 Assignment 2: Write a program to show back propagation network for XOR function with binary input

# and output.

# Name: Adwait Nitin Deshpande

# Class: TE-AIML

# Roll No: 33514

import numpy as np

# Define the sigmoid activation function and its derivative

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

    return sigmoid(x) \* (1 - sigmoid(x))

# Define the XOR function

def xor(inputs):

    return np.array([int(inputs[0] != inputs[1])])

# Define the input and target data

input\_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1],[0, 0], [0, 1], [1, 0], [1, 1]])

target\_data = np.array([[0], [1], [1], [0], [0], [1], [1], [0]])

# Define the neural network architecture

input\_size = 2

hidden\_size = 8

output\_size = 1

# Initialize the weights with random values

hidden\_weights = np.random.uniform(size=(input\_size, hidden\_size))

output\_weights = np.random.uniform(size=(hidden\_size, output\_size))

# Define the learning rate and number of epochs

learning\_rate = 0.1

epochs = 100000

# Train the neural network using backpropagation

for epoch in range(epochs):

    # Forward propagation

    hidden\_layer = sigmoid(np.dot(input\_data, hidden\_weights))

    output\_layer = sigmoid(np.dot(hidden\_layer, output\_weights))

    # Backward propagation

    output\_error = target\_data - output\_layer

    output\_delta = output\_error \* sigmoid\_derivative(output\_layer)

    hidden\_error = output\_delta.dot(output\_weights.T)

    hidden\_delta = hidden\_error \* sigmoid\_derivative(hidden\_layer)

    # Update the weights|

    output\_weights += hidden\_layer.T.dot(output\_delta) \* learning\_rate

    hidden\_weights += input\_data.T.dot(hidden\_delta) \* learning\_rate

# Test the neural network on new input data

test\_input = np.array([[1, 0], [0, 1], [1, 1], [0, 0]])

for i in range(len(test\_input)):

    prediction = sigmoid(np.dot(sigmoid(np.dot(test\_input[i], hidden\_weights)), output\_weights))

    print(f"Input: {test\_input[i]} Output: {prediction.round()} Target: {xor(test\_input[i])}")

Input: [1 0] Output: [1.] Target: [1]

Input: [0 1] Output: [1.] Target: [1]

Input: [1 1] Output: [0.] Target: [0]

Input: [0 0] Output: [0.] Target: [0]

 This is a Python code for a neural network that learns to solve the XOR problem, which is a classic problem in the field of artificial intelligence. The XOR function returns a 1 if the inputs are different, and 0 if they are the same. The XOR problem is challenging for traditional computing approaches because it is not linearly separable.

# Here is an explanation of the code:

# The numpy library is imported, which is used for matrix calculations.

# Two functions are defined: sigmoid() and sigmoid\_derivative(). The sigmoid function is used as the activation function for the neurons in the network, while the sigmoid\_derivative function is used to calculate the gradient during backpropagation.

# The XOR function is defined, which takes in two inputs and returns a binary output of 0 or 1 depending on whether the inputs are equal or different.

# The input and target data are defined. The input\_data is a 2D numpy array consisting of all possible inputs for the XOR function. The target\_data is a 2D numpy array consisting of the corresponding outputs for the XOR function.

# The neural network architecture is defined by specifying the number of input neurons, hidden neurons, and output neurons.

# The weights for the neural network are initialized with random values using the np.random.uniform() function. The hidden\_weights and output\_weights are numpy arrays with dimensions (input\_size, hidden\_size) and (hidden\_size, output\_size) respectively.

# The learning rate and number of epochs are defined. The learning rate determines how much the weights are adjusted during each iteration of training. The epochs determine how many times the entire dataset is used to train the neural network.

# The neural network is trained using backpropagation by iterating through each epoch.

# During each epoch, the forward propagation is performed, where the input data is multiplied by the hidden\_weights to get the hidden\_layer output, and the hidden\_layer output is multiplied by the output\_weights to get the output\_layer output. The sigmoid activation function is applied to both the hidden\_layer and output\_layer outputs.

# During backpropagation, the output\_error is calculated by subtracting the target\_data from the output\_layer. The output\_delta is calculated by multiplying the output\_error with the sigmoid\_derivative of the output\_layer. The hidden\_error is calculated by multiplying the output\_delta with the transpose of the output\_weights. The hidden\_delta is calculated by multiplying the hidden\_error with the sigmoid\_derivative of the hidden\_layer.

# The weights are updated using the learning rate and the delta values. The output\_weights are updated by multiplying the transpose of the hidden\_layer with the output\_delta. The hidden\_weights are updated by multiplying the transpose of the input\_data with the hidden\_delta.

# After training, the neural network is tested on new input data by iterating through each input and calculating the output using the sigmoid activation function and the learned weights. The prediction is rounded to the nearest integer, and the actual target output is calculated using the XOR function. The input, prediction, and target output are printed for each input.

# This code implements a neural network in Python that uses backpropagation to learn how to solve the XOR problem. The neural network architecture consists of an input layer, a hidden layer, and an output layer. The weights for the neural network are initialized with random values, and the network is trained using a learning rate and a specified number of epochs. After training, the neural network is tested on new input data, and the input, prediction, and target output are printed for each input.