QUESTION 3

Write a program for implementing BPN for training a single hidden layer back-propagation network with bipolar sigmoidal units (x = 1) to achieve the following mapping: y = sin(π x1) + cos(0.2 π x2) Set up two sets of data, each consisting of 20 input-output pairs, one for training and other for testing. The input-output data are obtained by varying input variables (x1, x2) within [-1, + 1] randomly. Also the output data is normalized within [-1, +1]. Apply training to find proper weights in the network.

In [1]:

```
#importing pkgs
import random
import numpy as np
```

In [2]:

```
import math
def output(x1,x2):
    output = math.sin(math.pi*x1)+math.cos(math.pi*0.2*x2)
    return output

#generating trainig dataset
train = []
for i in range(20):
    x1 = random.uniform(-1,1)
    x2 = random.uniform(-1,1)
    y = output(x1,x2)
    train.append([x1,x2,y])

#train data standardizAtion
max1,min1 = -2,2
for t in train:
    max1 = max(max1,t[2])
    min1 = min(min1,t[2])
diff = max1-min1
for i in range(20):
    train[i][2] = 1 - (2*(max1 - train[i][2])/diff)
```

In [3]:

```
#generating test data
test = []
for i in range(20):
    x1 = random.uniform(-1,1)
    x2 = random.uniform(-1,1)
    y = output(x1,x2)
    test.append([x1,x2,y])

#test data normalization
maxx,minn = -2,2
for t in test:
    maxx = max(maxx,t[2])
    minn = min(minn,t[2])
diff = maxx-minn
for i in range(20):
    test[i][2] = 1 - (2*(maxx - test[i][2])/diff)
```

In [4]:

```
def activation(x):
    val = (1-math.exp(-x))/(1+math.exp(-x))
    return val
```

In [5]:

```
def derivative(x):
    t = activation(x)
    val = ((1 + t)*(1 - t))/2
    return val
```

In [6]:

```
wghts = [[random.uniform(0,1),random.uniform(0,1)],[random.uniform(0,1),random.uniform(0
,1)]]
hidden_weights = [random.uniform(0,1),random.uniform(0,1)]
epochs=15
b0=0.2
b1=0.2
b2=0.2
alpha=0.0001

from operator import add
def BPN(train,wghts,hidden_weights):

    b = [b0,b1,b2]
    T = [t[2] for t in train]

    for l in range(epochs):
        zn,z = [0]*2,[0]*2
        Y = []
        for t in train:

            x1,x2,tk = t[0],t[1],t[2]

            #calculation at hidden layer
            for i in range(2):
                zn[i] = x1*wghts[0][i] + x2*wghts[1][i] + b[i]

            for j in range(2):
                z[j] = activation(zn[j])

            # calculation at output layer
            yin = z[0]*hidden_weights[0] + z[1]*hidden_weights[1] + b2
            y = activation(yin)
            Y.append(y)


            delta0 = (tk - y)*derivative(yin)
            h_corr =[0]*2

            for i in range(2):
                h_corr[i] = alpha*delta0*z[i]
            bias_correlation = [0]*3
            bias_correlation[2] = alpha*delta0

            #error correction between hidden and input layer
            del_in,delta = [0]*2,[0]*2
            for i in range(2):
                del_in[i] = delta0*(wghts[0][i] + wghts[1][i])
                delta[i] = del_in[i]*derivative(zn[i])
            weight_corr = [[0]*2,[0]*2]

            for i in range(2):
                for j in range(2):
                    weight_corr[i][j] = alpha*delta[j]*t[i]
                bias_correlation[i] = alpha*delta[i]

            for i in range(2):
                for j in range(2):
                    wghts[i][j] += weight_corr[i][j]

            b = list(map(add,b,bias_correlation))
            hidden_weights = list(map(add,hidden_weights,h_corr))

        print("Epoch : ",l+1)
```

```
        flag = False
        for i in range(20):
            if T[i] != Y[i]:
                flag = True
                break
        if flag == False:
            break
```

In [7]:

```
BPN(train,wghts,hidden_weights)
```

```
Epoch :  1
Epoch :  2
Epoch :  3
Epoch :  4
Epoch :  5
Epoch :  6
Epoch :  7
Epoch :  8
Epoch :  9
Epoch :  10
Epoch :  11
Epoch :  12
Epoch :  13
Epoch :  14
Epoch :  15
```

In [8]:

```
BPN(test,wghts,hidden_weights)
```

```
Epoch :  1
Epoch :  2
Epoch :  3
Epoch :  4
Epoch :  5
Epoch :  6
Epoch :  7
Epoch :  8
Epoch :  9
Epoch :  10
Epoch :  11
Epoch :  12
Epoch :  13
Epoch :  14
Epoch :  15
```

In [ ]:

```

```

Question 2

Write a program to classify the numbers between 0-9 using Adaline networks.

In [17]:

```
#importing libraries
from keras.datasets import mnist
import numpy as np
```

In [10]:

```
# Importing MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
print("Training data shape: ", x_train.shape)
print("Test data shape", x_test.shape)
```

```
Training data shape:  (60000, 28, 28)
Test data shape (10000, 28, 28)
```

In [11]:

```
#resizing data
image_vector_size = 28*28
x_train = x_train.reshape(x_train.shape[0], image_vector_size)
x_test = x_test.reshape(x_test.shape[0], image_vector_size)
```

In [12]:

```
#data shape
print(x_train.shape)
print(y_train.shape)
```

```
(60000, 784)
(60000,)
```

In [13]:

```
#adaline model

epoch=5
alpha=0.00000001
weight=(np.zeros(784)).reshape(784,1)
bias=(np.zeros(60000)).reshape(60000,1)
avg=0

for i in range(epoch):
    print("Epoch : ",i+1)
    for j in range(x_train.shape[0]):
        #calculation
        y_cal=np.dot(x_train[j].reshape(1,784),weight)+bias[j]

        dif=y_train[j] - y_cal

        #weight updates
        weight=weight + alpha*dif*(x_train[j].reshape(784,1))
        bias[j]=bias[j] + alpha*dif

        avg+=dif*dif

print("AVerage square training error: ",avg[0][0]/x_train.shape[0])
```

```
Epoch :  1
Epoch :  2
Epoch :  3
Epoch :  4
Epoch :  5
AVerage square training error:  18.839162099696313
```

In [16]:

```
test_avg=0
for k in range(x_test.shape[0]):

    y_pred=np.dot(x_test[k].reshape(1,784),weight)+bias[k]
    dif=y_test[k] - y_cal

    test_avg+=dif*dif

print("Average square testing error: ",test_avg[0][0]/x_test.shape[0])
```

```
Average square testing error:  10.763610468911597
```

In [ ]:

```

```

In [ ]:

In [ ]:

```

```