Day 1

Deep Learning

---

1 Introduction to Deep Learning (DL):

Deep Learning is a **subset of Machine Learning (ML)**. It focuses on algorithms that try to **mimic how the human brain works using artificial neurons**. These algorithms learn from **large datasets** using multi-layered neural networks.
Deep Learning is capable of **automatically learning features from raw data** like images, audio, text without manual feature extraction.
Applications: Face recognition, autonomous cars, medical diagnosis, chatbots.

---

2 Types of Deep Learning Models:

Common Types:

1 **Feedforward Neural Networks (FNN):** Data flows only in one direction, input to output.
2 **Convolutional Neural Networks (CNN):** Mainly used for **image-related tasks** (face detection, medical images).
3 **Recurrent Neural Networks (RNN):** Best for **sequential data** like speech, text, time-series.
4 **Generative Adversarial Networks (GANs):** Used for **creating realistic fake data** like images, videos.
5 **Autoencoders:** For **feature learning, data compression, noise reduction**.

---

3 History of Deep Learning:

Year	Event
1943	First idea of **Artificial Neuron (McCulloch-Pitts)**
1958	**Perceptron** invented by Frank Rosenblatt
1980s	**Backpropagation Algorithm** introduced for training neural networks
2012	**AlexNet (CNN)** wins ImageNet Challenge → DL becomes popular
2014+	GANs, LSTMs, ResNet gained popularity
Present	Deep Learning is everywhere (NLP, CV, Robotics, Healthcare)

□ 4□ Difference Between ML and DL:

Aspect	Machine Learning (ML)	Deep Learning (DL)
Feature Extraction	Done **manually by humans**	Model **learns automatically**
Data Requirement	Works with **small datasets**	Needs **large datasets**
Accuracy	Medium on complex tasks	High accuracy on complex tasks
Examples	Decision Trees, SVM, Linear Regression	CNN, RNN, GANs

---

□ 5□ What is Representation Learning?

Representation Learning means a model **learns useful patterns and features directly from raw data automatically**.
You don’t need to manually specify what features are important — the model discovers them itself.
Example: CNNs learning edges, shapes, and textures from images layer by layer.

□ 7□ Factors Behind the Success of Deep Learning:

1□ Availability of **large datasets** (Big Data)
2□ **Powerful hardware** (GPUs, TPUs, NPUs)
3□ **Improved algorithms** (CNN, RNN, GANs, Transformers)
4□ Open-source **libraries** (TensorFlow, PyTorch, etc.)
5□ High **investment from tech companies** (Google, Meta, Microsoft)

□ 8□ GPU, TPU, NPU – Hardware for DL & Why?

Hardware	Purpose
GPU (Graphics Processing Unit)	Highly parallel, speeds up matrix operations; essential for training DL models.
TPU (Tensor Processing Unit)	Developed by Google, specifically optimized for **TensorFlow** and deep learning workloads.
NPU (Neural Processing Unit)	Designed for **AI tasks on devices** like smartphones (efficient, low-power).

Why needed?
Deep Learning requires **millions of matrix multiplications**. These specialized hardware components perform such operations much faster than traditional CPUs.

□ 9□ Deep Learning Libraries:

Library	Purpose
TensorFlow	Google's library for DL, widely used for both research and production.
PyTorch	Developed by Facebook (Meta), favored for **research, flexibility, easy debugging**.
Keras	High-level API running on TensorFlow; easier and faster to build models.
Caffe2	Developed by Facebook; focused on **production deployment**, not research.

□ What is a Neural Network (NN)?

A **Neural Network** is a computing system **inspired by the human brain**.
Just like our brain consists of neurons connected together, a neural network is made up of **artificial neurons (also called nodes)** organized into **layers**.

□ How does it work?

1□ **Input Layer**: Takes the input features (like pixel values in an image, or words in a sentence).
2□ **Hidden Layers**: These layers do **most of the computation** through mathematical operations (matrix multiplication, activation functions like ReLU, Sigmoid).
3□ **Output Layer**: Gives the final prediction/output (e.g., is the image a cat or a dog).

Each neuron takes some **input, applies a weight, adds bias, and passes it through an activation function** to decide the output.

□ Why Neural Networks?

- They can **model complex non-linear relationships** in data.
- They are **flexible** and work for a wide variety of data: images, sound, text, tabular.
- They can automatically **learn useful features from data**.

□ Types of Neural Networks (Detailed Explanation):

1□ Feedforward Neural Network (FNN)

Simplest form of Neural Network.

- Data moves **in one direction only** (input → hidden layers → output).
- No loops or cycles.
- Mainly used for simple problems like classification and regression.

Example: Predicting house prices, spam detection.

