Savitribai Phule Pune University
Final Year of Information Technology (2019 Course)
(With effect from Academic Year 2022-23)

Semester VII

Course Code	Course Name	Teaching Scheme(Hours/week)			Examination Scheme and Marks						Credit Scheme			
		Lecture	Practical	Tutorial	Mid-Sem	End-Sem	Termwork	Practical	Oral	Total	Lecture	Practical	Tutorial	Total
414441	Information and Storage Retrieval	03	-	-	30	70	-	-	-	100	3	-	-	3
414442	Software Project Management	03	-	-	30	70	-	-	-	100	3	-	-	3
414443	Deep Learning	03	-	-	30	70	-	-	-	100	3	-	-	3
414444	Elective III	03	-	-	30	70	-	-	-	100	3	-	-	3
414445	Elective IV	03	-	-	30	70	-	-	-	100	3	-	-	3
414446	Lab Practice III	-	04	-	-	-	25	-	25	50	-	2	-	2
414447	Lab Practice IV	-	02	-	-	-	25	25	-	50	-	1	-	1
414448	Project Stage-I	-	-	02	-	-	50	-	-	50	-	-	2	2
414449	Audit Course7													
										Total Credit	15	03	02	20
	Total	15	06	02	150	350	100	25	25	650	15	03	02	20

Elective III:

- Mobile Computing
- High Performance Computing
- Multimedia Technology
- Smart Computing

Elective IV:

- Bioinformatics
- Introduction to DevOps
- Computer Vision
- Wireless Communications

Lab Practice-III:

It is based on subjects:

- Information and Storage Retrieval

Lab Practice-IV:

It is based on subjects:

- Deep Learning

B.E. (INFORMATION TECHNOLOGY)

SEMESTER-VII

Time– 2.00 p.m. to 3.00 p.m.

Day& Date	SUBJECT (2019 Course)	SUBJECT CODE
Tuesday 19/08/2025	Information and Storage Retrieval	414441
Wednesday 20/08/2025	Software Project Management	414442
Thursday 21/08/2025	Deep Learning	414443
Friday 22/08/2025	(ELECTIVE-III) Mobile Computing	414444A
	(ELECTIVE-III) High Performance Computing	414444B
	(ELECTIVE-III) Multimedia Technology	414444C
	(ELECTIVE-III) Smart Computing	414444D
Saturday 23/08/2025	(ELECTIVE-IV)Bioinformatics	414445A
	(ELECTIVE-IV) Introduction to DevOps	414445B
	(ELECTIVE-IV) Computer Vision	414445C
	(ELECTIVE-IV) Wireless Communications	414445D

Q1) a) Draw and explain the architecture of Multilayered Feedforward Neural network. **[5]**

b) What is the need of Regularization? Explain Dropout Regularization. **[5]**

c) Explain the concept of gradient based Learning. **[5]**

OR

Q2) a) What is the problem of vanishing Gradient? Describe various solutions to this problem. **[7]**

b) Explain the working of an Artificial neuron. Also explain the activation functions ReLU and LReLU. **[8]**

Q1) a) Explain following terms related to multi-layer feed-forward networks. **[5]**

i) Biases

ii) Activation functions

b) Explain loss function for regression operation. **[5]**

c) Define and explain the significance of learning rate of a model? **[5]**

OR

Q2) a) Explain loss function for classification operation. **[5]**

b) Differentiate among RELU, LRELU and ERELU. **[5]**

c) What is regularization? Explain the need for regularization. **[5]**

Q1) a) Define Deep Learning with suitable Architecture. **[5]**

b) What is hyper parameter? Describe categories of hyper parameters? **[5]**

c) Differentiate between Single layer feed forward Neural Network and Multi Layer feed Forward Neural Network . **[5]**

OR

Q2) a) Explain following Activation functions. **[7]**

i) ReLU

ii) LReLU

iii) EReLU

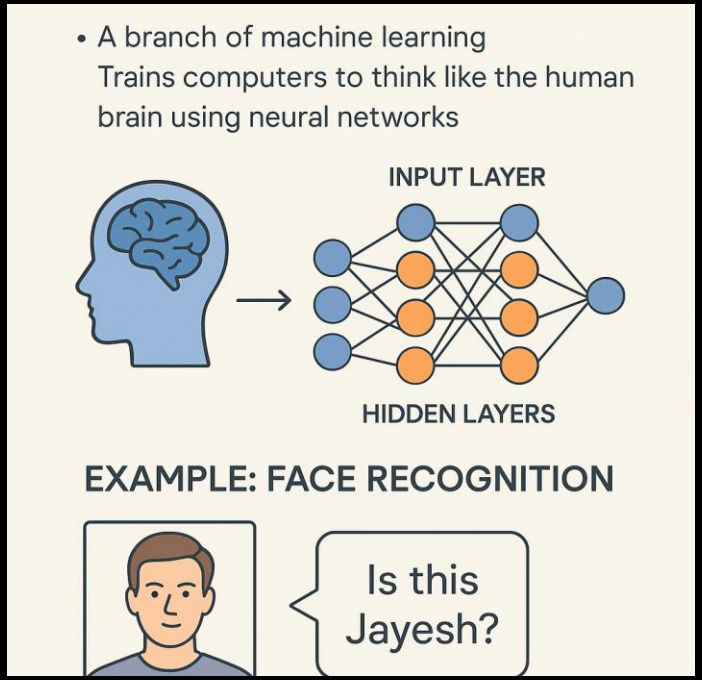
b) What is the problem of vanishing gradient? Explain the various solutions to avoid it? **[8]**

COURSE CONTENTS		
Unit I	**Fundamentals of Deep Learning**	**(06 hrs)**
What is Deep Learning?, Multilayer Perceptron ,Feed forward neural, Back propagation, Gradient descent, Vanishing gradient problem, **Activation Functions:** RELU, LRELU, ERELU, Optimization Algorithms, **Hyper parameters:** Layer size, Magnitude (momentum, learning rate),Regularization (dropout, drop connect, L1, L2)		
Mapping of Course Outcomes for Unit I	**CO1**	
Unit II	**Convolutional Neural Network:**	**(06 hrs)**
Introduction to CNN, Convolution Operation, Parameter Sharing, Equivariant Representation, Pooling, Variants of the Basic Convolution Function, The basic Architecture of CNN, Popular CNN Architecture – AlexNet.		

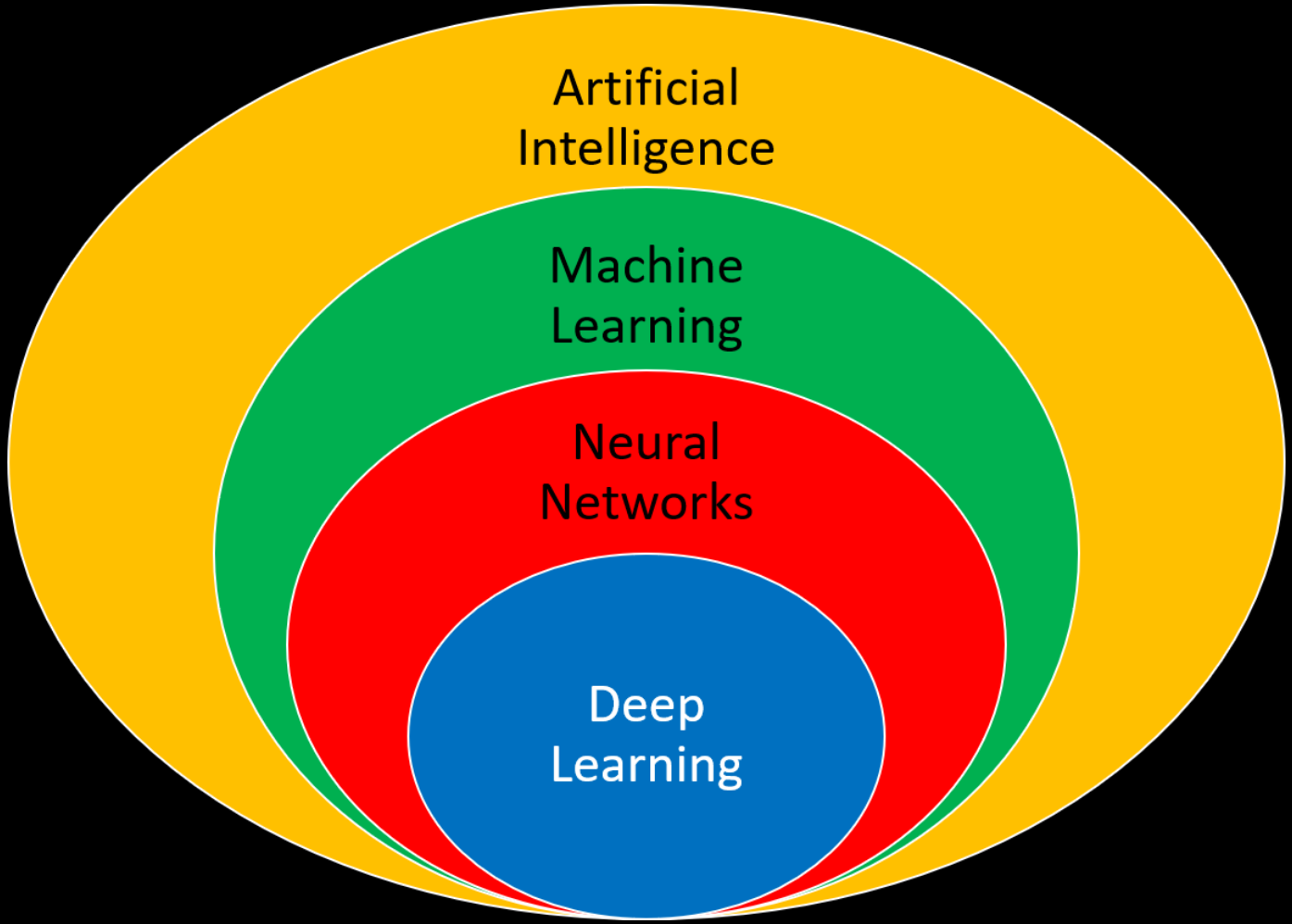
Unit I – Fundamentals of Deep Learning (06 hrs)

- **What is Deep Learning?**
- **Multilayer Perceptron**
- **Feedforward Neural Network**
- **Backpropagation**
- **Gradient Descent**
- **Vanishing Gradient Problem**
- **Activation Functions**:
  - RELU
  - LRELU (Leaky RELU)
  - ERELU
- **Optimization Algorithms**
- **Hyperparameters**:
  - Layer size
  - Magnitude (momentum, learning rate)
- **Regularization**:
  - Dropout
  - Drop connect
  - L1 Regularization
  - L2 Regularization

🔍 What is Deep Learning?



Deep Learning ek branch hai Machine Learning ka, jisme hum computers ko human brain jaise sochne ke liye train karte hai – lekin yeh neural networks ka use karta hai.
Yeh large data se patterns aur features automatically seekh leta hai bina manually programming ke.


Artificial
Intelligence
Machine
Learning
Neural
Networks
Deep
Learning

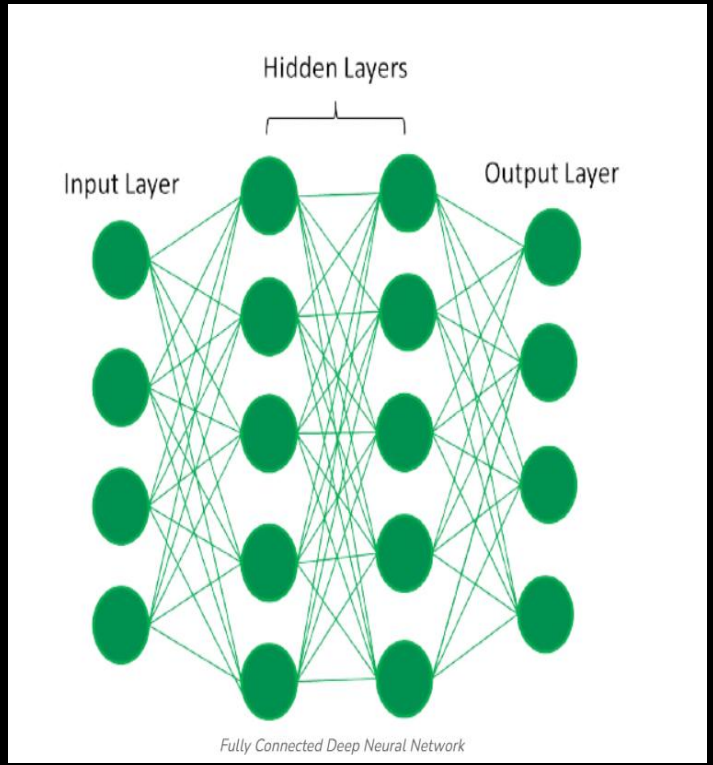
Real-Life Examples of Deep Learning:

1. **Face Recognition** – Facebook kisi photo me face detect karta hai.
2. **Voice Assistants** – Alexa, Siri aapka voice input samajh kar jawab dete hain.
3. **Recommendation System** – Netflix ya Amazon par personalized suggestions.
4. **Self-Driving Cars** – Road, sign, pedestrian detect karna.
5. **Medical Diagnosis** – MRI/CT scans se disease prediction.

