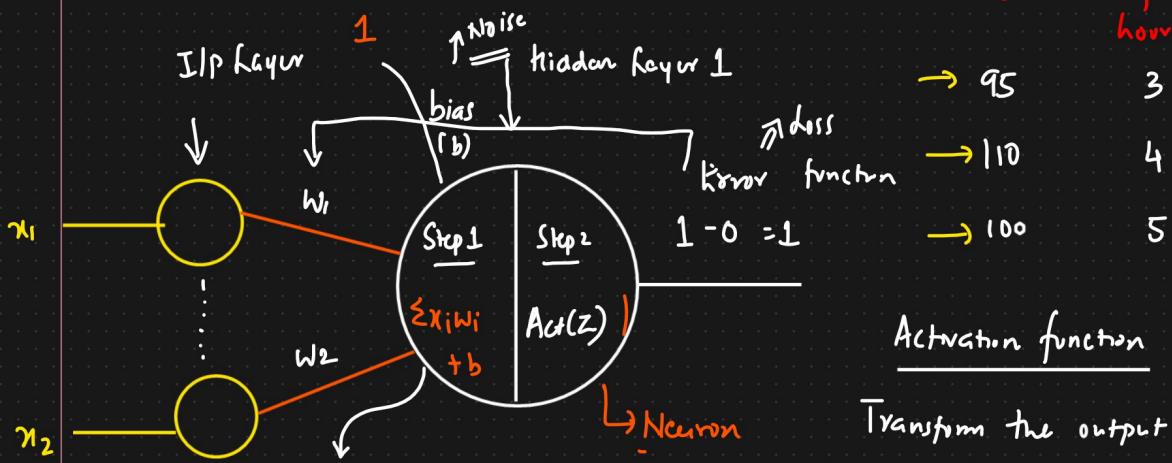
DATASET

x_1 x_2
ID No. of study hours

O/P Pers/Fac.