2️⃣ Convolutional Neural Network (CNN)

Specialized for Image and Visual data.

- Uses **Convolutional layers** to detect **patterns like edges, textures, shapes** in images.
- Highly efficient for **computer vision** tasks.
- Layers like pooling, flattening, fully connected layers are used.

Applications: Face recognition, medical imaging, object detection.

3️⃣ Recurrent Neural Network (RNN)

Designed for sequential data where past information matters.

- Has **loops (recurrent connections)**, allowing output of one step to influence the next.
- Can remember previous inputs through **hidden states**.

Applications: Text generation, speech recognition, stock price prediction.

4️⃣ Generative Adversarial Network (GAN)

Used to create new (fake but realistic) data.

- Consists of **two networks fighting each other:**
    - **Generator:** Tries to create fake data.
    - **Discriminator:** Tries to detect fake vs. real.

- Improves over time as both compete.

Applications: DeepFakes, art generation, realistic synthetic data for training.

5️⃣ Radial Basis Function Network (RBFN)

Used for classification problems where data points are grouped in space.

- Uses **Radial Basis Function** as an activation function.
- Measures how far an input is from a center point in the feature space.
- Good for problems where relationships are based on distance (like clustering).

Applications: Function approximation, pattern recognition.

6️⃣ Autoencoders

Used for unsupervised learning tasks like data compression or noise reduction.

- Learn to **encode data into a compressed form (latent space)** and then reconstruct it back.
- The middle layer (bottleneck) captures the most important information.

Applications: Image denoising, anomaly detection, feature learning.

□ Summary Table (Comparison Purpose):

Type	Purpose	Example Use
FNN	Simple input-output mapping	House prices, spam detection
CNN	Image pattern recognition	Face detection, MRI scans
RNN	Sequential data learning	Text, speech, time-series
GAN	Data generation	Deepfakes, art, avatars
RBFN	Distance-based classification	Pattern recognition
Autoencoder	Data compression, denoising	Noise reduction, anomaly detection

---

What is a Perceptron?

A **Perceptron** is the **simplest type of artificial neural network** and was the **foundation of deep learning models today**. It is inspired by the way biological neurons work.

How Perceptron Works:

1 **Takes multiple inputs (x_1, x_2, ..., xn).**
2 **Each input has a weight (w_1, w_2, ..., wn) attached.**
3 It calculates the **weighted sum**:
➔ $Z = (x_1 * w_1) + (x_2 * w_2) + ... + (xn * wn) + bias$
4 Passes this sum through an **activation function** (usually Step function for Perceptron).
5 **Gives output as either 0 or 1** (Binary classification).

If the weighted sum > threshold: output is 1
Else: output is 0

Mathematical Representation:

Output={1if ∑(wi·xi)+b>00otherwise\text{Output} = \begin{cases} 1 & \text{if} \ \sum (w_i \cdot x_i) + b > 0 \\ 0 & \text{otherwise} \end{cases}

Types of Perceptron:

1 Single-Layer Perceptron (SLP)

- Only one layer between input and output.
- Only capable of solving **linearly separable problems** (straight-line boundary between classes).
- Example: OR, AND logic gates.

---

2 Multi-Layer Perceptron (MLP)

- Contains **one or more hidden layers** between input and output.
- Can solve **non-linear problems** (complex decision boundaries).
- Uses **activation functions like ReLU, Sigmoid** in hidden layers.
- Trained using **backpropagation**.

History of Perceptron:

Year	Event
1958	Frank Rosenblatt invented the **Perceptron**.
1969	Marvin Minsky & Seymour Papert showed that Single-Layer Perceptron can't solve **XOR problem**. This led to **AI Winter** (lack of progress in AI for years).
1986	**Multi-Layer Perceptron (MLP)** + **Backpropagation** popularized by Rumelhart, Hinton, Williams, bringing AI research back to life.
Today	Perceptron concept evolved into **Deep Learning** networks (CNN, RNN, GAN, etc.).

□ Neuron vs. Perceptron (Difference):

Aspect	Neuron	Perceptron
Concept	Biological neuron (brain-inspired)	Mathematical model of a neuron
Functionality	Sends signals chemically/electrically	Computes weighted sum + activation
Output	Complex continuous outputs	Binary output (0 or 1)
Learning	Learns through connections	Learns by adjusting weights via error
Role in AI	Inspiration for NN design	First simple model of Artificial Neuron

Biological Neuron (Brain):

- Dendrites: Inputs
- Axon: Output
- Synapse: Weight

Artificial Perceptron (AI):

- Inputs (x_1, x_2,...): Inputs
- Weights (w_1, w_2,...): Strength of connection
- Activation: Decision
- Output: Result (0/1)

□ Quick Summary (Easy to Remember):

Neuron	Biological Inspiration. Our Brain's Cells.
Perceptron	First mathematical step to mimic neurons in AI.