🧠 Why the word "Deep"?

"Deep" ka matlab hai **multiple hidden layers** in the network.

•Traditional ML = 1–2 layers
•Deep Learning = 10+ layers bhi ho sakti hain!



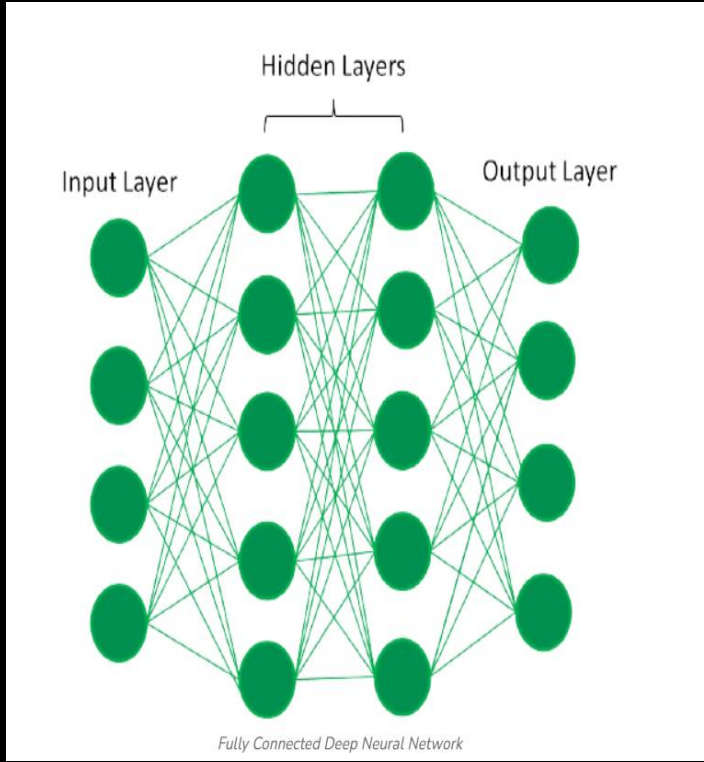
📍 Difference: Machine Learning vs Deep Learning

🔸 Point	🤖 Machine Learning (ML)	🧠 Deep Learning (DL)
1. Feature Selection	Manually features nikalne padte hain. 🧪 Example: Email spam ke liye "spam words" count.	Features automatically nikalta hai. 🧪 Example: CNN khud image se shape/edge detect karta hai.
2. Data Need	Kam data me bhi kaam karta hai. 🧪 Example: Student marks prediction (small dataset).	Bahut zyada data chahiye hota hai. 🧪 Example: Self-driving car ke liye lakhs of images.
3. Accuracy	Small data pe sahi result deta hai.	Big data pe high accuracy deta hai.
4. Training Time	Fast training, CPU se ho jaata hai. 🧪 Example: KNN/Decision Tree train in mins.	Slow training, GPU chahiye. 🧪 Example: CNN training can take hours.
5. Explainability	Easy to understand and explain. 🧪 Example: Decision Tree ke rules dikh jaate hain.	Kaise kaam karta hai samajhna mushkil (Black Box). 🧪 Example: CNN decisions are hard to trace.
6. Use Cases	Tabular/simple data ke liye. 🧪 Example: Loan approval, stock prediction.	Image, video, audio, NLP tasks ke liye. 🧪 Example: Face unlock, speech recognition.

Q1) a) Define Deep Learning with suitable Architecture. [5]

🔁 How Deep Learning Works :

1.Input data (e.g., image)
2.Goes through **Input Layer**
3.Passes multiple **Hidden Layers** (each layer extracts some features)
4.Reaches **Output Layer** (e.g., dog ya cat detect kiya?)
5.Result milta hai – prediction.



Fully Connected Deep Neural Network

"Aapko Netflix par horror movies pasand hain – Deep Learning aapke pattern ko samajh kar suggestions deta hai."

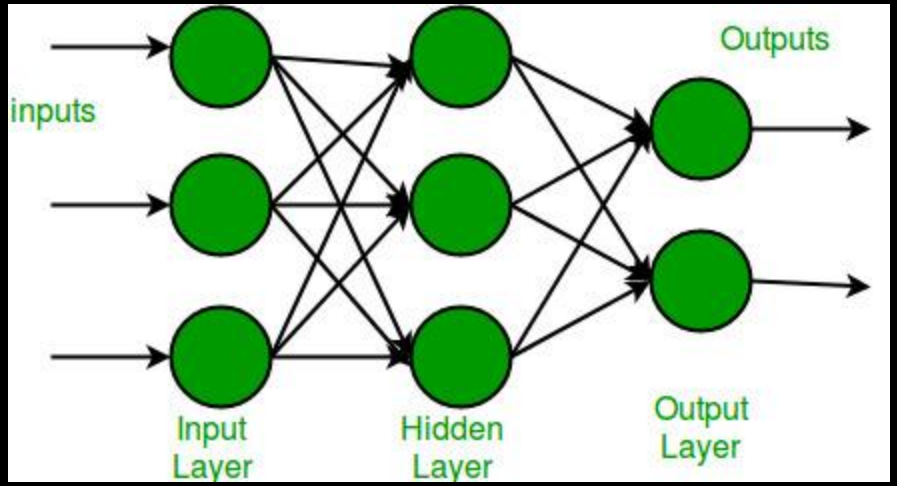
• 🧠 Multilayer Perceptron (MLP)

📌 What is Multilayer Perceptron?

Multilayer Perceptron (MLP) is a type of **Artificial Neural Network** that contains **multiple layers** of neurons — mainly:

1. **Input Layer**
2. **Hidden Layers** (1 or more)
3. **Output Layer**

It is a **supervised learning model** used for both **classification** and **regression** problems.



MLP mein 3 layers hoti hain:
1.Input Layer – Data input hota hai (x1, x2, x3...)
2.Hidden Layers – 1 ya usse zyada layers hoti hain jo processing karti hain
3.Output Layer – Final prediction output milta hai

⚙️ **Kaise kaam karta hai? (Working of MLP)**

1.Input aata hai – jaise image ke pixels ya student ke marks

2.Hidden layer:

- Har neuron weighted sum karta hai: $z = w1*x1 + w2*x2 + ... + b$
- Fir activation function lagta hai: $a = f(z)$ (ReLU ya sigmoid)

3.Output layer – result nikalta hai (e.g., 0 ya 1 for classification)

Mathematical Representation

a single neuron:

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

$$a = f(z) \quad \text{(Activation function)}$$

🧪 **Example: Digit Recognition**

Input: Image of number "5"

MLP hidden layers: curves, edges detect karta hai

Output: Predicts "5" as output

MLP ke Advantages:

- Non-linear problems solve karta hai
- Complex patterns ko samajh sakta hai
- Versatile hai – regression & classification dono mein kaam aata hai

💡 **MLP mein Activation Functions:**

- **ReLU**: Fast, vanishing gradient se bachata hai
- **Sigmoid**: 0 to 1 output deta hai
- **Tanh**: -1 to 1 output deta hai
- **Softmax**: Multiple class prediction ke liye

🔁 **Training Process:**

1. **Forward Propagation** – Input se output tak nikalna
2. **Loss Function** – Output ka error calculate karna
3. **Backpropagation** – Error ko reverse mein spread karna
4. **Weights update** – Using Gradient Descent

🔐 MLP mein Common Terms:

Term	Meaning
Neuron	Ek unit jo input leke output nikalta hai
Weight (w)	Har input ka importance batata hai
Bias (b)	Ek extra value to shift the output
Activation fn	Decide karta hai neuron ka output kya hoga
Loss Function	Kitna galat predict kiya, uska measure
Learning Rate	Har step mein kitna weight change hoga

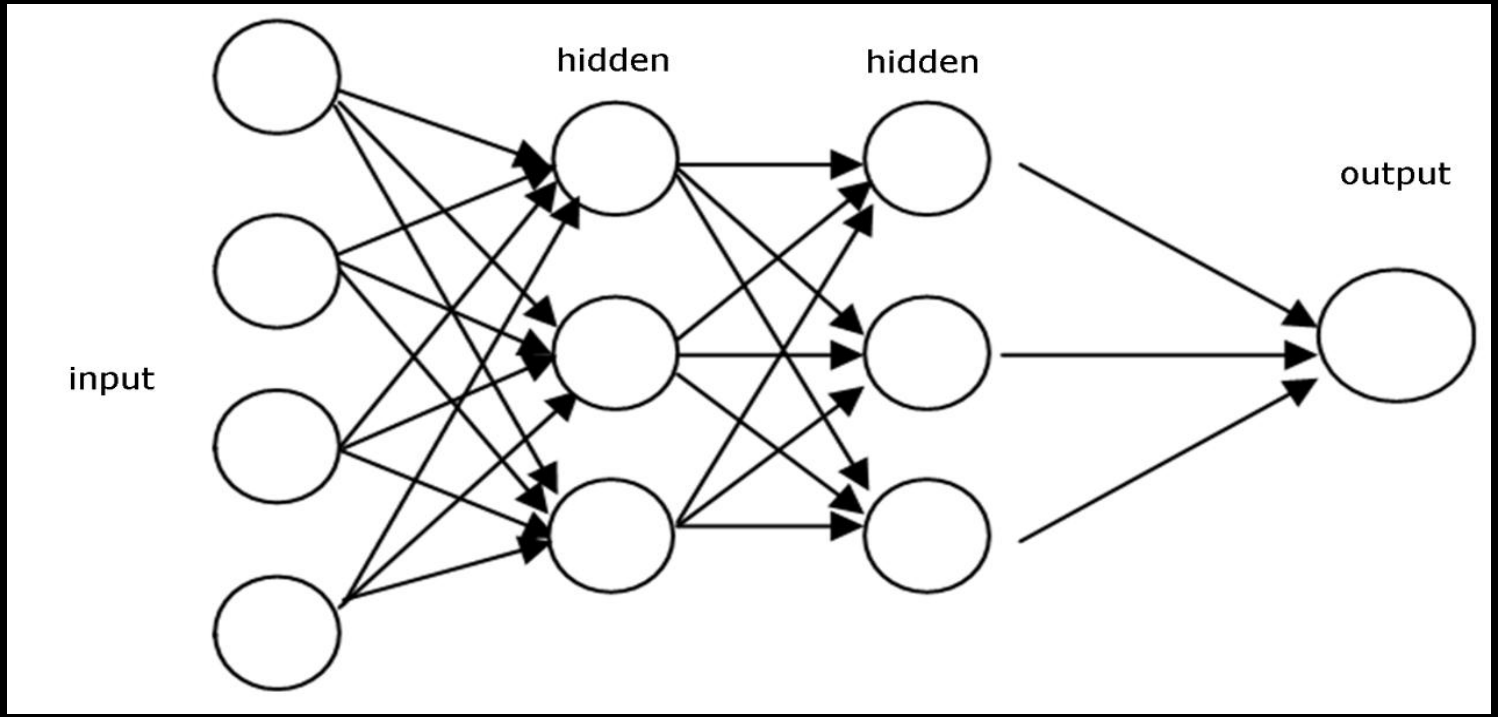
Feature	Single Layer	Multilayer Perceptron (MLP)
Layers	Sirf 1	1+ hidden layers
Problem Solve	Linear problems	Complex non-linear problems
Accuracy	Limited	Better on large datasets
Example	Basic spam filter	Digit recognition, face detection

🧠 Feedforward Neural Network (FFNN)

🔹 Definition

Feedforward Neural Network ek aisi neural network hoti hai jisme data **sirf ek direction mein flow karta hai – input → output**.

Isme **koi loop nahi hota**, aur **backward connection** bhi nahi hota.
Yeh Deep Learning ka **sabse basic architecture** hai, jisme neurons layers mein organized hote hain.



