**Assignment Code: DS-AG-019**

Neural Network - A Simple Perceptron |

Assignment

**Instructions:** Carefully read each question. Use Google Docs, Microsoft Word, or a similar tool to create a document where you type out each question along with its answer. Save the document as a PDF, and then upload it to the LMS. Please do not zip or archive the files before uploading them. Each question carries 20 marks.

**Total Marks**: 200

**Question 1:** What is Deep Learning? Briefly describe how it evolved and how it differs from traditional machine learning.

**Answer:**

Deep Learning is a subset of Machine Learning that uses artificial neural networks with multiple

hidden layers to automatically learn hierarchical representations from data. Unlike traditional

machine learning, which relies heavily on manual feature extraction, deep learning models can

directly learn features from raw inputs such as images, text, or audio. Evolution: - 1950s: Early

concept of neural networks (Perceptron). - 1980s: Backpropagation algorithm improved training of

multi-layer networks. - 2000s: Growth of big data, powerful GPUs, and better algorithms led to

practical adoption. - Modern era: Convolutional Neural Networks (CNNs), Recurrent Neural

Networks (RNNs), and Transformers enable breakthroughs in computer vision, natural language

processing, and speech recognition. Key Differences from Traditional Machine Learning: 1. Feature

Engineering: - Traditional ML: Requires manual feature extraction (e.g., edges, shapes in images). -

Deep Learning: Automatically learns features from raw data. 2. Data Requirements: - Traditional

ML: Works well with small datasets. - Deep Learning: Requires large datasets for accuracy and

generalization. 3. Model Complexity: - Traditional ML: Simpler models (Decision Trees, SVMs,

Linear Regression). - Deep Learning: Complex neural networks with millions of parameters. 4.

Computational Needs: - Traditional ML: Less computationally intensive. - Deep Learning: Requires

high computational power (GPUs/TPUs). In summary, Deep Learning is an advanced form of

machine learning enabled by large datasets and high computing power. It differs by automatically

learning features, scaling better with complex data, and achieving state-of-the-art performance across multiple domains.

**Question 2:** Explain the basic architecture and functioning of a Perceptron. What are its limitations?

**Answer:**

A Perceptron is the simplest type of neural network, consisting of a single neuron. It takes weighted

inputs,

sums them, applies an activation function, and produces an output.

Architecture:

- Inputs (x1, x2, ..., xn).

- Weights (w1, w2, ..., wn).

- Summation unit computes weighted sum.

- Activation function decides final output.

Limitations:

- Only works for linearly separable problems (e.g., AND, OR).

- Cannot solve XOR problem.

- Limited representational power compared to multilayer network

**Question 3:** Describe the purpose of activation function in neural networks. Compare Sigmoid, ReLU, and Tanh functions.

**Answer:**

**Question 4:** What is the difference between Loss function and Cost function in neural networks? Provide examples.

**Answer:**

Activation functions introduce non-linearity into neural networks, allowing them to model complex

relationships.

- Sigmoid: Range (0,1). Smooth but suffers from vanishing gradients.

- ReLU (Rectified Linear Unit): Range (0,¥). Efficient and reduces vanishing gradient but may

cause "dead neurons".

- Tanh: Range (-1,1). Zero-centered but still affected by vanishing gradients.

Purpose: Without activation functions, neural networks behave like linear regression models.

**Question 5:** What is the role of optimizers in neural networks? Compare Gradient Descent, Adam, and RMSprop.

**Answer:**

Optimizers update weights to minimize loss functions.

- Gradient Descent: Simple, updates in direction of negative gradient, may be slow.

- Adam: Combines momentum and adaptive learning rates. Efficient and widely used.

- RMSprop: Scales learning rate based on moving average of squared gradients. Works well for

non-stationary problems.

Role: Optimizers accelerate convergence and improve accuracy.

* **Use NumPy, Matplotlib, and Tensorflow/Keras for implementation.**

**Question 6:** Write a Python program to implement a single-layer perceptron from scratch using NumPy to solve the logical AND gate.