□ How Perceptron Works —

Structure of a Perceptron:

A perceptron has these key components:

- **Inputs (x_1, x_2, ..., xn)**
- **Weights (w_1, w_2, ..., wn)** — controls the importance of each input.
- **Bias (b)** — helps shift the decision boundary.
- **Weighted Sum (Z)**
- **Activation Function** (usually Step Function for binary output)

Working Steps of Perceptron:

□ Step 1: Inputs & Weights

Suppose inputs are:

x1=1,x2=0x_1 = 1, \quad x_2 = 0

And weights are:

w1=0.5,w2=0.5w_1 = 0.5, \quad w_2 = 0.5

Bias:

b=−0.7b = -0.7

□ Step 2: Calculate Weighted Sum (Z)

Z=(x1·w1)+(x2·w2)+bZ = (x_1 \cdot w_1) + (x_2 \cdot w_2) + b Z=(1·0.5)+(0·0.5)+(−0.7)=0.5−0.7=−0.2Z = (1 \cdot 0.5) + (0 \cdot 0.5) + (-0.7) = 0.5 - 0.7 = -0.2

□ Step 3: Apply Activation Function (Step Function)

- If `Z > 0`, output is **1**
- If `Z <= 0`, output is **0**

Here:

$Z=-0.2 \implies \text{Output}=0$Z = -0.2 \implies \text{Output} = 0

Visualization:

```
Inputs ---> Weighted Sum ---> Activation ---> Output
x1, x2     w1, w2, bias         Step          0 or 1
```

□ Why It Works?

- The perceptron is trying to find a **line (in 2D), plane (in 3D), or hyperplane (higher dimensions)** to **separate classes**.
- The weights and bias define this boundary.
- Activation function decides **which side of the boundary** the input is on.

□ Perceptron Example Use Case:

AND Gate

x_1	x_2	Output
0	0	0
0	1	0
1	0	0
1	1	1

A perceptron can easily learn this because the **AND gate is linearly separable**.

□ What Happens During Learning?

1□ Start with **random weights and bias**.
2□ Give input, calculate output.
3□ **Compare output with actual answer (label).**
4□ **Adjust weights using error:**

wnew=wold+(learning rate)×(error)×(input)w_{\text{new}} = w_{\text{old}} + (learning\ rate) \times (error) \times (input)

Repeat until outputs are correct.

□ Geometric Intuition of Perceptron

The **Perceptron Algorithm is fundamentally geometric** — it tries to find a **straight line (in 2D)**, **a plane (in 3D)**, or a **hyperplane (in higher dimensions)** to **separate two classes of data points.**

□ Geometric View in 2D Space:

Suppose you have two features:

- x_1 (horizontal axis)
- x_2 (vertical axis)

Your perceptron will try to find the best **straight line** that splits the data into:

- **Class 0 on one side**
- **Class 1 on the other side**

Equation of Line (in 2D):

w1·x1+w2·x2+b=0w_1 \cdot x_1 + w_2 \cdot x_2 + b = 0

- This is just like the equation of a straight line.
- The **weights (w_1, w_2)** determine the **slope and direction** of this line.
- The **bias (b)** shifts the line up, down, left, or right.

□ How Perceptron Adjusts This Line:

1□ Initially, the line may not separate the classes well (random weights).
2□ When the perceptron makes mistakes, it **updates weights and bias**.
3□ These updates **rotate or shift the line** slightly.
4□ After many iterations, the line becomes good at **splitting the classes**.

Intuition:

- Every **data point "pushes" the line** when misclassified.
- Over time, the line **aligns itself** so it correctly divides the two groups.

Visualization:

```
x₂  |                         ● Class 1
    |                        /
    |                      /
    |  Class 0 ●  ______/__________  (Decision Boundary)
    |                  /
    |                /
x₁  |______________/
```

Everything **above the line** might output `1` (Class 1).
Everything **below the line** might output `0` (Class 0).

Higher Dimensions:

Dimensions	**Decision Boundary Type**
2D	Line
3D	Plane
4D+	Hyperplane

Limitation:

Perceptron works **only if the data is linearly separable** — meaning you can draw a straight line (or plane) to separate the classes.

Example it cannot solve: XOR Problem (no straight line can separate it).

□ Summary:

Perceptron is like drawing a line (or plane) that splits points into two groups.

- Weights = direction of the line
- Bias = position of the line
- Learning = shifting/rotating the line till it separates correctly.