Input → Hidden Layer 1 → Hidden Layer 2 → Output
[x1,x2,x3,x4] → [h1,h2,h3] → [y]

🧩 FFNN ki Layers

1.Input Layer – Jo input data leta hai (jaise x1, x2, x3...)

2.Hidden Layer(s) – Data ko process karta hai (1 ya zyada ho sakte hain)

3.Output Layer – Final result nikalta hai

🔁 **Data hamesha left to right flow karta hai**, koi feedback loop nahi hota.

⚙️ Working (Step by Step)

1.Input layer: Real-world data enter karta hai (e.g., image pixels, numbers, etc.)

2.Hidden layers:

1. Har neuron input ko weighted sum karta hai:
2. $$z = w_1x_1 + w_2x_2 + ... + b$$
3. Activation function apply hoti hai (ReLU, Sigmoid, etc.)

3.Output layer: Final output generate hota hai – jaise classification result.

🔬 Mathematical View:

Ek neuron ka output:

$$z = \sum (w_i \cdot x_i) + b \quad \text{and} \quad a = f(z)$$

Feature	Description (Hinglish)
Data Flow	Sirf input → output direction mein hota hai
Fully Connected	Har neuron agle layer ke sabhi neurons se connected hota hai
No Loop	Feedback ya loop bilkul nahi hota
Training	Backpropagation + Gradient Descent se hoti hai
Activation Function	Non-linearity ke liye use hoti hai (ReLU, Sigmoid)

a) Draw and explain the architecture of Multilayered Feedforward Neural network. **[5]**

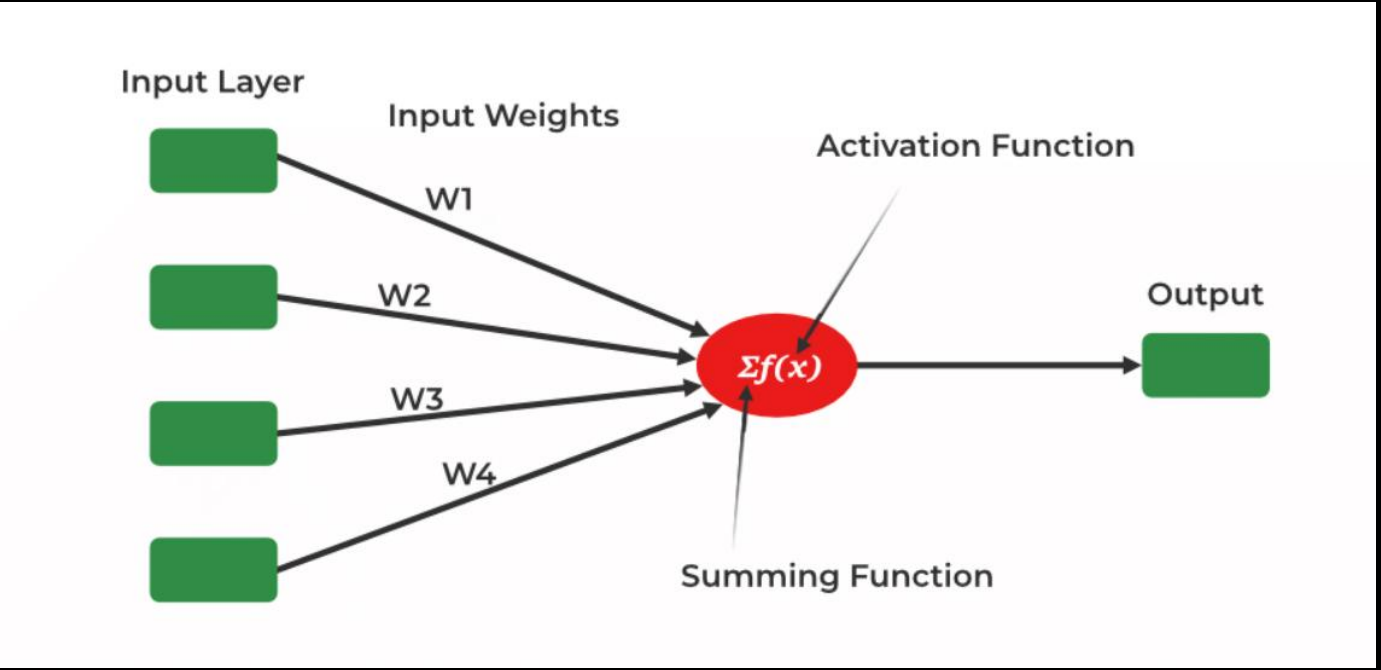
a) Explain following terms related to multi-layer feed-forward networks. **[5]**

i) Biases

ii) Activation functions

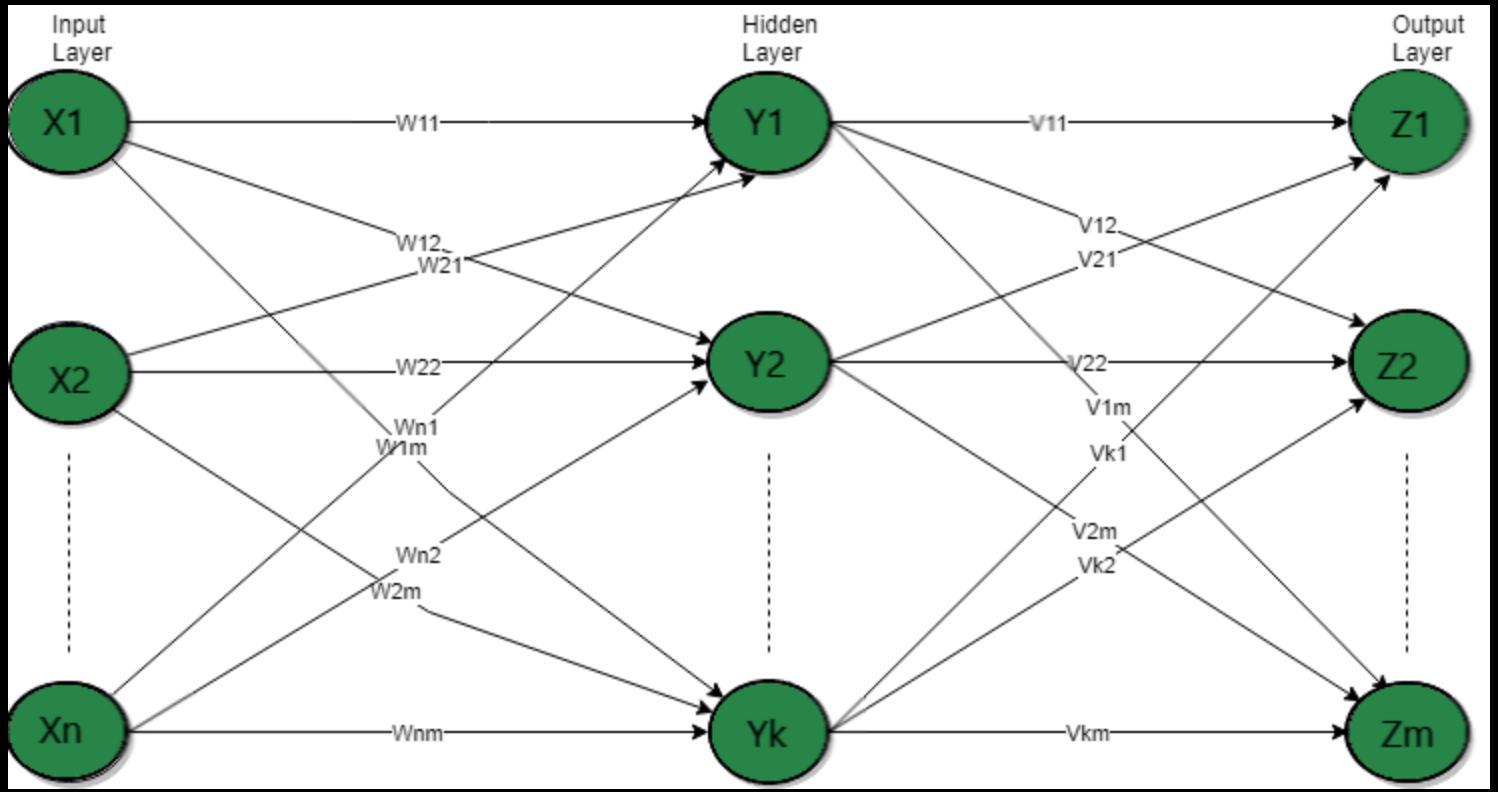
c) Differentiate between Single layer feed forward Neural Network and Multi Layer feed Forward Neural Network . **[5]**

🟦 1. Single-Layer Feedforward Neural Network (Perceptron)



🔢 Step	⚙️ Process	📘 Explanation (Hinglish)
1️⃣	Input Layer	Inputs aate hain: x_1, x_2, x_3, x_4 – yeh real-world values hote hain (e.g., marks, pixels)
2️⃣	Weights Assignment	Har input ke saath ek weight w_1, w_2, w_3, w_4 assign hota hai jo importance define karta hai
3️⃣	Summing Function	Weighted sum nikalte hain: $z = (w_1x_1 + w_2x_2 + \ldots + b)$
4️⃣	Activation Function Apply	Sum z ko activation function f(z) mein pass karte hain — jaise ReLU, Sigmoid
5️⃣	Output Generation	Activation ke baad final output $a = f(z)$ generate hota hai
✅	Prediction / Decision	Output use hota hai classification ya decision ke liye — jaise spam/not spam, pass/fail

📊 Multilayer Feedforward Neural Network (MLP)



🔢 Step	📌 Component	🧠 Explanation (Hinglish)
1️⃣	Input Layer	Neurons: X1, X2, ..., Xn – yeh layer real-world data input karti hai (pixels, scores, etc.)
2️⃣	Input to Hidden Weights (W)	Har input neuron (Xi) hidden neurons (Yj) se fully connected hota hai via weights Wij
3️⃣	Hidden Layer	Neurons: Y1, Y2, ..., Yk – yeh inputs ka weighted sum + bias calculate karke activation apply karta hai
4️⃣	Activation Function (Hidden Layer)	Activation functions jaise ReLU, Sigmoid, Tanh apply hote hain to learn complex features
5️⃣	Hidden to Output Weights (V)	Hidden neurons se output neurons tak connections via weights Vjk (e.g., V11 = Y1 → Z1)
6️⃣	Output Layer	Neurons: Z1, Z2, ..., Zm – final prediction ya classification deta hai
🔁	Flow Direction	Data sirf ek direction mein flow karta hai: Input → Hidden → Output (feedforward nature)

🔢 Point	🧩 Single-Layer Feedforward Neural Network	🧠 Multilayer Feedforward Neural Network (MLP)
1️⃣ Layers	Sirf 1 layer hoti hai (Input → Output)	Multiple layers hoti hain (Input → Hidden(s) → Output)
2️⃣ Hidden Layer	❌ Hidden layer nahi hota	✅ Ek ya zyada hidden layers hote hain
3️⃣ Complexity	Simple problems solve karta hai	Complex patterns aur relationships ko learn karta hai
4️⃣ Learning Capability	Sirf linear data learn karta hai	Non-linear data bhi learn kar sakta hai
5️⃣ Use Cases	AND/OR gate, binary classification	Image recognition, NLP, handwritten digit prediction, spam detection
6️⃣ Accuracy	Kam accuracy in complex tasks	High accuracy in complex tasks
7️⃣ Activation	Activation sirf output layer mein lagta hai	Har hidden + output layer mein activation lagta hai
8️⃣ Example	Spam/Not spam based on 1–2 features	Digit recognition from image using multiple features

a) Explain following terms related to multi-layer feed-forward networks.[5]

i) Biases

ii) Activation functions

i) Biases:

- Neural network ka har neuron ek *weighted sum* calculate karta hai:

$$z = (w_1 \times x_1) + (w_2 \times x_2) + \cdots + b$$

  Yaha b hi **bias** hai.
- Bias ek extra adjustable parameter hai jo output ko shift karne ka kaam karta hai, chahe input zero hi kyu na ho.
- Ye model ko *flexibility* deta hai, jisse neuron decision boundaries ko adjust kar sake aur better learning ho.
- Example: Agar ek line $y = mx + c$ hai, to c bias ki tarah kaam karta hai jo line ko upar–neeche shift karta hai.

