

DL – Unit 2 (Deep Neural Networks (DNNs)) – IN-SEM PYQ Answers

Q1. What is Activation Function? Enlist and Explain different Activation functions used in Neural Network. [5]

Q2. Enlist Activation functions used in Deep Neural Network. Explain any two of them in detail.[5]

Activation function

- **mathematical function applied to the output of a neuron**, deciding whether it should be “activated” and how strongly it contributes to the next layer.
- It **introduces non-linearity** into the neural network, enabling it to model **complex, non-linear patterns** in data.
- Without activation functions, a neural network behaves like a linear model, unable to learn advanced relationships.

Importance:

- They **add non-linearity**, enabling neural networks to solve complex problems like image classification and language recognition.
- They help determine which features and patterns are **important** during learning, preventing the network from collapsing to trivial linear models.
- Different functions are **chosen based on task requirements** (classification vs regression, binary vs multi-class).

Activation Function	Definition & Working	Key Characteristics
Linear	Outputs the input value directly (identity function). It does not transform data non-linearly.	<ul style="list-style-type: none"> • Range: $(-\infty, \infty)$. • Used mainly in output layer for regression. • Cannot introduce non-linearity; if used in hidden layers, limits learning.
Sigmoid (Logistic)	Produces an S-shaped curve that maps input to values between 0 and 1 .	<ul style="list-style-type: none"> • Useful for binary classification output. • Squashes outputs to $(0,1)$. • Can cause vanishing gradient in deep networks.
Tanh (Hyperbolic Tangent)	Like sigmoid but maps input to -1 to +1 , centering outputs around zero.	<ul style="list-style-type: none"> • Zero-centered outputs aid optimization. • Stronger gradients than sigmoid but still can suffer vanishing gradient.
ReLU (Rectified Linear Unit)	Outputs 0 for negative inputs and the input itself for positive values: $\max(0, x)$.	<ul style="list-style-type: none"> • Most commonly used in hidden layers of deep networks. • Reduces vanishing gradient problem. • Simple and computationally efficient.
Leaky ReLU	Variation of ReLU that allows a small negative slope rather than completely zero for $x < 0$.	Helps reduce “dying ReLU” problem by keeping small gradient for negative region.
Softmax	Generalized sigmoid for multi-class classification . Converts raw outputs into a probability distribution across classes.	<ul style="list-style-type: none"> • Outputs sum to 1. • Used in final layer of classification models.

Q3. What is the Biological Neuron & Perceptron? What are the steps involved for training a Perceptron in Deep Learning?[5]

Q4. What is a Perceptron? What are the steps involved for training a Perceptron in Deep Learning? [5]

1. Biological Neuron

- A **biological neuron** is the fundamental information-processing unit of the human nervous system, responsible for receiving, integrating, and transmitting **electrical signals** to other neurons through **synapses**, enabling cognition and decision-making.
- Incoming signals from **dendrites** are summed in the **soma**; if the combined signal crosses a threshold, an **action potential (nerve impulse)** is generated and sent through the **axon** to other neurons.
- The structure and functioning of biological neurons **inspire artificial neural networks** where simplified computational units simulate their behavior.

2. Perceptron (Artificial Neuron)

- A **Perceptron** is the **simplest form of an artificial neural network** and acts as a basic **computational unit** that mimics a biological neuron. It combines multiple inputs with associated **weights**, applies a **bias**, and uses an **activation function** to produce a **binary output** for **binary classification** tasks.
- It consists of:
 - **Input Layer:** Receives feature values.
 - **Weights:** Each input is multiplied by a weight representing its influence.
 - **Bias:** Added to adjust decision boundary.
 - **Activation Function:** Typically a **step function** that outputs 0 or 1.
- A single perceptron forms a **single-layer network**; multilayer arrangements form deeper networks.

3. Steps Involved in Training a Perceptron(Perceptron Training Algorithm (Supervised Learning))

- 1. Initialize Weights and Bias:**
 - Assign initial **small random values** to weights and bias to break symmetry.
 - This allows the perceptron to learn different patterns for each input.
- 2. Compute Weighted Sum:**
 - For each training sample, compute the **weighted sum (net input)** of inputs plus bias:

$$\text{Weighted sum} = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

$$= w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$
 - Apply the **activation function** (step function) to determine the **predicted output (0 or 1)**.
- 3. Calculate Error:**
 - Compare the **predicted output** with the **desired (true) output**.
 - $\text{Error} = (\text{desired output} - \text{predicted output})$.
 - If the prediction is correct, no changes are made; if incorrect, proceed to adjustment.
- 4. Update Weights and Bias:**
 - Adjust each weight using the **perceptron learning rule**:

$$w_i \leftarrow w_i + \eta \times (t - o) \times x_i$$

$$w_i \leftarrow w_i + \eta \times (t - o) \times x_i$$
 where η is the **learning rate**, t is the **true label**, and o is the **output**.
 - Bias is adjusted similarly to shift the decision boundary.
 - This step reduces future errors by modifying weights in the direction that minimizes error.
- 5. Iterate Over Training Data:**

- Repeat steps 2–4 for each training sample and over multiple **epochs** until the perceptron achieves **correct classification** (or acceptable accuracy) on the training set.
- For linearly separable data, the algorithm converges to a stable solution.

Q5. What is Perceptron? Write short note on Multilayer Feed-Forward network with figure[5]

Multilayer Feed-Forward Network

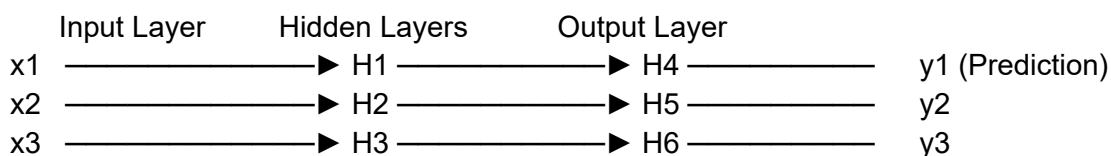
A **Multilayer Feed-Forward Network (MLFNN)** is a type of **artificial neural network** where information flows in one direction, from the input layer through one or more hidden layers to the output layer, without any feedback loops. It is the most common deep learning architecture used for **classification, regression, pattern recognition, and function approximation**.

Structure & Working

- **Input Layer:** Receives raw features (e.g., pixel values, numerical attributes).
- **Hidden Layers:** One or more layers of neurons where **non-linear transformations** are applied using **activation functions**. These layers extract **higher-level features** and patterns.
- **Output Layer:** Produces final predictions such as class labels or numeric outputs.
- **Feed-Forward Flow:** Signals pass **forward** only; there are no backward connections.
- **Learning:** Weights are optimized during training using **backpropagation** and **gradient descent**, minimizing an error function.

Key Characteristics

- **Non-linearity:** Activation functions (ReLU, sigmoid, tanh) introduce non-linear mapping, enabling the network to learn complex relationships.
- **Universal Approximation:** With sufficient hidden neurons and layers, it can approximate any continuous function.
- **Layered Feature Learning:** Early layers extract simple patterns; deeper layers learn abstract features.
- **Supervised Learning:** Typically trained with labeled data using backpropagation.



- **x1, x2, x3:** Input features
- **H1, H2, H3, H4, H5, H6:** Neurons in hidden layers
- **y1, y2, y3:** Outputs

Advantages

- Can learn **complex non-linear decision boundaries**.
- **Scalable** by adding more hidden layers and neurons.
- Widely used in **vision, speech, NLP, time-series forecasting**, and many real-world applications.

