**AML Algorithm #14 : back propogation neural network implementation**

import numpy as np

# Sigmoid activation function and its derivative

def sigmoid(x):

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

def sigmoid\_derivative(x):

return x \* (1 - x)

# Define the neural network architecture

class NeuralNetwork:

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

self.input\_size = input\_size

self.hidden\_size = hidden\_size

self.output\_size = output\_size

# Initialize weights and biases

self.weights\_input\_hidden = np.random.rand(self.input\_size, self.hidden\_size)

self.weights\_hidden\_output = np.random.rand(self.hidden\_size, self.output\_size)

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

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

def forward(self, inputs):

# Forward pass

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

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

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

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

return self.output\_layer\_output

def backward(self, inputs, targets, learning\_rate):

# Backward pass

error = targets - self.output\_layer\_output

output\_delta = error \* sigmoid\_derivative(self.output\_layer\_output)

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

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

# Update weights and biases

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

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

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

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

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

for epoch in range(epochs):

for i in range(len(inputs)):

input\_sample = np.array([inputs[i]])

target\_sample = np.array([targets[i]])

# Forward and backward pass

output = self.forward(input\_sample)

self.backward(input\_sample, target\_sample, learning\_rate)

if epoch % 1000 == 0:

error = np.mean(np.square(targets - self.forward(inputs)))

print(f"Epoch {epoch}, Error: {error}")

# Example usage for XOR problem

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

# XOR input and output

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

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

# Initialize and train the neural network

nn = NeuralNetwork(input\_size=2, hidden\_size=4, output\_size=1)

nn.train(inputs, targets, epochs=10000, learning\_rate=0.1)

# Test the trained model

predictions = nn.forward(inputs)

print("Predictions:")

print(predictions)

**OUTPUT :**

Epoch 0, Error: 0.31612137491303927

Epoch 1000, Error: 0.24945301996634694

Epoch 2000, Error: 0.24566375890883224

Epoch 3000, Error: 0.2099654810357422

Epoch 4000, Error: 0.12630196192973245

Epoch 5000, Error: 0.04300009251937542

Epoch 6000, Error: 0.01665936158260744

Epoch 7000, Error: 0.009040353061010255

Epoch 8000, Error: 0.0059113982203200036

Epoch 9000, Error: 0.0042922312111392386

Predictions:

[[0.06480613]

[0.94556391]

[0.9457127 ]

[0.05656011]]