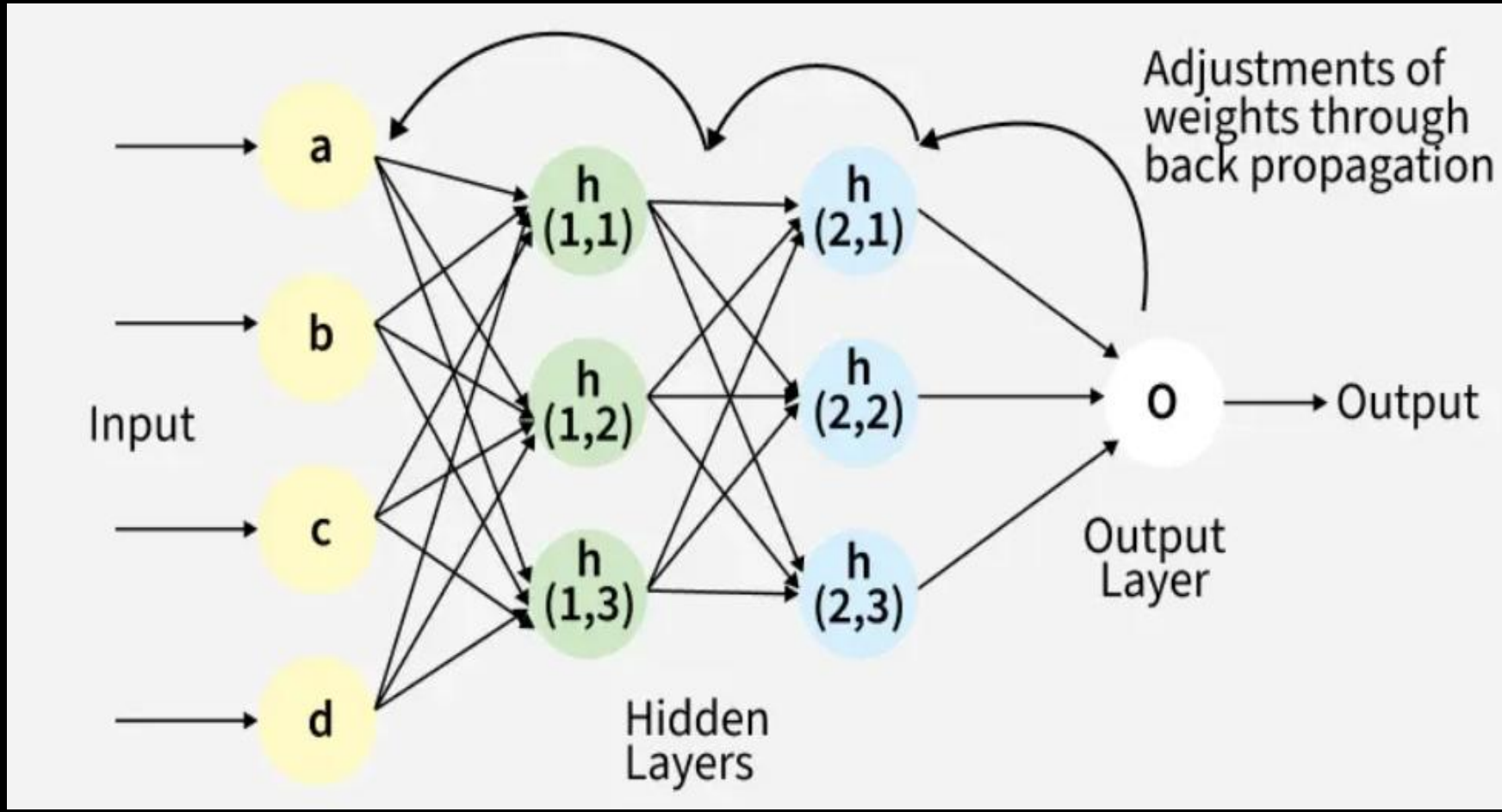
ii) Activation Functions:

- Activation function neuron ke calculated value z (weighted sum + bias) ko process karke output deta hai.
- Iska main kaam: **non-linearity introduce karna**, taaki network simple linear relation se aage badh kar complex patterns seekh sake.
- Without activation function, pura network ek simple linear model ban jaata, chahe layers kitni bhi ho.
- Common activation functions:
  - **Sigmoid**: Output 0 se 1 ke beech deta hai (useful for probabilities).
  - **ReLU (Rectified Linear Unit)**: Agar input positive hai to wahi return karta hai, warna 0. Fast aur widely used.
  - **Tanh**: Output -1 se 1 ke beech deta hai, zero-centered hota hai.

•Backpropagation

Definition:

Backpropagation ek algorithm hai jo *neural networks* ko train karne ke liye use hota hai. Isme model ke prediction aur actual output ke beech ka difference (error) calculate karke, weights aur biases ko adjust kiya jaata hai taaki error kam ho.


Adjustments of weights through back propagation
a
b
c
d
Input
h (1,1)
h (1,2)
h (1,3)
h (2,1)
h (2,2)
h (2,3)
Hidden Layers
O
Output
Output Layer

Step	Explanation	Diagram Reference
1. Input Layer	Input neurons a, b, c, d network ko data dete hain.	Yellow circles: a, b, c, d
2. Forward Pass	Data input layer se hidden layers tak flow karta hai. Har neuron weighted sum + bias ke baad activation lagata hai.	Arrows from input to h(1,x) and h(2,x)
3. Output Generation	Last layer ka neuron O final output deta hai (model ka prediction).	White circle: O (Output)
4. Error Calculation	Actual output aur predicted output ke beech ka error calculate hota hai using loss function.	Happens after O is generated (not shown)
5. Backpropagation	Error ko reverse direction me propagate kiya jaata hai. Har neuron ka error contribution calculate hota hai.	Back arrows from O to hidden layers
6. Weight Update	Gradient descent ke through har connection ka weight adjust hota hai taaki next time error kam ho.	Label: "Adjustments of weights..."

✍️Summary:

- Neural network pehle **forward pass** karta hai → output deta hai.
- Fir **error** calculate hota hai between predicted vs actual output.
- Error ko reverse me **backpropagate** kiya jaata hai using chain rule.
- **Weights & biases adjust** hote hain to reduce error.
- Ye process bar-bar repeat hota hai till network learns properly.

📉 Gradient Descent

Gradient Descent ek optimization algorithm hai jo machine learning & neural networks me use hota hai error (loss) ko kam karne ke liye by adjusting the weights and biases of the model.

Mathematical Formula:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{dJ}{dw}$$

η = learning rate (step size)

$\frac{dJ}{dw}$ = gradient of the loss w.r.t. weight

🔍 Kaise Kaam Karta Hai

Step	Explanation (Hinglish)
1. Initialization	Random weights choose kiye jaate hain.
2. Prediction	Forward pass se output nikalta hai.
3. Error Calculation	Actual output se difference (error) nikalte hain using loss function.
4. Gradient Calculation	Loss function ka derivative (slope) nikalte hain — ye batata hai weight ka error me kitna role hai.
5. Weight Update	Weight ko update karte hain using formula
6. Repeat	Ye process bar-bar repeat hota hai until loss minimum ho jaaye.

📌 **Gradient Descent Problem :**

"Given a cost function:
$J(w) = (w - 3)^2$

Use Gradient Descent to find the value of **w** that minimizes the cost function.
Use learning rate $\eta=0.1$, and initial weight $w=0$. Perform 3 iterations."

🧮 Step-by-Step Solution:

🔹 Step 1: Find derivative (gradient) of cost function

Cost function:

$$J(w) = (w - 3)^2$$

Gradient (derivative):

$$\frac{dJ}{dw} = 2(w - 3)$$

🔹 Step 2: Gradient Descent Formula

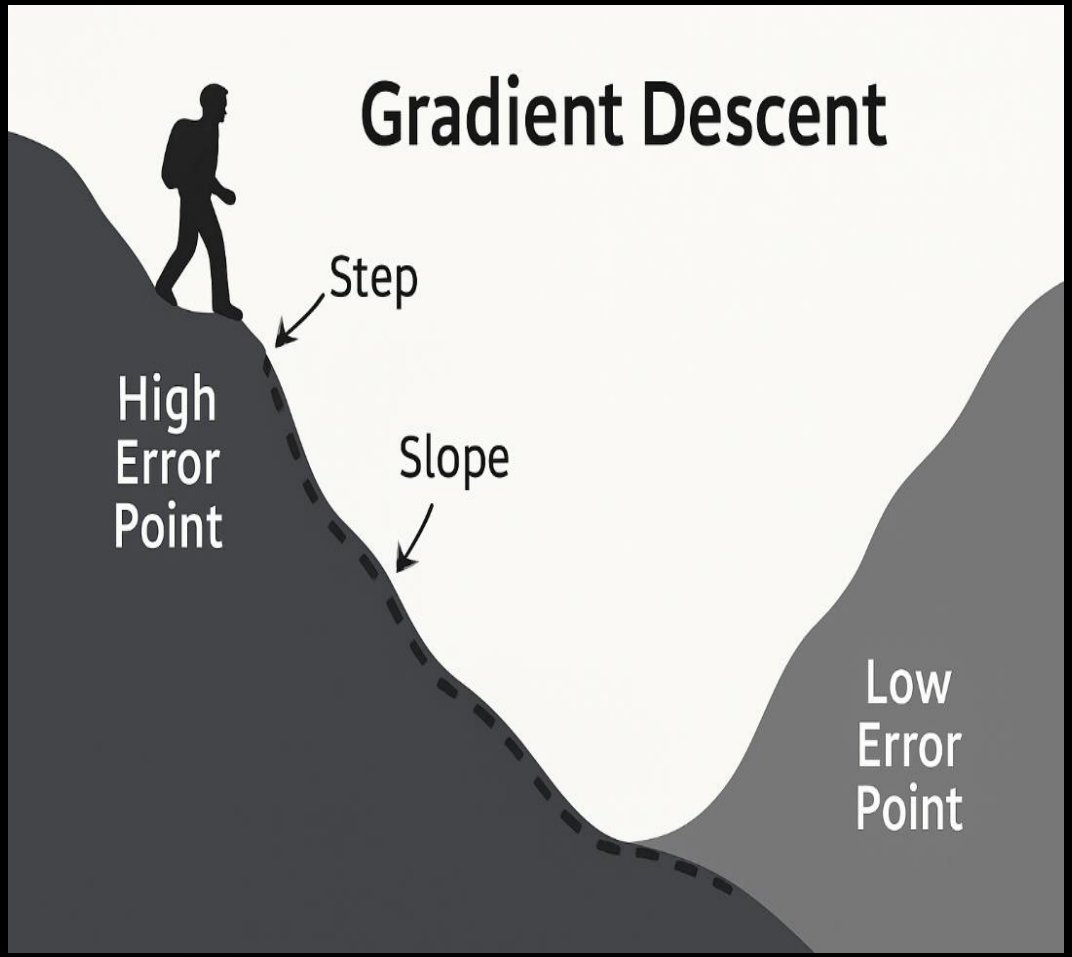
$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{dJ}{dw}$$

Step 3: Iterations

Iteration	w (old)	Gradient $2(w-3)$	Update Rule	w (new)
0	0	-6	$0 - 0.1 \times (-6) = 0 + 0.6$	0.6
1	0.6	-4.8	$0.6 + 0.48 = 1.08$	1.08
2	1.08	-3.84	$1.08 + 0.384 = 1.464$	1.464

After 3 iterations, weight **w** has moved closer to **3** (which is the minimum of the function).

👉 **Gradient Descent** slowly adjusts weights to minimize the error.



Socho tum ek andhere pahad pe khade ho (high error point), aur neeche valley me jaana hai (low error point). Tum har step me slope dekh ke decide karte ho kaha kadam rakhna hai. Agar kadam bada rakha (learning rate zyada), to gir jaoge. Agar chhota rakha, to dheere-dheere sahi jagah pahuchoge.
Yehi hai **Gradient Descent** ka concept.

🎯 Goal of Gradient Descent:

Minimum error tak pahuchna by finding the best possible weights.

c) Explain the concept of gradient based Learning.

Gradient-based learning ek method hai jisme machine learning models (especially neural networks) loss/error ko minimize karne ke liye gradients ka use karte hain — taaki model better predictions kare.

🔁 Kaise kaam karta hai?

Step	Explanation
1. Prediction	Model input data ko use karke output banata hai.
2. Error Calculation	Actual output se compare karke error nikalta hai.
3. Gradient Find	Loss function ka derivative (gradient) calculate kiya jaata hai with respect to weights.
4. Update	Gradient ka use karke weights update kiye jaate hain (mostly using gradient descent).
5. Repeat	Ye cycle bar-bar repeat hoti hai jab tak model achhe se learn nahi kar leta.

