Prac7

import numpy as np

# Activation function and its derivative

def sigmoid(x):

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

def sigmoid\_derivative(x):

return x \* (1 - x)

class XORNeuralNetwork:

def \_\_init\_\_(self, input\_neurons, hidden\_neurons, output\_neurons):

# Initialize network architecture

self.input\_neurons = input\_neurons

self.hidden\_neurons = hidden\_neurons

self.output\_neurons = output\_neurons

# Initialize weights and biases

self.weights\_input\_hidden = np.random.uniform(size=(input\_neurons, hidden\_neurons))

self.weights\_hidden\_output = np.random.uniform(size=(hidden\_neurons, output\_neurons))

self.bias\_hidden = np.random.uniform(size=(1, hidden\_neurons))

self.bias\_output = np.random.uniform(size=(1, output\_neurons))

def forward\_propagation(self, inputs):

# Calculate hidden layer activation

self.hidden\_layer\_input = np.dot(inputs, self.weights\_input\_hidden) + self.bias\_hidden

self.hidden\_layer\_output = sigmoid(self.hidden\_layer\_input)

# Calculate output layer activation

self.output\_layer\_input = np.dot(self.hidden\_layer\_output, self.weights\_hidden\_output) + self.bias\_output

self.predicted\_output = sigmoid(self.output\_layer\_input)

return self.predicted\_output

def backward\_propagation(self, inputs, actual\_output, learning\_rate):

# Calculate error

error = actual\_output - self.predicted\_output

# Calculate output layer gradient

output\_gradient = error \* sigmoid\_derivative(self.predicted\_output)

# Propagate error back to hidden layer

hidden\_error = output\_gradient.dot(self.weights\_hidden\_output.T)

hidden\_gradient = hidden\_error \* sigmoid\_derivative(self.hidden\_layer\_output)

# Update weights and biases

self.weights\_hidden\_output += self.hidden\_layer\_output.T.dot(output\_gradient) \* learning\_rate

self.weights\_input\_hidden += inputs.T.dot(hidden\_gradient) \* learning\_rate

self.bias\_output += np.sum(output\_gradient, axis=0, keepdims=True) \* learning\_rate

self.bias\_hidden += np.sum(hidden\_gradient, axis=0, keepdims=True) \* learning\_rate

return np.mean(np.abs(error)) # return loss

def train(self, inputs, outputs, epochs, learning\_rate):

for epoch in range(epochs):

self.forward\_propagation(inputs)

loss = self.backward\_propagation(inputs, outputs, learning\_rate)

if epoch % 1000 == 0:

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

return self.predicted\_output

# XOR input and output

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

[0, 1],

[1, 0],

[1, 1]])

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

# Network architecture

input\_neurons = inputs.shape[1]

hidden\_neurons = 4

output\_neurons = 1

# Create and train the neural network

nn = XORNeuralNetwork(input\_neurons, hidden\_neurons, output\_neurons)

final\_output = nn.train(inputs, outputs, epochs=10000, learning\_rate=0.1)

print("\nFinal predicted outputs:")

print(final\_output)