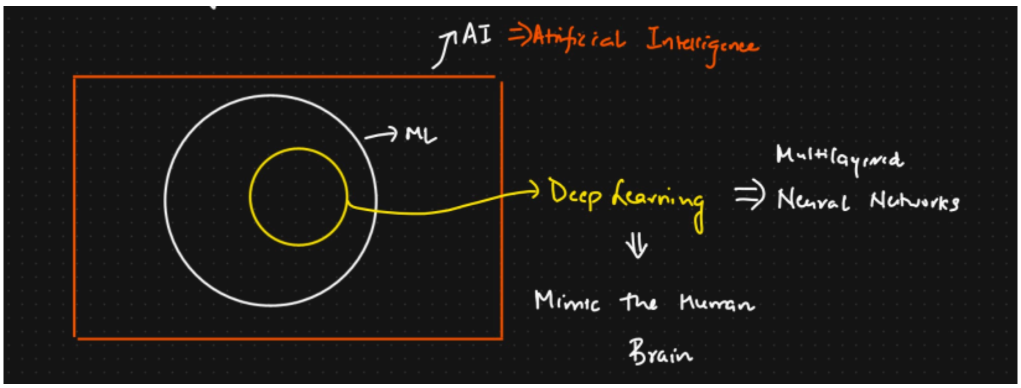
Deep Learning

1. What is Deep Learning?

- **Subset of Machine Learning:** Deep Learning (DL) is a specialized subfield of Machine Learning (ML), which itself is a subfield of Artificial Intelligence (AI).
- **Inspired by the Human Brain:** It uses artificial neural networks with multiple layers (hence "deep") to learn and make decisions, drawing inspiration from the structure and function of the human brain's interconnected neurons.
- **Learning Representations:** Instead of being explicitly programmed, deep learning models learn hierarchical representations of data. Lower layers learn simple features, and higher layers combine these to learn more complex concepts.
- 

🏗️ Key Components

1. Neurons and Layers

- **Neuron**: Basic unit that receives inputs, applies a weight and bias, passes through an activation function.

- **Layers**:
  - **Input Layer**: Receives raw features
  - **Hidden Layers**: Process and transform inputs
  - **Output Layer**: Produces final predictions

2. Activation Functions

Introduce non-linearity. Common ones:

- ReLU (Rectified Linear Unit)
- Sigmoid
- Tanh

- Softmax (for classification)

3. Loss Function

Measures the error between predicted and true values.

- Examples: MSE (regression), Cross-Entropy (classification)

4. Optimization

Backpropagation + Gradient Descent to adjust weights

- Optimizers: SGD, Adam, RMSprop

🛠️ Popular Architectures

Type	Description	Example Use
Feedforward NN (FNN)	Basic dense layers	Tabular data
Convolutional NN (CNN)	Spatial hierarchies	Image classification
Recurrent NN (RNN)	Sequence modeling	Language modeling
Transformers	Attention-based, parallel	NLP, Vision
Autoencoders	Encoding/decoding for compression	Anomaly detection
GANs	Generator vs Discriminator	Image synthesis

📚 Frameworks & Tools

Framework	Language	Strength
TensorFlow	Python	Industry-grade, scalable
PyTorch	Python	Research-friendly, dynamic
Keras	Python	High-level, easy prototyping
ONNX	Interop	Model deployment/interchange

📈 Use Cases

- **Computer Vision**: Image classification, detection, segmentation
- **NLP**: Sentiment analysis, translation, chatbots
- **Speech**: Recognition, synthesis
- **Recommendation Systems**
- **Medical Diagnosis**, **Finance**, **Autonomous Vehicles**

Core Architectural Paradigms (High-Level):

- **Artificial Neural Networks (ANNs):**
  - **Concept:** The foundational building blocks. Consist of interconnected nodes (neurons) organized in layers (input, hidden, output). Each connection has a weight that is learned during training.
  - **What we study:** Neuron models (e.g., perceptron), activation functions (Sigmoid, ReLU, Tanh), feedforward propagation, backpropagation (for learning weights), loss functions, and optimization algorithms (e.g., Gradient Descent and its variants).
  - **Relevance:** Basis for all other deep learning architectures. Used for simpler classification and regression tasks on structured data.
- **Convolutional Neural Networks (CNNs or ConvNets):**
  - **Concept:** Specialized for processing grid-like data, such as images. Employ operations like convolution (to detect local features like edges, textures) and pooling (to downsample and create invariance).

    - **What we study:** Convolutional layers, filters/kernels, pooling layers (max, average), padding, stride, receptive fields, and well-known architectures (e.g., LeNet, AlexNet, VGG, ResNet, Inception).
    - **Relevance:** Dominant in computer vision tasks (image classification, object detection, segmentation), but also applied to other domains like NLP.
- **Recurrent Neural Networks (RNNs):**
    - **Concept:** Designed to handle sequential data (e.g., text, time series, speech) by having connections that form directed cycles, allowing information from previous steps to persist (memory).
    - **What we study:** Handling sequences, vanishing/exploding gradients problem, Long Short-Term Memory (LSTM) units, Gated Recurrent Units (GRU), sequence-to-sequence models, attention mechanisms (often studied alongside RNNs/Transformers).
    - **Relevance:** Crucial for Natural Language Processing (machine translation, sentiment analysis), speech recognition, time series forecasting.
- **Transformers:**
    - **Concept:** A more recent architecture (emerged prominently around 2017) that has revolutionized NLP and is increasingly used in other domains like vision. Relies heavily on "self-attention" mechanisms to weigh the importance of different parts of the input sequence, allowing for parallel processing of sequences (unlike traditional RNNs).
    - **What we study:** Self-attention, multi-head attention, positional encodings, encoder-decoder structures (e.g., BERT, GPT).
    - **Relevance:** State-of-the-art for many NLP tasks and showing great promise in computer vision (Vision Transformers - ViT) and other areas.

5. Key Enabling Concepts & Tools:

- **Transfer Learning:** Reusing a pre-trained model (trained on a large benchmark dataset like ImageNet or a massive text corpus) as a starting point for a new, related task. This is highly effective when you have limited data for your specific problem.
- **Embeddings:** Dense vector representations of discrete inputs (like words or categories) learned by the network, capturing semantic relationships.
- **Regularization Techniques:** Methods (e.g., L1/L2 regularization, Dropout, Batch Normalization) to prevent overfitting, where the model performs well on training data but poorly on unseen data.
- **Optimization Algorithms:** Advanced optimizers beyond basic SGD, such as Adam, RMSprop, AdaGrad, which help in efficiently navigating the complex loss landscapes of deep networks.
- **Hyperparameter Tuning:** The process of finding the optimal set of hyperparameters (e.g., learning rate, number of layers, number of neurons per layer) for a model, often requiring systematic experimentation.

Deep Learning

Computer Vision
Objection Detection
⇑

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

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

I/P : Text, Time Series
⇓
{ Word Embedding, LSTM RNN, GRU RNN, Bidirectional LSTM RNN, Encoder Decoder, Transformers, BERT }

FRAMEWORK

TENSORFLOW

End to End Project

② Why Deep Learning Is Becoming popular ?

2005 ⟶ Facebook, Youtube, Whatsapp, Linkedin, Twitter } ⇒ Social Media platform
⇓
DATA WAS GENERATED Exponentially

2011-2012

Images, Text, Documents, Videos ↳ Big DATA ⟶ HADOOP ⟶ unstructured DATA
⇓
2011 ⇒ Cloudera, hortonworks

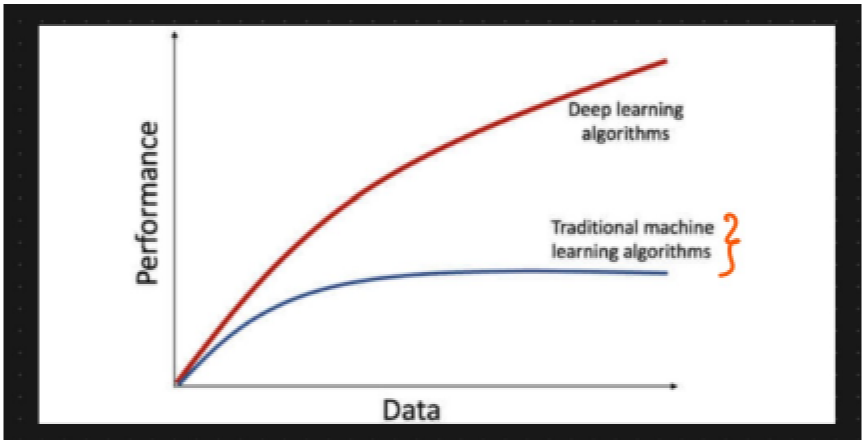
Netflix ⟶ movie streaming platform
⇓
Recommendation System
⇓
Increase Revenue

2011-2012 :

