**Practical Number 10**

**Aim:** Create a Multilayer Perceptron basic model in python and train for any data

**Software Used :** Pycharm Community Edition 2023.1, Python(3.9.12)

**Theory:**

Multi-Layer perceptron defines the most complex architecture of artificial neural networks. It is substantially formed from multiple layers of the perceptron. TensorFlow is a very popular deep learning framework released by, and this notebook will guide to build a neural network with this library. If we want to understand what is a Multi-layer perceptron, we have to develop a multi-layer perceptron from scratch using Numpy..

**Code:**

import numpy as np

# Define the sigmoid activation function

def sigmoid(x):

return 1 / (1 + np.exp(-x))

# Define the derivative of the sigmoid activation function

def sigmoid\_derivative(x):

return x \* (1 - x)

# Define the mean squared error loss function

def mean\_squared\_error(y\_pred, y\_true):

return ((y\_pred - y\_true)\*\*2).mean()

# Define the minimal dataset

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

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

# Define the weights and biases for the hidden layer and the output layer

weights\_hidden = np.random.rand(3, 3)

weights\_output = np.random.rand(3, 1)

bias\_hidden = np.zeros((1, 3))

bias\_output = np.zeros((1, 1))

# Train the MLP for 1000 epochs

for epoch in range(1000):

# Forward pass

layer\_hidden = np.dot(X, weights\_hidden) + bias\_hidden

activation\_hidden = sigmoid(layer\_hidden)

layer\_output = np.dot(activation\_hidden, weights\_output) + bias\_output

activation\_output = sigmoid(layer\_output)

# Compute the mean squared error

loss = mean\_squared\_error(activation\_output, y)

# Backward pass

error\_output = activation\_output - y

derivative\_output = sigmoid\_derivative(activation\_output)

delta\_output = error\_output \* derivative\_output

error\_hidden = delta\_output.dot(weights\_output.T)

derivative\_hidden = sigmoid\_derivative(activation\_hidden)

delta\_hidden = error\_hidden \* derivative\_hidden

# Update the weights and biases

weights\_output -= activation\_hidden.T.dot(delta\_output) \* 0.1

bias\_output -= np.sum(delta\_output, axis=0, keepdims=True) \* 0.1

weights\_hidden -= X.T.dot(delta\_hidden) \* 0.1

bias\_hidden -= np.sum(delta\_hidden, axis=0, keepdims=True) \* 0.1

if epoch % 100 == 0:

print(f'Epoch: {epoch}, Loss: {loss}')

# Test the MLP with a new input

x\_test = np.array([1, 0, 0])

layer\_hidden = np.dot(x\_test, weights\_hidden) + bias\_hidden

activation\_hidden = sigmoid(layer\_hidden)

layer\_output = np.dot(activation\_hidden, weights\_output) + bias\_output

activation\_output = sigmoid(layer\_output)

print(f'Predicted output: {activation\_output}')

**Results:**

Epoch: 0, Loss: 0.30115693022879086

Epoch: 100, Loss: 0.24945620326306622

Epoch: 200, Loss: 0.24936287748226743

Epoch: 300, Loss: 0.24925445206218427

Epoch: 400, Loss: 0.24912701144344526

Epoch: 500, Loss: 0.24897671249355646

Epoch: 600, Loss: 0.24879893238624654

Epoch: 700, Loss: 0.24858812113283157

Epoch: 800, Loss: 0.24833763087935687

Epoch: 900, Loss: 0.24803951743278813

Predicted output: [[0.49164764]]

**Conclusion** : In this practical, we have successfully studied and implemented a Multilayer Perceptron basic model in python and train for any data