# **Aim:** Write a python Program for back propagation algorithm.

import numpy as np

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

    return x \* (1 - x)

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

        self.learning\_rate = 0.1

        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)

    def feedforward(self, X):

        self.hidden\_layer\_input = np.dot(X, self.weights\_input\_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.output\_layer\_output = sigmoid(self.output\_layer\_input)

    def backward(self, X, y):

        self.error = y - self.output\_layer\_output

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

        self.hidden\_layer\_error = delta\_output.dot(self.weights\_hidden\_output.T)

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

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

        self.weights\_input\_hidden += X.T.dot(delta\_hidden) \* self.learning\_rate

    def train(self, X, y, epochs):

        for \_ in range(epochs):

            self.feedforward(X)

            self.backward(X, y)

    def predict(self, X):

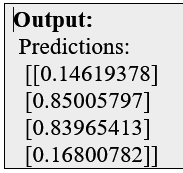
        self.feedforward(X)

        return self.output\_layer\_output

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

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

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

    neural\_network = NeuralNetwork(2, 4, 1)

    neural\_network.train(X, y, epochs=10000)

    predictions = neural\_network.predict(X)

    print("Predictions:")

    print(predictions)

# **Aim:** Write a python Program for error back propagation algorithm.

import numpy as np

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

    return x \* (1 - x)

class NeuralNetwork:

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

        self.input\_size = input\_size

        self.hidden\_size = hidden\_size

        self.output\_size = output\_size

        self.learning\_rate = learning\_rate

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

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

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

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

    def forward(self, x):

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

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

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

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

    def backward(self, x, y):

        loss = y - self.output

        delta\_output = loss \* sigmoid\_derivative(self.output)

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

        delta\_hidden = hidden\_error \* sigmoid\_derivative(self.hidden\_output)

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

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

        self.weights\_input\_hidden += x.T.dot(delta\_hidden) \* self.learning\_rate

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

    def train(self, x, y, epochs):

        for \_ in range(epochs):

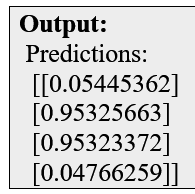
            self.forward(x)

            self.backward(x, y)

    def predict(self, x):

        self.forward(x)

        return self.output

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

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

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

    neural\_network = NeuralNetwork(input\_size=2,hidden\_size=4,

output\_size=1, learning\_rate=0.1)

    neural\_network.train(X, y, epochs=10000)

    predictions = neural\_network.predict(X)