① Hardware Requirement ⟶ GPU's cost↓↓↓ ⟶ NVidia GPU Price↓↓↓

TRAIN MODELS

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



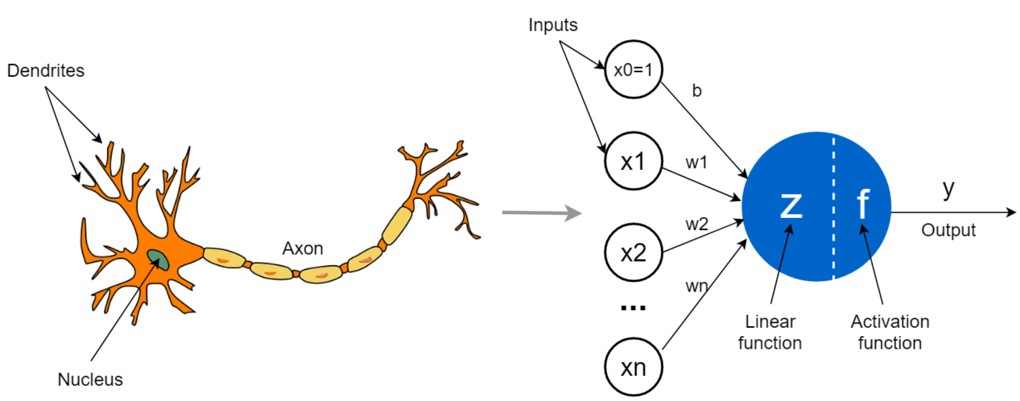
The Perceptron

1. Introduction & History:

- The **Perceptron** is the simplest form of a neural network, a foundational algorithm for **supervised learning** of **binary classifiers**.
- It's a type of **linear classifier**, meaning it learns a linear decision boundary to separate two classes.
- It's inspired by a single biological neuron, aiming to mimic basic information processing.

2. Analogy to a Biological Neuron:

- **Inputs:** Like dendrites receiving signals.
- **Weights:** Represent the strength or importance of each input signal (synaptic strength).
- **Summation:** The cell body (soma) integrates the incoming signals.
- **Activation Function (Threshold):** If the integrated signal exceeds a certain threshold, the neuron "fires" (sends an output signal down its axon).



3. Components of a Perceptron:

- **Inputs (x1,x2,...,xn):** A vector of numerical features representing an instance.
- **Weights (w1,w2,...,wn):** A vector of real-valued numbers, one for each input feature. These are learned during training.
- **Bias (b):** A special weight that is not associated with any input feature (or can be thought of as a weight for an input that is always +1). It allows shifting the decision boundary away from the origin, increasing model flexibility.
- **Summation Function (Net Input):** Calculates the weighted sum of the inputs plus the bias.
  - Mathematically: $z = (\sum_{i=1}^{n} w_i x_i) + b = \mathbf{w} \cdot \mathbf{x} + b$
- **Activation Function (Step Function):** Determines the output of the Perceptron based on the net input. The classical Perceptron uses a **Heaviside step function**:
  - Output is 1 if z≥θ (threshold).
  - Output is 0 (or sometimes -1) if z<θ.
  - Often, the threshold θ is incorporated into the bias term, so the activation function becomes:
    - Output is 1 if z≥0.
    - Output is 0 (or -1) if z<0.

How a Perceptron Works (Forward Pass):

1. **Receive Inputs:** The Perceptron takes an input vector x.
2. **Compute Weighted Sum:** Each input xi is multiplied by its corresponding weight wi. These products are summed up, and the bias b is added: z=w1x1+w2x2+...+wnxn+b
3. **Apply Activation Function:** The result z is passed through a step activation function to produce the final output ypred (typically 0 or 1 for binary classification).
   - ypred=1 if z≥0
   - ypred=0 (or -1) if z<0
4. This output ypred is the Perceptron's prediction for the given input.

5. The Perceptron Learning Algorithm:

The goal of the learning algorithm is to find the optimal weights (w) and bias (b) that correctly classify the training examples.

1. **Initialization:**
   - Initialize weights (w) and bias (b) to small random values or zeros.
   - Define a **learning rate (η)**: A small positive value (e.g., 0.01, 0.1) that controls the step size of weight updates.
2. **Iterate through Training Data (Epochs):** For each training example (x(j),d(j)) where x(j) is the input vector and d(j) is the desired (true) output label (e.g., 0 or 1):
   - **a. Calculate Predicted Output:**
     - Compute the net input: z(j)=w⋅x(j)+b
     - Apply the step function to get the predicted output y(j).