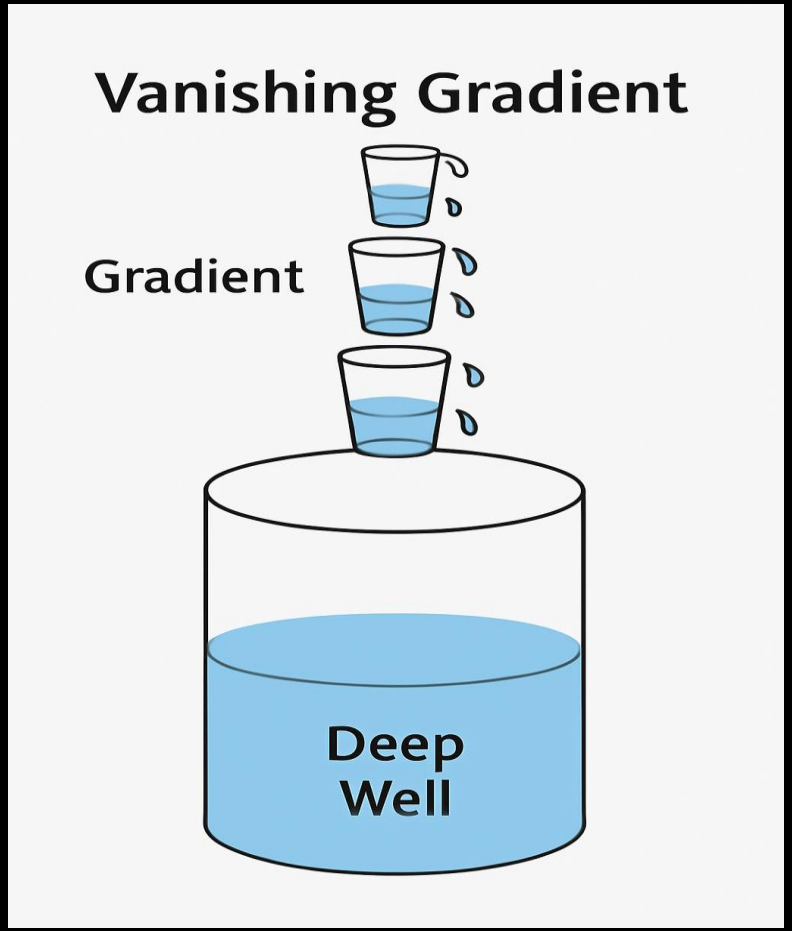
🔢 Gradient-Based Learning vs Gradient Descent

Gradient Descent	Gradient-Based Learning
Ek specific algorithm hai	Ek broader learning approach hai
Loss ko minimize karta hai	Gradient ka use karke model training karta hai
Ek part of gradient-based learning	Umbrella term – multiple methods (SGD, Adam, etc.) use hote hain

a) What is the problem of vanishing Gradient? Describe various solutions to this problem. [7]

🧠 Vanishing Gradient Problem

Jab neural network me **backpropagation** ke time **gradients (slopes)** bahut chhote ho jaate hain (≈ 0), to **initial layers ke weights update nahi hote**, aur network theek se learn nahi kar pata.
Ye problem **vanishing gradient** kehlata hai.



Socho ek deep well (kuan) hai jisme paani neeche hai. Agar tum ek glass (gradient) paani upar laane ke liye pass karte ho har level se, aur har level me thoda sa paani girta jaye... to final glass almost khaali hoga.

Yahi hota hai vanishing gradient.

🔍 **Kya hota hai isme?**

- Deep networks me backpropagation reverse me hoti hai (output layer → input layer).
- Har layer ka gradient previous layer me multiply hota hai.
- Agar activation function ka gradient < 1 ho (like sigmoid or tanh), to multiply karte karte **value zero ke paas** aa jaati hai.
- Result: **Initial layers "freeze" ho jaati hain**, learning stop ho jaata hai.

❗ Problems caused by Vanishing Gradient:

Problem	Impact
Gradient ≈ 0	Weights update nahi hote
Slow or No Learning	Model converge nahi karta ya bahut time leta hai
Poor Performance	Training fail ho sakta hai, accuracy low hoti hai

✅ Solutions to Vanishing Gradient Problem:

Solution	Explanation (Hinglish)
1. ReLU Activation	ReLU (Rectified Linear Unit) ka gradient 0 ya 1 hota hai → vanishing gradient avoid karta hai.
2. Weight Initialization (He/Xavier)	Proper initialization (Xavier for tanh, He for ReLU) se gradient flow improve hota hai.
3. Batch Normalization	Har layer ke input ko normalize karta hai → gradient stable rehta hai.
4. Skip Connections (Residual Nets)	Deep networks me direct connections add karte hain → gradient directly flow karta hai.
5. Use of LSTM/GRU in RNNs	RNNs me vanishing gradient common hota hai, to LSTM aur GRU cells design hi is problem ke liye hue hain.

🔢 Mathematical Explanation:

Let's say:

- Neural network me **loss function**: $\mathcal{L}$
- Weights: w
- Activation function: Sigmoid (σ), jiska derivative hamesha $0 < \sigma'(x) < 0.25$ hota hai

---

🔁 Backpropagation step:

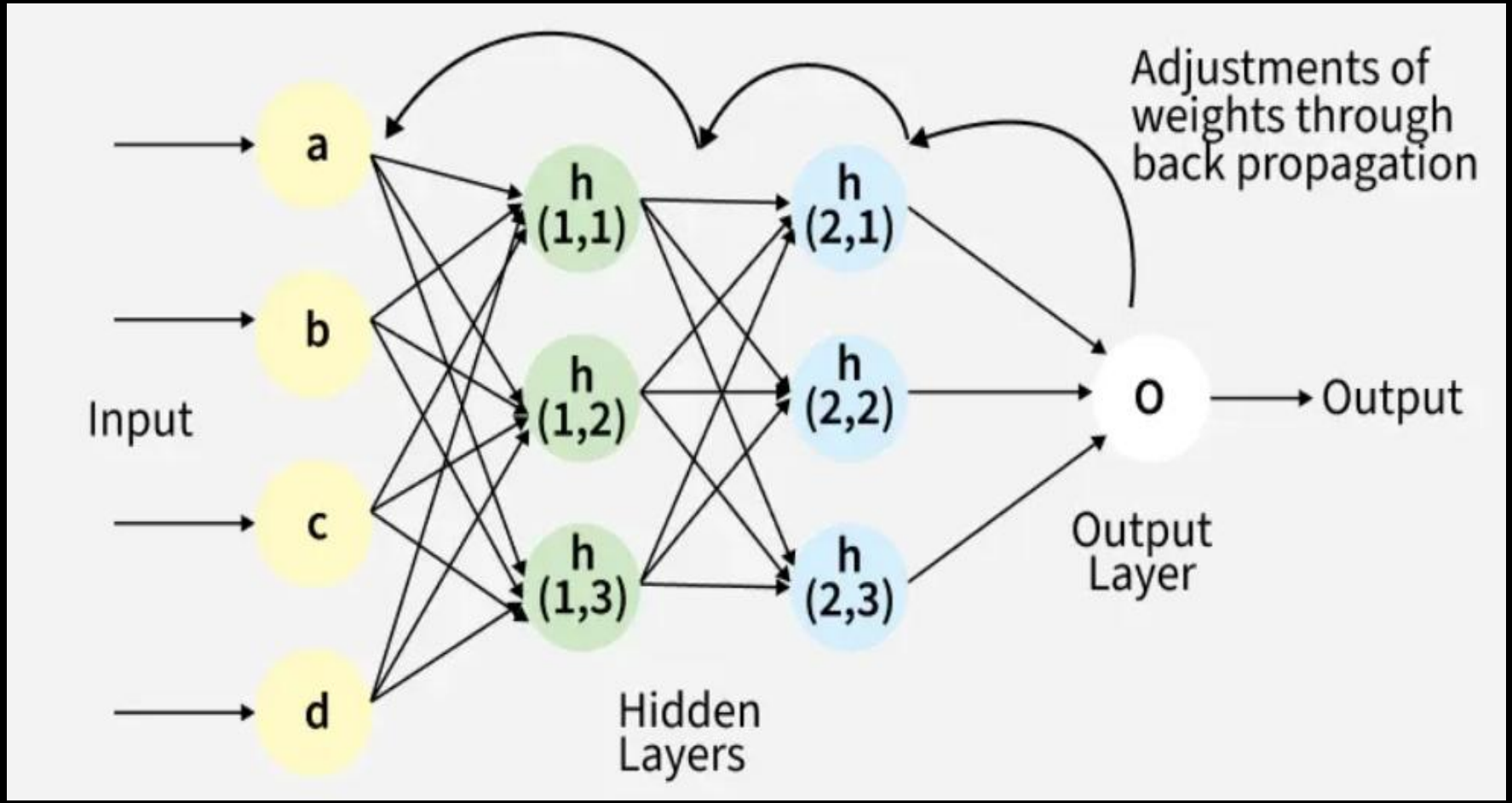
During backpropagation, gradients are calculated like this:

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial a_n} \cdot \frac{\partial a_n}{\partial a_{n-1}} \cdot \dots \cdot \frac{\partial a_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial w}$$

Each term (∂a/∂a) is often **< 1**, especially with sigmoid/tanh.

Toh jab ye terms multiply hote hain, toh final gradient:

$$\frac{\partial \mathcal{L}}{\partial w} \to 0$$


Adjustments of weights through back propagation
a
b
c
d
Input
h (1,1)
h (1,2)
h (1,3)
h (2,1)
h (2,2)
h (2,3)
Hidden Layers
O
Output
Output Layer

- **Input Layer** → 1 Neuron
- **Hidden Layer 1** → 1 Neuron (Sigmoid activation)
- **Hidden Layer 2** → 1 Neuron (Sigmoid activation)
- **Output Layer** → 1 Neuron

Step 1: Input and Initial Weights

- Input $x = 1$
- Weight from input to H1: $w_1 = 0.5$
- Weight from H1 to H2: $w_2 = 0.5$
- Weight from H2 to output: $w_3 = 0.5$

Step 2: Forward Pass

Using sigmoid:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- H1 output:

$$h1 = \sigma(w_1 \cdot x) = \sigma(0.5) \approx 0.622$$

- H2 output:

$$h2 = \sigma(w_2 \cdot h1) = \sigma(0.311) \approx 0.577$$

- Final output:

$$\hat{y} = \sigma(w_3 \cdot h2) = \sigma(0.2885) \approx 0.5715$$

Step 3: Backpropagation (Finding Gradients)

Assume Loss Function:

$$L = \frac{1}{2}(y - \hat{y})^2$$

Suppose target $y = 1$

Gradient at output layer:

$$\frac{\partial L}{\partial w_3} = (\hat{y} - y) \cdot \sigma'(w_3 \cdot h2) \cdot h2$$

Since:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \approx 0.5715 \cdot (1 - 0.5715) \approx 0.245$$

Gradient:

$$= (0.5715 - 1) \cdot 0.245 \cdot 0.577 \approx -0.06$$

🔁 **Gradient at w_2 and w_1:**

When backpropagating to earlier layers:

- Derivatives chain multiply with:

$$\sigma'(w_2 \cdot h1) \approx 0.245, \quad \sigma'(w_1 \cdot x) \approx 0.235$$

So gradient becomes:

$$\text{Final gradient} \approx -0.06 \cdot 0.245 \cdot 0.235 \approx -0.0035$$

🔍 Observation:

- Output layer gradient ≈ -0.06
- But initial layers get only ≈ -0.0035

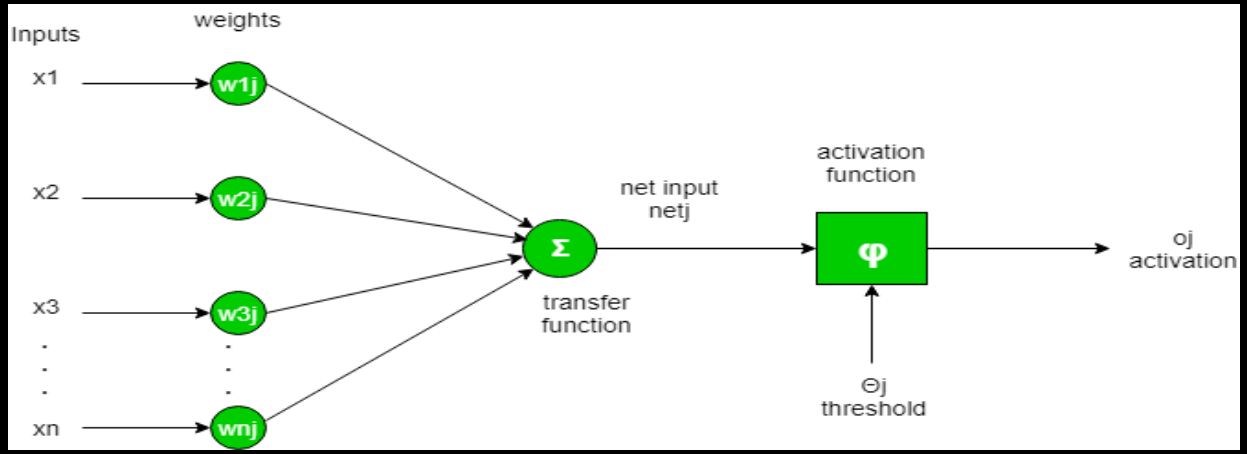
➡️ **Gradient almost vanish ho gaya** — that's vanishing gradient.

