```
import numpy as np


class NeuralNetwork:
    def __init__(self):

        self.input_size = 3
        self.hidden_size1 = 2
        self.hidden_size2 = 2
        self.output_size = 1


        self.W1 = np.random.randn(self.hidden_size1, self.input_size)
        self.b1 = np.zeros((self.hidden_size1, 1))
        self.W2 = np.random.randn(self.hidden_size2, self.hidden_size1)
        self.b2 = np.zeros((self.hidden_size2, 1))
        self.W3 = np.random.randn(self.output_size, self.hidden_size2)
        self.b3 = np.zeros((self.output_size, 1))


        self.lr = 0.01

    def sigmoid(self, Z):
        return 1 / (1 + np.exp(-Z))

    def sigmoid_derivative(self, Z):
        return self.sigmoid(Z) * (1 - self.sigmoid(Z))

    def forward(self, X):

        self.Z1 = np.dot(self.W1, X.T) + self.b1
        self.A1 = self.sigmoid(self.Z1)
        self.Z2 = np.dot(self.W2, self.A1) + self.b2
        self.A2 = self.sigmoid(self.Z2)
        self.Z3 = np.dot(self.W3, self.A2) + self.b3
        self.A3 = self.sigmoid(self.Z3)
        return self.A3

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

        m = X.shape[1]

        dZ3 = output - Y
        dW3 = 1 / m * np.dot(dZ3, self.A2.T)
        db3 = 1 / m * np.sum(dZ3, axis=1, keepdims=True)

        dA2 = np.dot(self.W3.T, dZ3)
        dZ2 = dA2 * self.sigmoid_derivative(self.Z2)
        dW2 = 1 / m * np.dot(dZ2, self.A1.T)
        db2 = 1 / m * np.sum(dZ2, axis=1, keepdims=True)

        dA1 = np.dot(self.W2.T, dZ2)
        dZ1 = dA1 * self.sigmoid_derivative(self.Z1)
        dW1 = 1 / m * np.dot(dZ1, X)
        db1 = 1 / m * np.sum(dZ1, axis=1, keepdims=True)


        self.W1 -= self.lr * dW1
        self.b1 -= self.lr * db1
        self.W2 -= self.lr * dW2
        self.b2 -= self.lr * db2
        self.W3 -= self.lr * dW3
        self.b3 -= self.lr * db3

    def train(self, X, Y, epochs):
        for epoch in range(epochs):

            output = self.forward(X)


            self.backward(X, Y, output)


            if epoch % 100 == 0:
                loss = np.mean(np.square(output - Y))
                print(f'Epoch {epoch+1}, Loss: {loss}')



X = np.array([[1, 1, 8],
              [6, 5, 7]])
Y = np.array([[1, 1]])
```

```
X = X / np.max(X)


model = NeuralNetwork()


model.train(X, Y, epochs=1)
```

```
Epoch 1, Loss: 0.08513999601482586
```