Q6. What is Back Propagation? Compare Back Propagation and Forward Propagation. [5]

Forward Propagation	Back Propagation
It is the process in a neural network where input signals pass forward through the layers — input → hidden → output — computing activations at each neuron using weights and activation functions to produce the predicted output .	It is the error-correction learning algorithm used to train neural networks where the error at the output is propagated backward through the network to update weights using gradient descent.
Computes the output of the network given current weights.	Computes the adjustments to weights to minimize the error based on gradients.
Works during inference and evaluation as well as the forward pass in training.	Works only during training to optimize the model.
No error feedback is involved; it is a one-way pass of signals.	Involves computation of error gradients and feedback from output layer towards input.
Faster and less computationally expensive compared to back propagation.	More computationally intensive due to gradient calculations.
Does not change weights; uses existing parameters to produce output.	Updates weights and biases to reduce loss over epochs.

Q7. Explain how a Neural Networks can be trained with Back propagation and Forward propagation methods.[5]

Forward Propagation

- 1. Start with Input Features:** Training begins with presenting an **input vector** (features) to the network's **input layer**.
- 2. Compute Weighted Sum at Each Neuron:** Each neuron multiplies inputs by their **weights** and adds a **bias**, computing a weighted sum: $z = w_1x_1 + w_2x_2 + \dots + b$
- 3. Apply Activation Functions:** The weighted sum passes through an **activation function** (e.g., ReLU, sigmoid) to introduce **non-linearity** and compute the neuron's **activation output**.
- 4. Signal Flows Layer by Layer:** Outputs of one layer become inputs to the next, continuing in a **forward direction** through all **hidden layers** until the **output layer**.
- 5. Produce Final Prediction:** The final layer processes activations to produce a **predicted output** vector corresponding to classes or regression values.
- 6. Purpose of Forward Pass:** This step computes a **prediction using current weights** and sets up the basis for error calculation in back propagation.

Back Propagation

1. Compute Prediction Error: After forward propagation, the predicted output is compared with the **true label/target** using a **loss function** (e.g., mean squared error, cross-entropy). This difference is the **error** the network must reduce.

2. Calculate Gradients (Error Signals): Back propagation computes the **gradient of the loss** with respect to each weight using the **chain rule**, determining how much each weight contributed to the error.

3. Propagate Error Backward: The error gradients are propagated **backwards from the output layer** to earlier layers, updating influence on each weight.

4. Update Weights and Biases: Using a **learning rate (η)** and gradients, weights and biases are adjusted in the direction that **reduces the error**:

– Weight update: $w: w - \eta \times (d \text{ Loss} / d w)$.

This step **optimizes parameters** and moves the model toward lower loss.

5. Iterate Over Training Set: Steps 1–4 are repeated for **each training sample or mini-batch** and over multiple **epochs** until the model converges to a low error.

Forward Propagation	Back Propagation
Computes prediction output using current weights.	Computes and back-propagates error to adjust weights.
Only involves signal flow forward through layers.	Involves error feedback and gradient computation.
No weight changes occur here.	Updates weights and biases to reduce loss.
Used in both training and inference.	Used only during training.
Determines model's current output for a sample.	Trains the model by reducing error iteratively.

Q8. What are the methods for tuning hyperparameters[5]

Hyperparameter tuning refers to the process of choosing the best set of **hyperparameter values** that control how a model is trained to achieve optimal performance. Hyperparameters are set **before training** and significantly affect model accuracy, generalization, convergence and computational cost.

Methods for Tuning Hyperparameters

1. Manual Search

- This involves **hand-tuning hyperparameters** based on intuition and observing model performance.
- Useful for simple models or initial exploration but can be **time-consuming and subjective**.

- Often combined with domain expertise to guide choices.

2. Grid Search

- Systematically defines a **grid of possible values** for each hyperparameter and evaluates every **combination**.
- Guarantees finding a good set in the predefined grid but becomes **computationally expensive** for large search spaces.
- Commonly used in smaller models or limited parameter sets.

3. Random Search

- Instead of exhaustively checking all combinations, it **randomly samples** values from the search space and evaluates them.
- More efficient than grid search since it explores more of the space with fewer evaluations.
- Suitable for high-dimensional tuning spaces.