An artificial neuron is modeled (loosely) on how biological neurons work:

•Biological neurons take in signals from other neurons.

•They combine those signals, decide whether to “fire” or not, and send the result onward.

Artificial neurons do the same thing — but with math.

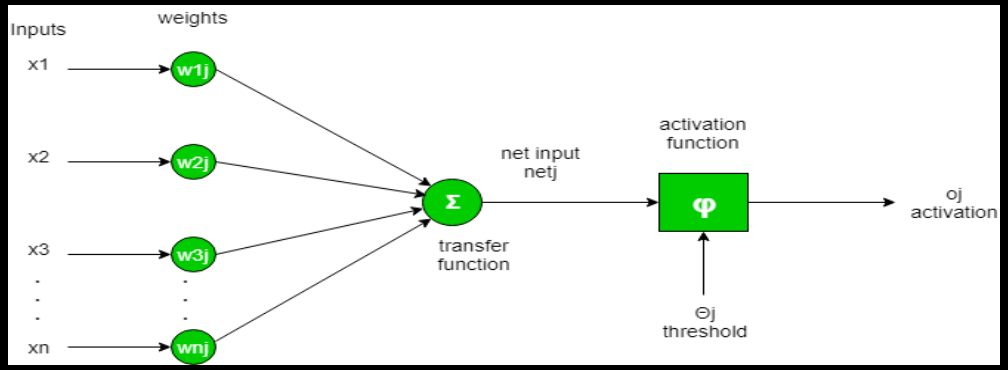


$$\text{Net Input} = \sum (\text{Weight} \times \text{Input}) + \text{Bias}$$

Step	Name	Kya Hota Hai	Formula	Real-Life Analogy
1	Inputs	Data ke alag-alag values lena (jaise features)	$x_1, x_2, x_3, \ldots, x_n$	Cooking ke liye alag-alag ingredients ready karna (flour, sugar, eggs)
2	Weighting	Har input ko uske weight se multiply karna	$w_1 \cdot x_1, w_2 \cdot x_2, \ldots$	Decide karna kitna ingredient dalna hai (zyada sugar → sweet zyada)
3	Summation	Sab weighted inputs ko add karna	$\sum_{i=1}^{n}(w_i \cdot x_i)$	Sab ingredients ko bowl mein mix karna
4	Add Bias	Ek constant add karna jo output ko shift kare	$z = \sum_{i=1}^{n}(w_i \cdot x_i) + b$	Thoda namak ya special masala dalna jo har dish mein hota hai
5	Activation Function	zz ko transform karna non-linear function se	$a = f(z)$ (sigmoid, ReLU, tanh)	Taste karke decide karna dish serve hone layak hai ya nahi
6	Output	Final output next layer ya result ko dena	a	Tayar dish ko guest ko serve karna

Activation Function

Activation function ek mathematical function hota hai jo neuron ke output ko decide karta hai.
Ye non-linearity introduce karta hai taaki neural network complex problems (like images, language) solve kar sake.



- **Jab neuron input receive karta hai (weighted sum + bias), activation function us input pe lagta hai.**
- **Fir decide hota hai output kya hoga — fire karega ya nahi.**
- **Without activation function, pura neural network just linear ban jaata.**

🛠️ Why Needed?

Without Activation	With Activation
Only linear relationships	Can model non-linear, complex patterns
Limited learning capacity	Deep learning becomes possible

Name	Formula / Graph Shape	Range	Speciality (Hinglish)
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	(0, 1)	Smooth curve. Good for probability. But causes **vanishing gradient**.
Tanh	$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	(−1, 1)	Zero-centered. Better than sigmoid, but still vanishes.
ReLU	$f(x) = \max(0, x)$	[0, ∞)	Fast & simple. No vanishing for positive values. Very popular.
Leaky ReLU	$f(x) = \max(0.01x, x)$	(−∞, ∞)	ReLU ke negative side me thoda output deta hai. Avoids dead neurons.
Softmax	$\frac{e^{z_i}}{\sum e^{z_j}}$	(0, 1), sums to 1	Used in output layer for multi-class classification.
Swish / GELU	Newer functions used in modern deep learning	—	Smarter smooth activation, better results in deep models

1 ReLU (Rectified Linear Unit)

Formula:

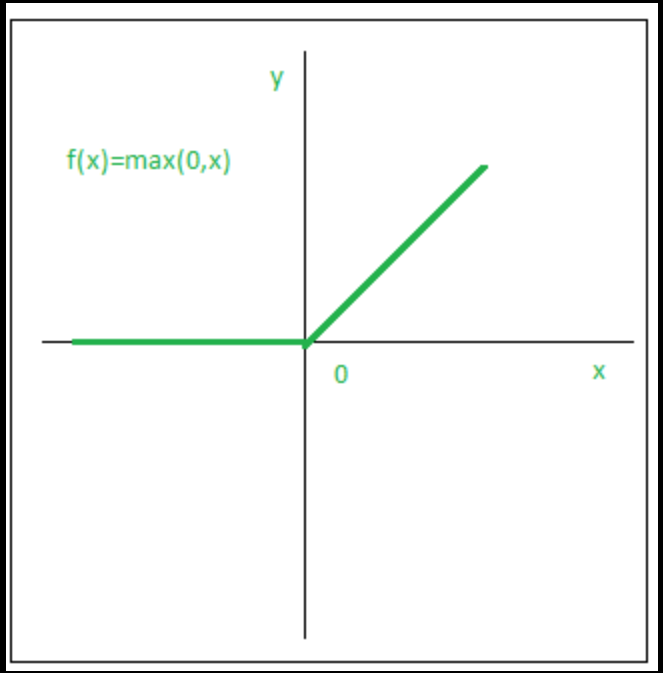
$$f(x) = \max(0, x)$$

Kaise kaam karta hai:

- Agar $x > 0$ hai → output x hota hai.
- Agar $x \leq 0$ hai → output 0 hota hai.
- Simple thresholding karta hai zero par.

Graph idea:

Ek straight line hai positive side me aur negative side me flat line (zero).



Pros:

•Fast computation (no exponentials, no complex math).

•Vanishing gradient ka problem kam hota hai (positive side ka slope 1 hai).

•Sparse activation (kaafi neurons off hote hain, network efficient hota hai).

Cons:

•“Dying ReLU problem”: agar neuron ka weight aise adjust ho gaya ki hamesha x<=0 ho, to wo neuron permanently zero output dega aur seekhna band kar dega.

2 Leaky ReLU (LReLU)

Formula:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

Jaha α ek chhota constant hai (e.g., 0.01).

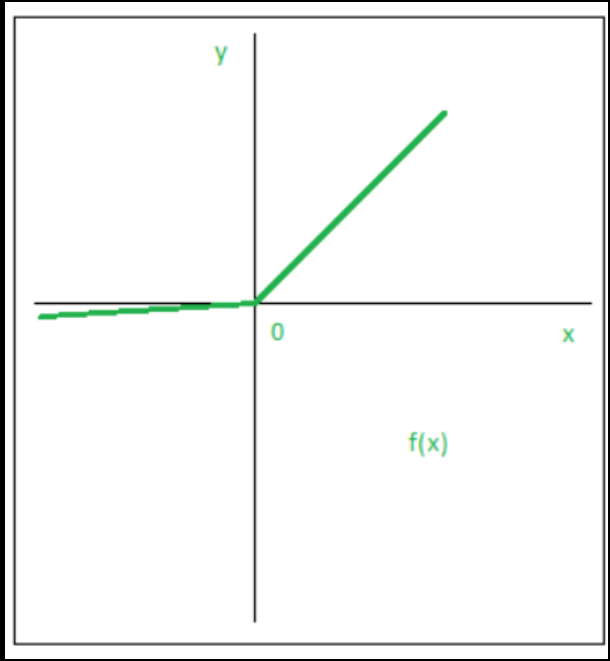
Kaise kaam karta hai:

- Positive side me ReLU jaisa hi kaam karta hai.
- Negative side me output zero nahi hota, balki ek chhoti slope hoti hai.

Graph idea:

Positive side → straight line slope 1.

Negative side → halka slope α ke saath neeche jata hua.



Pros:

•Dead neuron ka problem solve hota hai.

•Gradient kabhi zero nahi hota, isliye training smooth hoti hai.

Cons:

•Thoda randomness aa sakta hai agar α\alphaα fixed ho.

•ReLU se thoda slower hai (extra multiply hota hai negative side me).

3 ELU (Exponential Linear Unit)

Formula:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$$

Jaha $\alpha > 0$ ek parameter hota hai (usually 1).

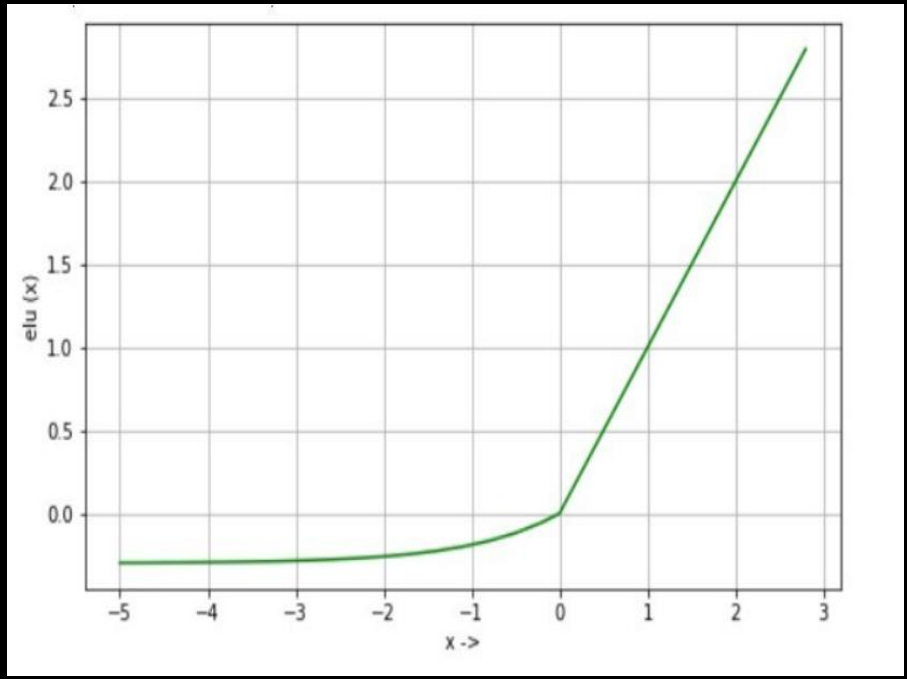
Kaise kaam karta hai:

- Positive side: straight line slope 1 (ReLU jaisa).
- Negative side: exponential curve jo $-\alpha$ pe smoothly saturate hota hai.

Graph idea:

Positive side me linear growth.

Negative side me smooth exponential decay jo flat ho jata hai neeche.



Pros:

•Gradient smooth hota hai, training me stability badhata hai.

•Output zero-centered hota hai (negative values allow karta hai), jo learning ko accelerate karta hai.

•Deep networks me vanishing gradient kam hota hai.

Cons:

•Thoda slow computation (because of exe^xex).

•Parameters tune karne padte hain (α\alphaα ka value important hota hai).

Feature	ReLU	Leaky ReLU	ELU
Negative values	0	Small slope (αx)	Smooth curve → −α
Vanishing gradient	Kam hota hai	Almost avoid	Almost avoid
Dead neuron issue	Haan	Nahi	Nahi
Computation speed	Fastest	Fast	Thoda slow (exp)
Output range	[0, ∞)	(-∞, ∞) small slope negative	(-α, ∞)

Explain the working of an Artificial neuron. Also explain the activation functions ReLU and LReLU. [8]

Explain loss function for regression operation.