- **b. Calculate Error:**
  - The error is the difference between the desired output and the predicted output: error(j)=d(j)−y(j)
- **c. Update Weights and Bias:** If error(j)=0 (i.e., the prediction is wrong):

  Weight update rule: For each weight w_i: $w_i(\text{new}) = w_i(\text{old}) + \eta \cdot error^{(j)} \cdot x_i^{(j)}$

  Bias update rule: $b(\text{new}) = b(\text{old}) + \eta \cdot error^{(j)}$

  - If $error^{(j)} = 0$ (prediction is correct), no update is needed for that example.

3. **Repeat:** Repeat step 2 for a fixed number of epochs (passes through the entire training dataset) or until all training examples are correctly classified, or some other stopping criterion is met.

Intuition behind the update rule:

If $d^{(j)} = 1$ and $y^{(j)} = 0$ (false negative, error = 1): Weights are increased for positive inputs $x_i^{(j)}$, pushing z higher.

If $d^{(j)} = 0$ and $y^{(j)} = 1$ (false positive, error = -1): Weights are decreased for positive inputs $x_i^{(j)}$, pushing z lower.

- The learning rate η scales the magnitude of the update.

Convergence: The Perceptron learning algorithm is guaranteed to converge and find a solution (a separating hyperplane) **if and only if** the training data is **linearly separable**. If the data is not linearly separable, the algorithm will typically not converge and may oscillate.

6. Mathematics Summary:

- **Input Vector:** x=[x1,x2,...,xn]
- **Weight Vector:** w=[w1,w2,...,wn]
- **Bias:** b
- **Net Input (Weighted Sum):** $z = \mathbf{w}^T\mathbf{x} + b = \sum_{i=1}^{n} w_i x_i + b$
- **Activation (Output y for binary classification, labels 0 and 1)**
- **Error (for desired output d and predicted output y):** e=d−y
- **Weight Update:** wi←wi+η⋅e⋅xi
- **Bias Update:** b←b+η⋅e

Geometrically, the equation w⋅x+b=0 defines a hyperplane that separates the input space into two regions.

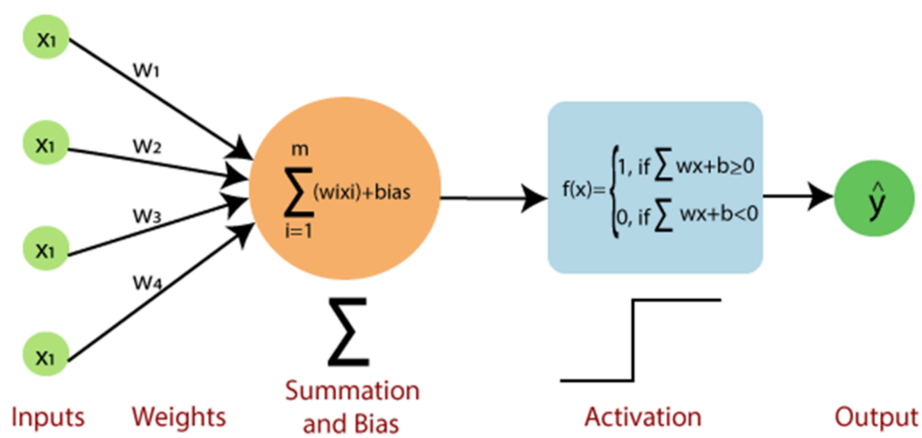
7. Limitations of the Perceptron:

- **Linear Separability:** The most significant limitation is that a single Perceptron can only classify data that is **linearly separable**. It cannot solve non-linearly separable problems, famously illustrated by the **XOR problem**.

- **Binary Output:** The classical Perceptron with a step function produces only binary outputs (0 or 1, or -1 or 1). It doesn't provide probability scores.
- **Sensitivity to Learning Rate:** If the learning rate is too high, the algorithm might overshoot the optimal weights and oscillate. If too low, convergence can be very slow.
- **Noisy Data:** Performance can degrade with noisy data or outliers.
- **Convergence on Non-Separable Data:** If the data is not linearly separable, the Perceptron learning rule will not converge; the weights will continue to change.

8. Significance & Legacy:

- **Foundation of Neural Networks:** Despite its limitations, the Perceptron was a crucial first step and forms the basic building block (neuron) for more complex, multi-layer neural networks (Multi-Layer Perceptrons or MLPs).