4. Bayesian Optimization

- Uses a **probabilistic model** to predict promising hyperparameter configurations based on past evaluations.
- Iteratively updates a surrogate model to focus search in regions likely to produce better performance, reducing the number of trials needed.
- More efficient for expensive models and larger search spaces.

5. Gradient-Based / Advanced Optimization Methods

- Techniques like **gradient-based optimization**, **Evolutionary strategies** or **Hyperband** use **adaptive search** strategies to improve tuning efficiency.
- These methods aim to allocate computational resources more effectively and handle large search spaces with less computation.

Q9. What is the Loss function? Enlist all and Explain Loss functions.[5]

A **loss function** (also called **error function**): a **mathematical function that quantifies the error between a model's predicted output and the actual target (ground truth) value**.

- It measures how well (or poorly) a deep learning model is performing and provides a numerical "loss" value that the training process seeks to **minimize** through weight updates.
- Loss functions are central to training models using optimization algorithms like **gradient descent**.

Role of Loss Functions in Deep Learning:

- Loss functions provide a **measure of model performance** by quantifying prediction error.
- During training, deep learning models use this loss value to **guide weight and bias updates** via backpropagation and optimization algorithms.
- A well-chosen loss function ensures the model learns meaningful patterns and **improves accuracy** over training iterations.

Types of Loss Functions and Their Explanation

Loss Function	Explanation
Mean Squared Error (MSE)	Used for regression problems where the output is continuous. It computes the average of squared differences between predicted and actual values, penalizing larger errors more heavily.

Mean Absolute Error (MAE)	Another regression loss that calculates the average of absolute differences between predictions and actual targets. It treats all errors with equal weight and is less sensitive to outliers than MSE.
Binary Cross-Entropy Loss	Used for binary classification tasks (two output classes). It measures the divergence between the predicted probabilities and the actual binary labels, penalizing incorrect confident predictions.
Categorical Cross-Entropy Loss	Applied in multi-class classification problems. It measures the difference between the true class probability distribution (one-hot encoded) and the predicted probability distribution over classes.
Hinge Loss	Commonly used with margin-based classifiers such as SVM. It penalizes predictions that are not only wrong but also within a margin of error relative to the decision boundary, encouraging better class separation.

Q10. Explain Gradient Descent. Why does the vanishing or exploding gradients problem happen? [5]

Gradient Descent

- iterative **optimization algorithm** used to **minimize the loss function** during neural network training by adjusting weights in the direction of the steepest decrease of the loss.
- It computes the **gradient (partial derivatives)** of the loss with respect to each parameter and updates them to reduce error.
- The fundamental idea is that the **loss function decreases fastest** when moving opposite to the gradient of the loss at the current point.
- At each iteration:
 1. Calculate the **gradient of the loss** with respect to model parameters using **backpropagation** (chain rule).
 2. Update each weight by subtracting the **learning rate multiplied by the gradient** (steering towards a local minimum).
 3. Repeat until convergence or until a predefined number of epochs.
- Variants include **Batch Gradient Descent**, **Stochastic Gradient Descent (SGD)** and **Mini-batch Gradient Descent**, each balancing convergence speed and stability.
- Gradient descent enables deeper layers to learn by propagating error signals backward and refining weights to lower the loss.

Reason for Vanishing gradient and Exploding gradient

(a) Vanishing Gradient Problem

- **Definition:** In deep networks, gradients can become **very small (close to zero)** as they are back-propagated through many layers.
- **Cause:** When activation functions like **sigmoid or tanh** squash inputs into narrow ranges, their derivatives are small (<1). During backpropagation, repeated multiplication of small derivatives makes gradient values shrink exponentially as they traverse backward through layers.

- **Effect:**
 - **Early layers learn very slowly** because tiny gradients fail to update weights effectively.
 - The network struggles to capture **long-range dependencies** in deep hierarchies.
- **Common Occurrence:** Deep networks with many layers and **saturating activation functions** (sigmoid, tanh) are especially prone.