Define and explain the significance of learning rate of a model?

OR

Explain loss function for classification operation.

1 Loss Function

Loss function ek mathematical formula hai jo **model ki prediction aur actual values ke beech ka difference** measure karta hai.

- **Purpose:** Batana ki model kitna galat hai.
- **Output:** Ek single number (error value).
- **Goal:** Is value ko minimize karna.

A) Loss Function for Regression — Mean Squared Error (MSE)

Concept:

Regression problems me hum continuous numbers predict karte hain (e.g., ghar ka price, temperature).
Loss function ka kaam hai batana ki model ki prediction actual value se kitni dur hai.

Formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Explanation:

- y_i = actual/true value
- $\hat{y}_i$ = predicted value
- n = total data points
- Squared term ensures ki negative aur positive errors cancel na ho.
- Square ka ek fayda ye hai ki **bada error zyada penalty deta hai**, isliye model bade galtiyo ko pehle fix karta hai.

Example:

Agar actual values [3, 5] aur predicted values [2, 7] hain:

$$MSE = \frac{(3-2)^2 + (5-7)^2}{2} = \frac{1+4}{2} = 2.5$$

Kyon use hota hai:

- Smooth function → gradient nikalna easy.
- Outliers par thoda sensitive hota hai, jo kabhi kabhi disadvantage bhi ho sakta hai.

B) Loss Function for Classification — Cross-Entropy Loss

Concept:

Classification problems me hum categories predict karte hain (e.g., "Cat" vs "Dog").

Prediction probability hoti hai, jaise:

- Cat: 0.9, Dog: 0.1 → model sure hai ki cat hai.

Cross-Entropy Loss ye measure karta hai ki predicted probability actual label se kitni match karti hai.

Binary Classification Case:

Formula:

$$Loss = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

- Agar $y_i = 1$ (positive class) → pehla term active hoga: $-\log(\hat{y}_i)$
- Agar $y_i = 0$ (negative class) → doosra term active hoga: $-\log(1 - \hat{y}_i)$

Example:

Actual label $y = 1$ (Cat), Predicted probability $\hat{y} = 0.9$:

$$Loss = -\log(0.9) \approx 0.105$$

(Chhota loss → prediction sahi aur confident)

Agar $\hat{y} = 0.2$ hota:

$$Loss = -\log(0.2) \approx 1.609$$

(Bada loss → prediction galat aur confident)

Kyon use hota hai:

- Probabilities ke saath naturally kaam karta hai.
- Model ko correct class ke liye high confidence dena sikhata hai.
- Multi-class classification ke liye bhi extend hota hai (Softmax + Cross-Entropy).

The **Optimization Algorithms** deep learning me ek tarah ka GPS hote hain jo tumhare model ko best parameters (weights) tak le jaate hain — taaki loss function minimum ho jaye.

Optimization Algorithm — Concept

- Neural network me training ka main goal hota hai **Loss Function ko minimize karna**.
- Optimization algorithm ek mathematical method hai jo **weights and biases** ko aise update karta hai ki loss gradually kam ho.
- Ye mostly **Gradient Descent** aur uske variants pe based hote hain.

1. Gradient Descent (Basic)

Formula:

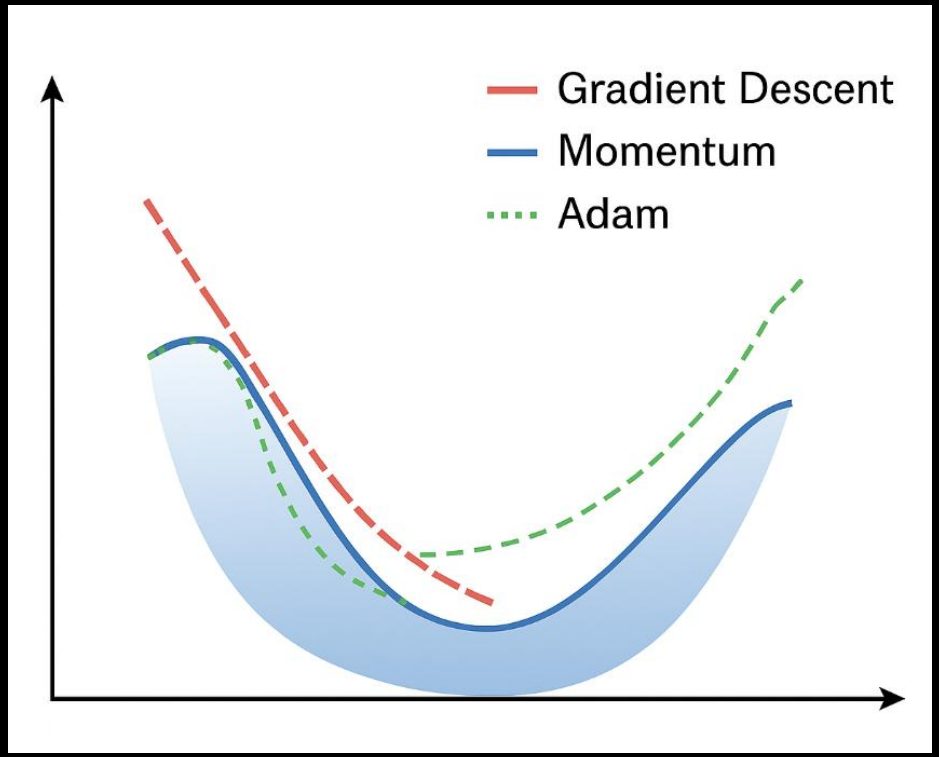
$$\theta = \theta - \eta \cdot \nabla_\theta L(\theta)$$

- $\theta \rightarrow$ model parameters (weights)
- $\eta \rightarrow$ learning rate
- $\nabla_\theta L(\theta) \rightarrow$ loss function ka gradient w.r.t. parameters

Drawback: Slow ho sakta hai, local minima me fas sakta hai.

2. Gradient Descent ke Variants

Algorithm	Key Idea	Pros	Cons
SGD (Stochastic Gradient Descent)	Har update me ek ya chhote batch ka use karta hai	Fast updates, large datasets ke liye efficient	Noisy updates
Momentum	Past gradients ka weighted average use karta hai	Local minima se nikalne me help karta hai	Extra parameter (momentum)
AdaGrad	Each parameter ke liye alag learning rate	Sparse data me useful	Learning rate bahut kam ho sakta hai
RMSProp	Learning rate ko moving average se control karta hai	RNNs ke liye acha	Still needs tuning
Adam	Momentum + RMSProp ka combination	Most popular, fast convergence	Sometimes overfits



Socho tum ek pahad se neeche utar rahe ho (loss surface). Gradient Descent: Sidha slope ke opposite direction me chalte ho. Momentum: Thoda inertia ke saath smoothly neeche jaate ho. Adam: Smartly har direction me step size adjust karke neeche jaate ho.

- **Hyperparameters**:
  - Layer size
  - Magnitude (momentum, learning rate)

b) What is hyper parameter? Describe categories of hyper parameters? **[5]**

1. Hyperparameter kya hai?

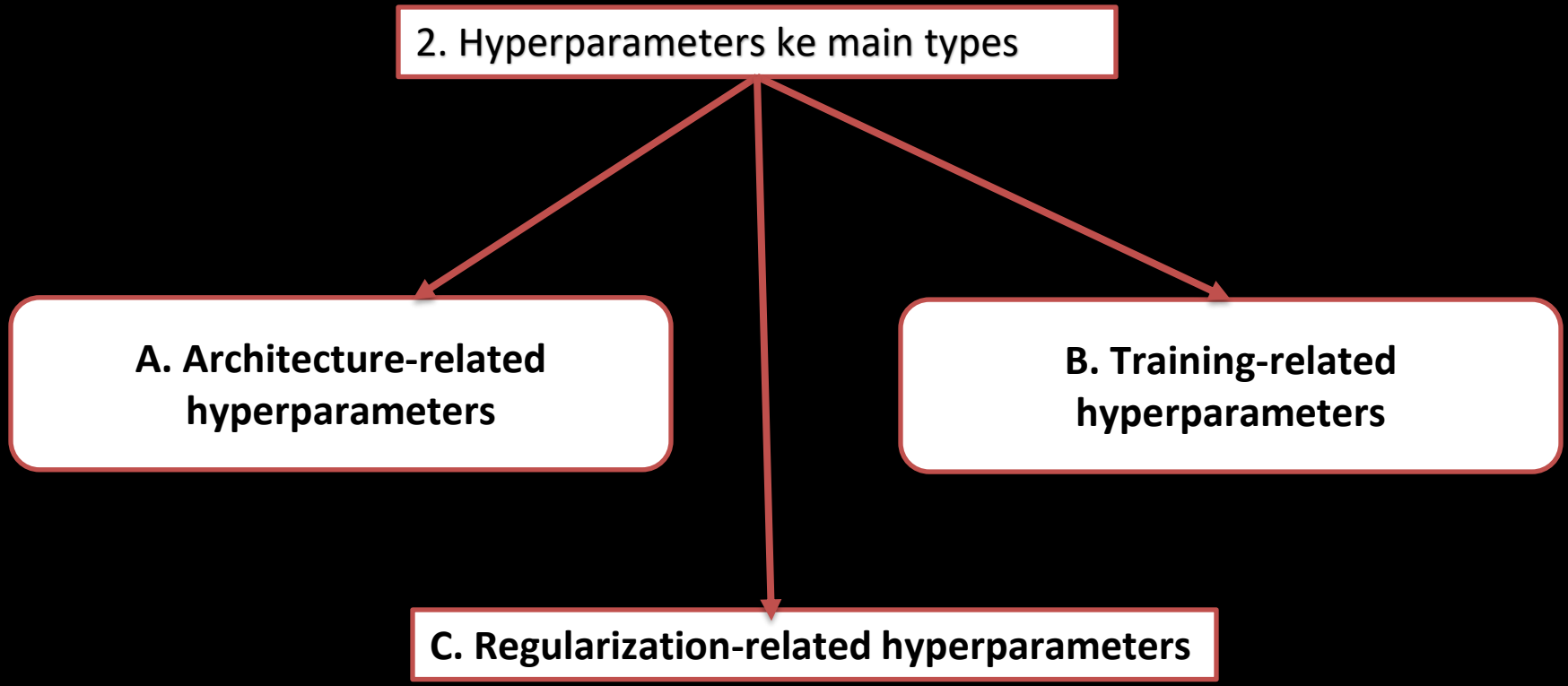
•Machine learning / deep learning me **hyperparameters** wo settings hote hain jo hum **model training se pehle set karte hain**.

•Ye model ke learning process ko control karte hain, lekin **model ke parameters (weights, biases)** ki tarah training ke during automatically learn nahi hote.

•Example: learning rate, number of layers, batch size, dropout rate, optimizer type, etc.

💡 Simple analogy:

Agar model ek student hai aur training ek exam preparation —
•**Parameters** = student ka knowledge jo woh padhai karke improve karta hai.
•**Hyperparameters** = padhai ka plan (kitne ghante padhna, kaunsa book use karna, kis speed se padhna).


2. Hyperparameters ke main types
A. Architecture-related hyperparameters
B. Training-related hyperparameters
C. Regularization-related hyperparameters

1. Layer Size (ek layer me kitne neurons hote hain)

•Definition:

Neural network me har layer me kuch neurons hote hain jo input ko process karke agle layer ko bhejte hain.

Layer size = ek layer me neurons ki total ginti.

•Effect on model capacity:

- **Zyada neurons:**
  - Network ki representational capacity badh jati hai → complex patterns seekh sakta hai (jaise images me chhoti chhoti details ya NLP me rare word patterns).
  - Risk: Overfitting ho sakta hai, kyunki model training data ko yaad karne lagta hai.
  - Training time zyada lagta hai, aur memory usage bhi badh jata hai.
- **Kam neurons:**
  - Model simple ho jata hai → fast training, kam memory usage.
  - Risk: Underfitting, yani model data ki complexity ko capture nahi kar paata.

💡 Example:

Socho tum ek team bana rahe ho problem solve karne ke liye.
•Zyada members (zyada neurons) → team me alag-alag skills, har cheez ka solution mil sakta hai, lekin coordination mushkil aur kaam slow ho sakta hai.

•Kam members (kam neurons) → fast decision making, lekin har problem solve nahi ho paayegi.