③ Perceptron [Artificial Neuron or Neural Network Unit]

① Input Layer ✓

② Hidden Layer ✓

③ Weights ✓

④ Activation function ✓

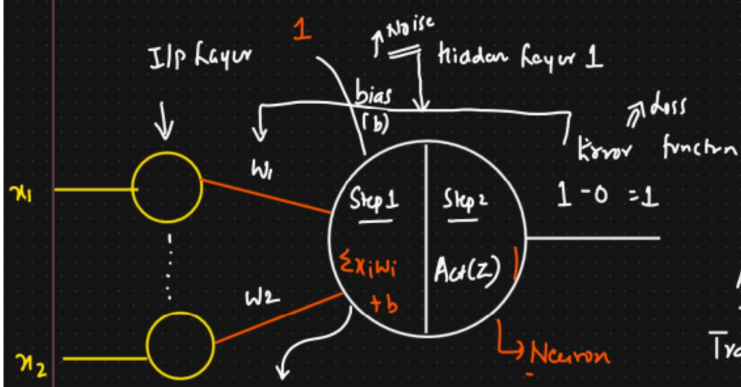
[Single Layered NN]

⇓

Binary classifier

DATASET

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



$z = w_1x_1 + w_2x_2 + b$

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

Activation function

Transform the output between 0 to 1

-1 to 1

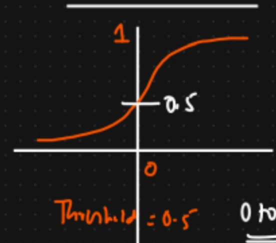
Step Function

0 to 1

Threshold val = 0

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

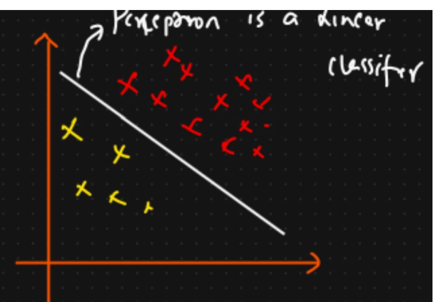
Sigmoid Function



Threshold = 0.5

0 to 1

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



Step 1

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

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

$$\Downarrow$$

$$y = mx + c$$

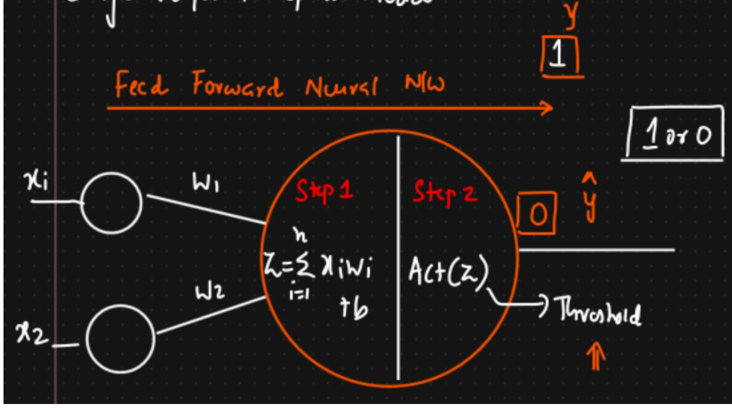
$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 \cdots + \beta_n x_n$$

Linear problem statement

Perceptron Models

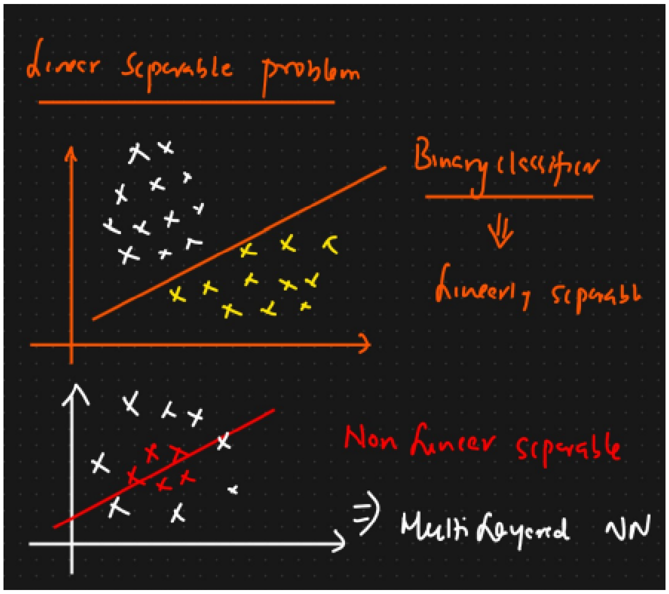
[ANN]

Single Layer Perceptron Model



Multi Layered Perceptron Model

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


Linear Seperable problem
Binary classifier
Linearly seperable
Non Linear seperable
Multi Layered NN