(b) Exploding Gradient Problem

- **Definition:** In contrast, gradients can become **very large** as they propagate backward in deep networks.
- **Cause:** When derivatives are greater than one or weights become large, repeated multiplication causes gradients to **grow exponentially**, leading to unstable updates.
- **Effect:**
 - Large updates to weights cause the training process to **diverge** instead of converging.
 - Loss may oscillate or fail to decrease.
- **Common Occurrence:** Networks with **poor weight initialization** or large learning rates can trigger exploding gradients.

Summary (Why These Problems Occur)

Vanishing Gradients	Exploding Gradients
Gradients shrink as they move backward	Gradients grow uncontrollably
Caused by activation functions with small derivatives	Caused by large weight values or high derivative values
Leads to slow or stalled learning in early layers	Leads to instability and divergence in training
Common in deep networks with sigmoid/tanh	Common in deep networks with improper scaling

Q11. Write short note on PyTorch and Colab. [5]

PyTorch

- **PyTorch** is an **open-source deep learning library/framework** that provides tools to build, train and evaluate neural networks using Python.
- It was developed by **Meta AI (formerly Facebook)** and is widely used in research and industry because of its **flexibility and ease of use**.
- PyTorch uses **tensors** as core data structures (similar to NumPy arrays) and supports **GPU acceleration** to speed up computation on large datasets.
- It features **automatic differentiation (Autograd)** which builds computation graphs dynamically, enabling efficient **backpropagation** for training neural networks.
- PyTorch's design supports **dynamic computation graphs**, making it intuitive for model experimentation and debugging in tasks such as image classification, NLP and reinforcement learning.

Google Colab

- **Google Colab (Colaboratory)** is a **free cloud-based Jupyter Notebook environment** provided by Google that lets users write and execute Python code in a web browser without local setup.
- It is especially useful for **machine learning and deep learning projects**, offering **pre-installed libraries** like PyTorch, TensorFlow, NumPy, Pandas, etc., so users can start coding immediately.
- Colab provides **free access to GPUs and TPUs**, enabling training of large models efficiently without requiring powerful local hardware.
- Notebooks are stored in **Google Drive**, auto-saved and easily shareable with others, supporting collaboration and real-time editing.
- It integrates with **Google Drive and GitHub**, and allows execution of shell commands and installation of additional packages, making it a flexible platform for experimentation and prototyping.

Q12. Explain Sentimental Analysis in detail and its types. [5]

Sentiment Analysis (also called **opinion mining**):

- **natural language processing (NLP) technique** that automatically **identifies and interprets emotional tone or opinion expressed in textual data**, such as reviews, social media posts, emails or surveys. It classifies text based on the **emotional polarity** whether it is **positive, negative or neutral** or even more nuanced feelings and attitudes.
- Its goal is to help organizations and systems understand user opinions and feelings from large volumes of unstructured text data, enabling **data-driven decision-making**.

Types of Sentiment Analysis

- 1. Polarity/Document-Level Sentiment Analysis**
 - Classifies the **overall sentiment of an entire document or sentence** as **positive, negative, or neutral** based on the expressed opinion.
 - Useful for general opinion mining in long texts or full reviews.
- 2. Fine-Grained Sentiment Analysis**
 - Extends polarity classification to **graded levels** such as *very positive, positive, neutral, negative, and very negative*, providing more **detailed emotional intensity**.
 - Often expressed through scales (e.g., 1–5 star ratings).
- 3. Aspect-Based Sentiment Analysis (ABSA)**
 - Focuses on detecting sentiment **towards specific aspects or features** of a product, service, or entity rather than overall sentiment.
 - For example, distinguishing sentiment about *camera quality* vs *battery life* in a phone review.
- 4. Emotion Detection**
 - Goes beyond polarity to identify **specific emotions** expressed in text such as *joy, anger, sadness, frustration*, etc.
 - Helps understand the psychological state or deeper motivations behind language.
- 5. Multilingual / Context-Sensitive Analysis**
 - Analyses sentiment across different **languages** or **contextual nuances**, requiring more complex preprocessing and language resources.
 - Useful when handling global or culturally diverse text data.