2. Number of Layers

•**Definition:**
Neural network me kitni sequential layers hain jo input se output tak ka processing chain banati hain.
•**Effect on learning:**

- **Zyada layers (Deep Network):**
  - Model complex hierarchical features seekh sakta hai.
    - Example: Image me
      - First layers: edges detect karte hain
      - Middle layers: shapes detect karte hain
      - Last layers: complete objects detect karte hain
  - Risk: Overfitting + vanishing/exploding gradient problems (agar architecture aur optimization sahi na ho).
  - Training me zyada compute power aur data ki zaroorat hoti hai.

- **Kam layers (Shallow Network):**
  - Simple tasks ke liye kaafi hote hain.
  - Complex data patterns ko capture nahi kar paate.

💡 **Example:**
Socho tum ek detective ho jo ek mystery solve kar raha hai.
•Deep investigation (zyada layers) → tum chhote clues se badi kahani samajh sakte ho.

•Surface-level investigation (kam layers) → tum bas basic info pakadoge, lekin hidden details miss ho sakti hain.

B. Training-related hyperparameters

1. Learning Rate

•**Definition:**

Ye ek numerical value hai jo decide karti hai ki har training step me hum weights kitna update karenge.

•**Kyun important hai:**

Gradient descent me, hum loss function ke slope ke opposite direction me move karte hain. Learning rate decide karta hai ki ye move *kitna bada* hoga.

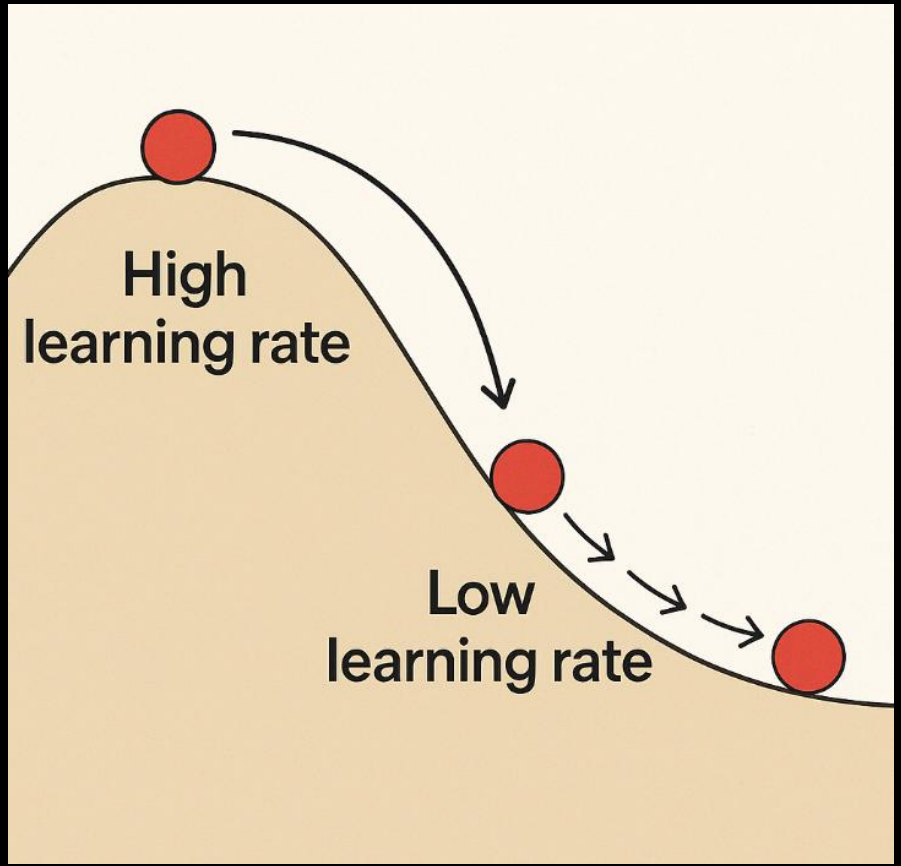
•**Case 1 — High Learning Rate:**

- Pros: Training fast hota hai.
- Cons: Loss function me chhote minima miss ho sakte hain, ya training bilkul diverge ho sakti hai (loss barh sakta hai).

•**Case 2 — Low Learning Rate:**

- Pros: Training stable hota hai.
- Cons: Bahut slow hota hai, kabhi kabhi local minima me atak jata hai.

Example:
Agar ek pahadi se neeche girte hue ball ho, aur tum ek hi jump me neeche pohchne ki koshish karo (high LR) to ball chala jaa sakta hai aur wapas upar chadh sakta hai. Agar chhote chhote steps lo (low LR), to tum safe rahoge lekin zyada time lagega.



2. Momentum

•Definition:
Ye ek technique hai jo gradient descent me past updates ka effect current update me include karti hai.

•Kyun important hai:
Jab loss surface me zig-zag pattern hota hai (especially steep slopes me), momentum help karta hai ki hum smooth aur fast direction me move karein.
•Kaise kaam karta hai:

- **Har update me hum gradient + previous update ka kuch percentage add karte hain.**
- **Isse hum consistent direction me speed gather karte hain, aur oscillations kam ho jaati hain.**

•**Example:**
Cycle chalate hue downhill me agar tum continuously pedal maarte raho, to tum gradually speed gain karoge aur chhote pathar ya obstacles ko easily cross kar loge — yahi momentum ka kaam hai training me.

3. Batch Size

•**Definition:**
Ek training step me jitne samples ka use karke gradient calculate kiya jaata hai, usko batch size kehte hain.

•**Kyun important hai:**
Ye memory usage, training speed, aur convergence behavior ko impact karta hai.

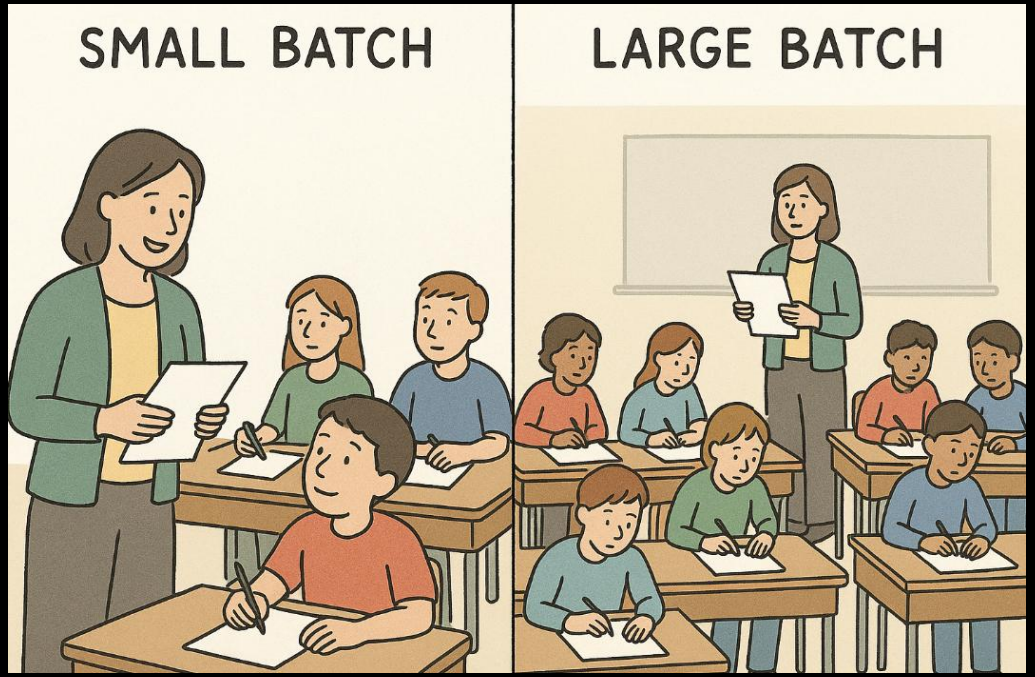
•**Types:**

- **Small Batch (e.g., 16, 32)**:
  - Pros: Memory kam lagti hai, gradients me thoda noise hota hai jo generalization improve kar sakta hai.
  - Cons: Zyada steps lagte hain ek epoch complete karne me.

- **Large Batch (e.g., 256, 1024)**:
  - Pros: Training fast hota hai per epoch.
  - Cons: Memory zyada lagti hai, generalization kabhi kabhi weak hota hai.

•**Example:**
Agar tum school me ho aur teacher tumse homework check kare:

- Small batch = har 2-3 students ke baad check karna (frequent feedback, thoda noisy).
- Large batch = poora class ka homework ek saath check karna (consistent but feedback late).



- **Regularization**:
  - Dropout
  - Drop connect
  - L1 Regularization
  - L2 Regularization

Regularization kya hota hai?

•Regularization ek **technique** hai jo machine learning / deep learning models me **overfitting** ko kam karne ke liye use hoti hai.

•Iska kaam hai model ko simple rakhna aur ensure karna ki wo **training data ke noise** ya **irrelevant details** ko yaad na kare.

•Ye training ke loss function me ek extra penalty add karke ya model ke architecture/training process me randomness introduce karke achieve hota hai.

Kyon zaroori hai?

1. Overfitting ka problem

•Jab model training data par bahut acche se fit ho jata hai (even noise ko bhi memorize kar leta hai) lekin new/unseen data par poor performance deta hai → ise overfitting kehte hain.

•Overfit model real-world me fail ho jata hai kyunki wo **generalize** nahi kar pata.

2. Regularization ka role

•Regularization model ko "too complex" hone se rokti hai.

•Isse model sirf **essential patterns** seekhta hai, random noise nahi.

•Result: Better generalization → unseen data par zyada accurate predictions.

💡 **Simple analogy:**

Socho tum ek exam ke liye prepare kar rahe ho:

•Agar tum har ek question ka exact wording yaad karte ho (overfitting), to tum sirf wahi paper likh paoge jo tumne padhe hain.
•Agar tum concept samajh ke practice karte ho (regularization), to tum naye pattern ke questions bhi solve kar paoge.

Regularization methods

1. Dropout

•**Kya hai:**

Training ke time randomly kuch neurons ko temporarily deactivate kar dete hain.

•**Kyun:**

Taaki model kisi specific neuron par over-rely na kare, aur distributed representation seekhe.

•**Kaise kaam karta hai:**

Har training step me ek probability (p) set hoti hai. Example: p = 0.5 → har step me 50% neurons randomly off.

•**Effect:**

Training thoda noisy hota hai (good for generalization), lekin test time me saare neurons active hote hain.

💡 **Example:**

Socho ek cricket team me har practice session me randomly 2-3 players ko rest de diya jata hai. Isse har player alag-alag roles me trained ho jata hai.

2. DropConnect

•Kya hai:
•
Dropout neurons ke outputs ko remove karta hai, lekin DropConnect neurons ke connections (weights) ko randomly deactivate karta hai.

•Kyun:
Ye ek aur tarika hai model ko robust banane ka, taaki wo kisi ek specific weight par over-rely na kare.

•Effect:
Zyada fine-grained randomness introduce hoti hai, jo kabhi kabhi Dropout se better regularization deta hai.

💡 Example:
Agar cricket me player ko rest dene ke bajay unke kuch shots ya skills temporarily ban kar diye jayein (e.g., sirf defensive shots practice karna), to team multiple strategies seekh legi.

3. L1 Regularization (Lasso)

•**Kya hai:**

Loss function me weights ke absolute value ka sum add karte hain.

•**Formula:**

$$Loss' = Loss + \lambda \sum |w_i|$$

•**Kyun:**

Ye large weights ko penalize karta hai aur bahut se weights ko zero banane ki tendency rakhta hai → sparse model (feature selection me useful).

•**Effect:**

Features me se sirf relevant ones ko use karta hai.

💡 **Example:**

Socho tum apne study table se sirf zaroori books rakhte ho aur baaki hata dete ho — taaki distractions kam ho.

4. L2 Regularization (Ridge)

•**Kya hai:**

Loss function me weights ke squared value ka sum add karte hain.

•**Formula:**

$$Loss' = Loss + \lambda \sum w_i^2$$

•**Kyun:**

Ye large weights ko heavily penalize karta hai, lekin unhe zero nahi banata — instead unhe small rakhta hai.

•**Effect:**

Model smoother ho jata hai aur overfitting kam hota hai.

💡 **Example:**

Agar tum ek library me ho aur har book ka size balance karte ho taaki koi book bahut heavy na ho, lekin sab available rahe.

✨ Thank You for Watching! ✨

📲 Follow us on Instagram:**@jayesh_kande_**
🔗 Connect with us on LinkedIn:[**Jayesh Kande**]