→ 95 3
→ 110 4
→ 100 5

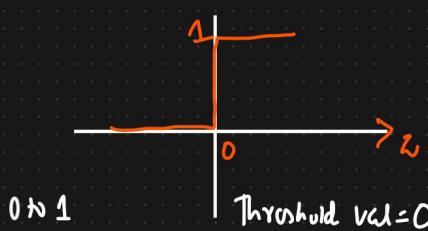
0 1
1 1
1



Imp. $z = w_1 x_1 + w_2 x_2 + b \rightarrow \beta_0 x_1 + \beta_1 x_2$

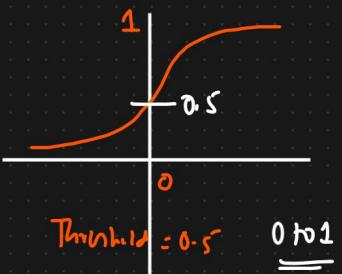
$$z = \sum_{i=1}^n x_i w_i + b$$

Step Function

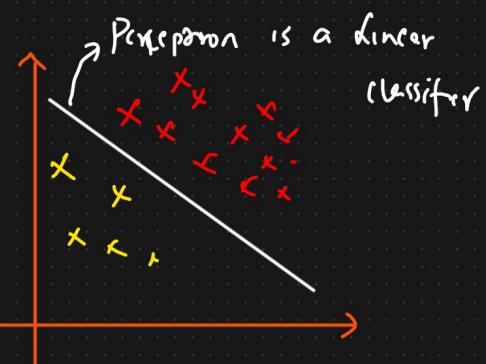


$$\begin{cases} 0 & z \leq 0 \\ 1 & z > 0 \end{cases}$$

Sigmoid Function



$$\begin{cases} 1 & z > 0.5 \\ 0 & z \leq 0.5 \end{cases}$$



Step 1

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

$$z = b + w_1 x_1 + w_2 x_2 + w_3 x_3 + \dots + w_n x_n$$

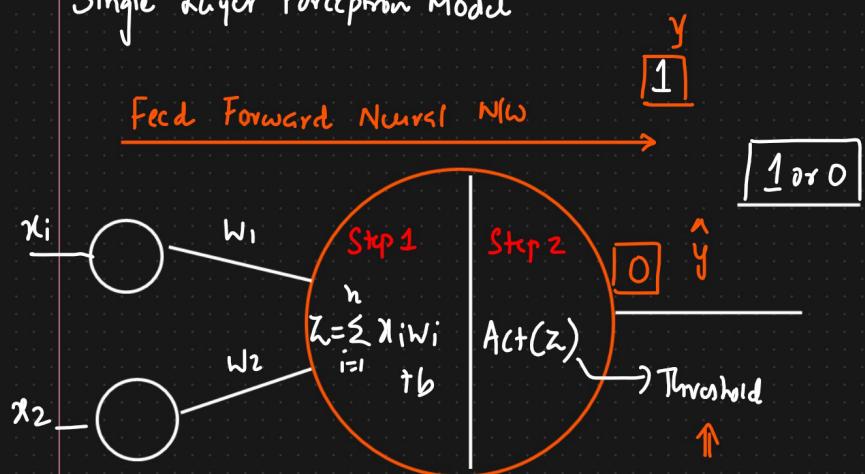
$$y = mx + c$$

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

↓
linear
problem
statement

Perceptron Models

Single layer Perceptron Model

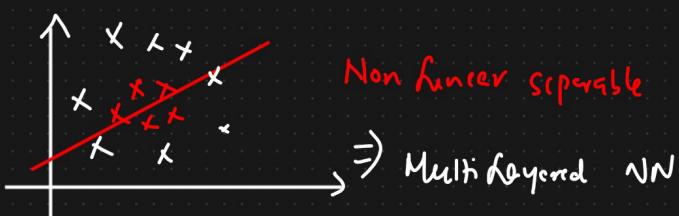
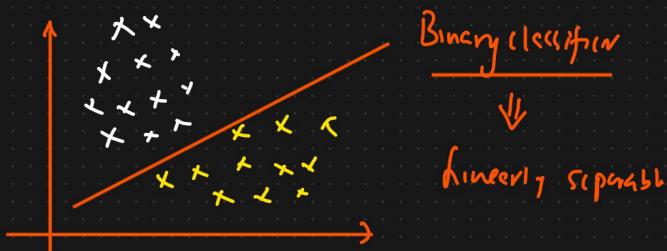


↓
[ANN]

Multi Layered Perceptron Model

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

Linear Separable problem



Activation function

→ $\frac{1}{e^{-y}} = \frac{1}{e^{-(w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n)}}$ → 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)

Output ← Activation functions
to get probability

$$\sum x_i w_i + b$$

- 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

- Single layer

- No hidden layers

- Only linear tasks

- And / OR Gate

- Layer = 1

multi. layered

Neural Network

- Multiple layers

- Hidden layers

- Complex & non linear

- Image classifier / NLP

- Layers ≥ 2

Qn. What is non-linearity concept? Usually Activation functions ability of a model to learn complex patterns; like curves, shapes etc. not just simple linear 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.

Qn. 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: $Z = W \cdot X + b$
 - Activation function is applied: $A = \text{activation}(Z)$
- Final output (\hat{y}) is produced.

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

Backward Propagation (Learning Phase)

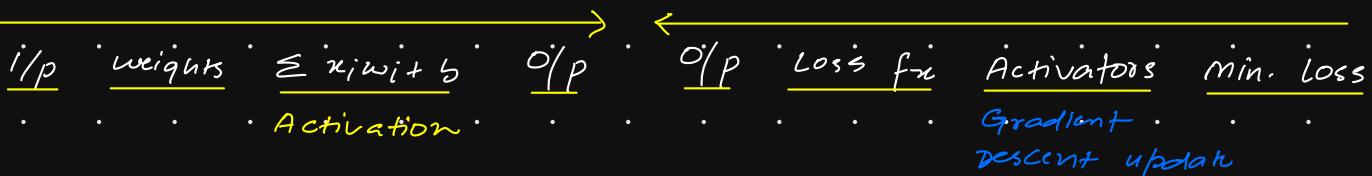
♦ Purpose To update the weights to reduce error/loss.

♦ 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. |



→ 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:

$$\text{ReLU}(x) = x \text{ if } x > 0, \text{ else } 0$$

⚠ 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.

Q: 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 \rightarrow output = input.
- For negative inputs: output a small negative value (like $0.01 \times$ 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

Q: 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.

✓ 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.

-1 to 1
unlike softmax

0 to 1

Conclusion;

1. I/P layers
 2. weight
 3. $\sum x_i w_i + \text{Bias}$
 4. Activation function
 5. Loss function ($\hat{y} - y$)
 6. minimize loss with optimizers
 7. update weights
- } Forward propagation
- } Backward propagation

Imp.

Machine learning = white Box
(except Random Forest)

Deep learning = Black Box models