16. implement Feed forward neural Network

import numpy as np

# Activation function (Sigmoid) and its derivative

def sigmoid(x):

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

def sigmoid\_derivative(x):

return x \* (1 - x) # derivative of sigmoid

# Feed Forward Neural Network

class FeedForwardNN:

def \_\_init\_\_(self, input\_size, hidden\_size, output\_size, learning\_rate=0.1):

# Initialize weights and biases

self.W1 = np.random.uniform(-1, 1, (input\_size, hidden\_size))

self.b1 = np.zeros((1, hidden\_size))

self.W2 = np.random.uniform(-1, 1, (hidden\_size, output\_size))

self.b2 = np.zeros((1, output\_size))

self.lr = learning\_rate

def forward(self, X):

# Input to hidden layer

self.z1 = np.dot(X, self.W1) + self.b1

self.a1 = sigmoid(self.z1)

# Hidden to output layer

self.z2 = np.dot(self.a1, self.W2) + self.b2

self.a2 = sigmoid(self.z2)

return self.a2

def backward(self, X, y, output):

# Error in output

error = y - output

d\_output = error \* sigmoid\_derivative(output)

# Error for hidden layer

error\_hidden = d\_output.dot(self.W2.T)

d\_hidden = error\_hidden \* sigmoid\_derivative(self.a1)

# Update weights and biases

self.W2 += self.a1.T.dot(d\_output) \* self.lr

self.b2 += np.sum(d\_output, axis=0, keepdims=True) \* self.lr

self.W1 += X.T.dot(d\_hidden) \* self.lr

self.b1 += np.sum(d\_hidden, axis=0, keepdims=True) \* self.lr

def train(self, X, y, epochs=10000):

for i in range(epochs):

output = self.forward(X)

self.backward(X, y, output)

if i % 1000 == 0:

loss = np.mean((y - output) \*\* 2)

print(f"Epoch {i}, Loss: {loss:.4f}")

# Example: XOR problem

if \_\_name\_\_ == "\_\_main\_\_":

# XOR inputs and outputs

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

[0,1],

[1,0],

[1,1]])

y = np.array([[0],

[1],

[1],

[0]])

# Create neural network: 2 inputs, 2 hidden neurons, 1 output

nn = FeedForwardNN(input\_size=2, hidden\_size=2, output\_size=1, learning\_rate=0.5)

# Train the network

nn.train(X, y, epochs=10000)

# Test predictions

print("\nPredictions after training:")

for inp in X:

print(f"{inp} -> {nn.forward(inp)}")

output:

