CODE:

# Assignment 4: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using  
# appropriate data sets.  
  
  
from csv import reader  
from random import random  
from math import exp  
  
  
def sigmoid(x): # https://en.wikipedia.org/wiki/Sigmoid\_function  
 *"""Activation function."""* return **1** / (**1** + exp(-x))  
  
  
with open('dataset.csv') as csv\_file:  
 dataset = reader(csv\_file)  
  
 print('\nAttributes:'**,** \*next(dataset)**,** '\n')  
 lr = input('Learning Rate (b/w 0 and 1; leave empty for default (.5)): ')  
 if lr:  
 lr = float(lr)  
 else:  
 lr = **.5** print(lr)  
 print('\n')  
  
 for num**,** data in enumerate(dataset**,** start=**1**):  
  
 print('DATA'**,** num)  
 i1**,** i2**,** b1**,** b2**,** to1**,** to2 = (float(i) for i in data)  
 print('input 1:'**,** i1)  
 print('input 2:'**,** i2)  
 print('bias 1:'**,** b1)  
 print('bias 2:'**,** b2)  
 print('target output 1:'**,** to1)  
 print('target output 2:'**,** to2)  
 print()  
  
 w = [None**,** \*[random() for \_ in range(**8**)]] # [None, .15, .2, .25, .3, .4, .45, .5, .55]  
 print('Initial weights (random) w1 to w8:'**,** w[**1**:])  
  
 back\_props = **0** while True: # will back-prop till error is not minimized  
  
 h1 = sigmoid(i1 \* w[**1**] + i2 \* w[**2**] + b1)  
 h2 = sigmoid(i1 \* w[**3**] + i2 \* w[**4**] + b1)  
 po1 = sigmoid(h1 \* w[**5**] + h2 \* w[**6**] + b2)  
 po2 = sigmoid(h1 \* w[**7**] + h2 \* w[**8**] + b2)  
 error = **.5** \* ((to1 - po1) \*\* **2** + (to2 - po2) \*\* **2**)  
  
 print('hidden node 1:'**,** h1)  
 print('hidden node 2:'**,** h2)  
 print('predicted output 1:'**,** po1)  
 print('predicted output 2:'**,** po2)  
 print('Error:'**,** error**,** '\n')  
  
 if error: # back-propagate  
 back\_props += **1** w5 = w[**5**] - (lr \* ((po1-to1) \* (po1\*(**1**-po1)) \* h1))  
 w6 = w[**6**] - (lr \* ((po1-to1) \* (po1\*(**1**-po1)) \* h2))  
 w7 = w[**7**] - (lr \* ((po2-to2) \* (po2\*(**1**-po2)) \* h1))  
 w8 = w[**8**] - (lr \* ((po2-to2) \* (po2\*(**1**-po2)) \* h2))  
  
 w[**1**] = w[**1**] - (lr \* (((po1-to1)\*(po1\*(**1**-po1))\*w[**5**]) \* (h1\*(**1**-h1)) \* i1))  
 w[**2**] = w[**2**] - (lr \* (((po1-to1)\*(po1\*(**1**-po1))\*w[**6**]) \* (h1\*(**1**-h1)) \* i2))  
 w[**3**] = w[**3**] - (lr \* (((po2-to2)\*(po2\*(**1**-po2))\*w[**7**]) \* (h2\*(**1**-h2)) \* i1))  
 w[**4**] = w[**4**] - (lr \* (((po2-to2)\*(po2\*(**1**-po2))\*w[**8**]) \* (h2\*(**1**-h2)) \* i2))  
  
 w[**5**]**,** w[**6**]**,** w[**7**]**,** w[**8**] = w5**,** w6**,** w7**,** w8 # now updating the weights  
  
 # Formula's references:  
 # https://youtu.be/QZ8ieXZVjuE  
 # https://youtu.be/GJXKOrqZauk  
 # https://youtu.be/0e0z28wAWfg  
 # https://en.wikipedia.org/wiki/Backpropagation#Finding\_the\_derivative\_of\_the\_error  
  
 if '-' not in str(error):  
 print(f'Weights after {back\_props} back-propagations (w1 to w8):'**,** w[**1**:])  
 else: # stopping when error reduces to 0.0000x  
 break  
  
 else: # desired output received  
 break

OUTPUT:

Attributes: i1 i2 b1 b2 o1 o2

Learning Rate (b/w 0 and 1; leave empty for default (.5)): 1

DATA 1

input 1: 0.05

input 2: 0.1

bias 1: 0.35

bias 2: 0.6

target output 1: 0.01

target output 2: 0.99

Initial weights (random) w1 to w8: [0.9137277288548736, 0.02533089863097593, 0.018060319724300777, 0.9060079942807503, 0.32138324223435943, 0.390791993737296, 0.3759603314150709, 0.6851702875805843]

hidden node 1: 0.5982597950923112

hidden node 2: 0.6086173014621932

predicted output 1: 0.7369378882058164

predicted output 2: 0.7758995091754513

Error: 0.2871388567402224

Weights after 2757 back-propagations (w1 to w8): [1.0466634253519709, 0.285317041418153, 0.07969868070192626, 1.0570137486462006, -3.687754814829549, -3.679706354807677, 2.530871469899533, 2.870590641751935]

hidden node 1: 0.6060801737715001

hidden node 2: 0.612939757253339

predicted output 1: 0.02002563130012471

predicted output 2: 0.9800311373560595

Error: 9.994575268989611e-05