(*Include your Python code and output in the code box below.*)

**Answer:**

Python implementation of a Perceptron for AND gate:

import numpy as np

# Inputs and outputs

X = np.array([[0,0],[0,1],[1,0],[1,1]])

y = np.array([0,0,0,1])

# Initialize weights and bias

weights = np.zeros(2)

bias = 0

lr = 0.1

# Training

for epoch in range(10):

for i in range(len(X)):

linear = np.dot(X[i], weights) + bias

pred = 1 if linear > 0 else 0

error = y[i] - pred

weights += lr \* error \* X[i]

bias += lr \* error

print("Trained weights:", weights)

print("Trained bias:", bias)

**Question 7**: Implement and visualize Sigmoid, ReLU, and Tanh activation functions using Matplotlib.

(*Include your Python code and output in the code box below.*)

**Answer:**

Visualization of activation functions:

import numpy as np

import matplotlib.pyplot as plt

x = np.linspace(-10,10,100)

sigmoid = 1/(1+np.exp(-x))

relu = np.maximum(0,x)

tanh = np.tanh(x)

plt.plot(x, sigmoid, label='Sigmoid')

plt.plot(x, relu, label='ReLU')

plt.plot(x, tanh, label='Tanh')

plt.legend()

plt.show()

**Question 8**: Use Keras to build and train a simple multilayer neural network on the MNIST digits dataset. Print the training accuracy.

(*Include your Python code and output in the code box below.*)

**Answer:**

Keras implementation on MNIST:

import tensorflow as tf

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

(x\_train,y\_train),(x\_test,y\_test)=mnist.load\_data()

x\_train, x\_test = x\_train/255.0, x\_test/255.0

model = Sequential([Flatten(input\_shape=(28,28)),

Dense(128, activation='relu'),

Dense(10, activation='softmax')])

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

model.fit(x\_train,y\_train,epochs=5,validation\_data=(x\_test,y\_test))

print("Training complete")

**Question 9**: Visualize the loss and accuracy curves for a neural network model trained on the Fashion MNIST dataset. Interpret the training behavior.

(*Include your Python code and output in the code box below.*)

**Answer:**

Visualize training curves for Fashion MNIST:

import matplotlib.pyplot as plt

history = model.fit(x\_train,y\_train,epochs=5,validation\_data=(x\_test,y\_test))

plt.plot(history.history['loss'], label='loss')

plt.plot(history.history['val\_loss'], label='val\_loss')

plt.plot(history.history['accuracy'], label='accuracy')

plt.plot(history.history['val\_accuracy'], label='val\_accuracy')

plt.legend()

plt.show()

Interpretation:

- If training loss decreases while validation loss increases, overfitting occurs.

- Parallel trends indicate good generalization.

**Question 10:** You are working on a project for a bank that wants to automatically detect fraudulent transactions. The dataset is large, imbalanced, and contains structured features like transaction amount, merchant ID, and customer location. The goal is to classify each transaction as fraudulent or legitimate.

Explain your real-time data science workflow:

* How would you design a deep learning model (perceptron or multilayer NN)?
* Which activation function and loss function would you use, and why?
* How would you train and evaluate the model, considering class imbalance?
* Which optimizer would be suitable, and how would you prevent overfitting?

(*Include your Python code and output in the code box below.*)

**Answer:**

Fraud Detection Workflow:

- Model Design: Use a multilayer neural network with input features (transaction amount, merchant

ID, location).

- Activation Function: ReLU in hidden layers for efficiency; Sigmoid in output for binary

classification.

- Loss Function: Binary Cross-Entropy (suited for classification).

- Handling Class Imbalance: Use resampling techniques, class weights, or SMOTE.

- Optimizer: Adam for fast convergence.

- Prevent Overfitting: Apply dropout, early stopping, and regularization.

This approach ensures balanced learning and robust fraud detection in real-time.