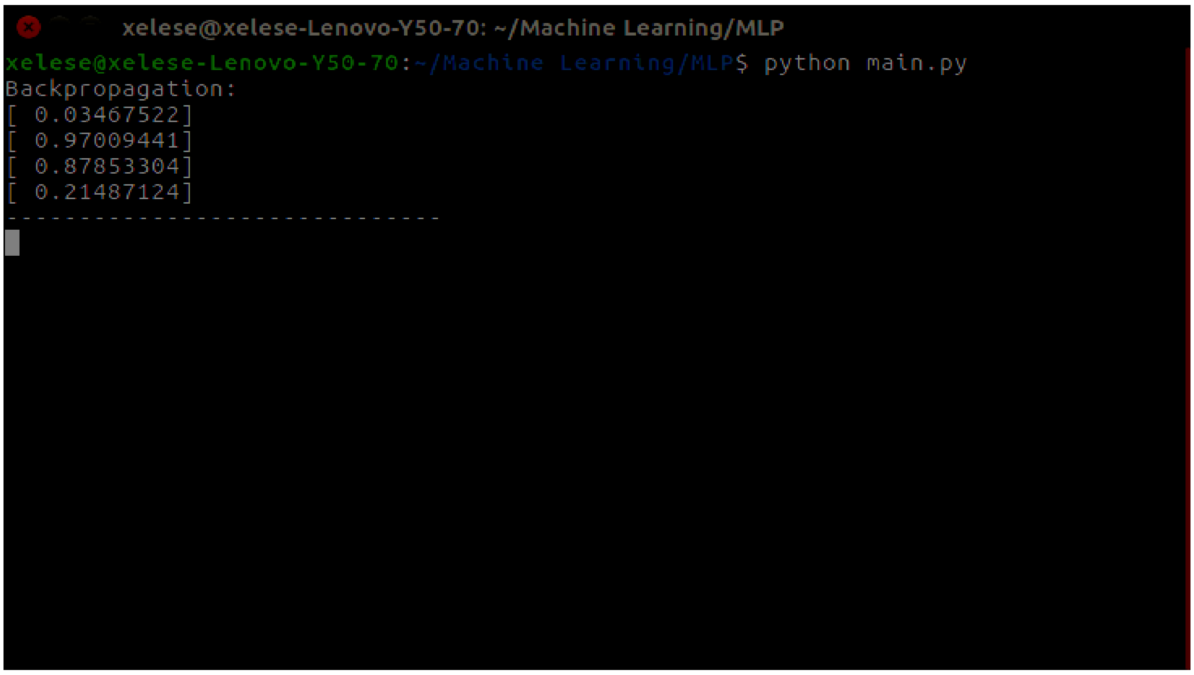
MACHINE LEARINING
LAB ASSESSMENT – III

15BCE0082 VOLETI RAVI

CODE:

```
import mlpModule
import numpy as np
from pylab import meshgrid,cm,imshow,contour,clabel,colorbar,axis,title,show

myMLP = mlpModule.MLP(2,8,1)
myBack = mlpModule.Backpropagation(myMLP, 0.3, 0.001)

print("Backpropagation:")
for i in range(5000):
    myBack.iterate([[0,0],[0,1],[1,0],[1,1], [0.5, 0.5], [0.75, 0.5], [0.3, 0.5], [0.45, 0.2], [0.2, 0.7]],

                [[0],[1],[1],[0],[0],[1],[1],[0],[1]])
print(myMLP.compute([0,0]))
print(myMLP.compute([0,1]))
print(myMLP.compute([1,0]))
print(myMLP.compute([1,1])) #tender a 01 11 11 01
print("------------------------------")

x = np.arange(0,1.0,0.1)
y = np.arange(0,1.0,0.1)
X,Y = np.meshgrid(x, y)
Z = X

for i in range(10) :
    for j in range(10):
        Z[i][j] = myMLP.compute2Arg(X[i][j],Y[i][j])

im = imshow(Z,cmap=cm.RdBu) # drawing the function
cset = contour(Z,np.arange(0,1,0.2),linewidths=2,cmap=cm.Set2)
clabel(cset,inline=True,fmt='%1.1f',fontsize=10)
colorbar(im) # adding the colobar on the right
show()

import numpy as np
import math

class MLP:

    def sigmoid(self, x):
        return float(1 / (1 + math.exp(-x)))

    def __init__(self,nInput,nHidden,nOutput):
        self.WList = [np.random.rand(nHidden, nInput+1), np.random.rand(nOutput, nHidden+1)]
        #self.WList = [[[7.171643527861244, -3.708726348229882, -3.6256393221664],
        #              [-1.5453363462164666,-1212.28078998915,-10.97290255045335]],
        #             [[-6.979517695806317,9.546594522649979,-78.86862110309602]]]
        self.nInput = nInput;
        self.nHidden = nHidden;
        self.nOutput = nOutput;

        self.sigmoid = np.vectorize(self.sigmoid)

    def compute2Arg(self, v1,v2):
        return self.compute([v1,v2])[0]

    def compute(self, inputValues):
        inputValues = np.append([1], inputValues)
        self.lastInput = inputValues;
        self.lastHiddenOutput = np.dot(self.WList[0], np.transpose(inputValues))
        self.lastHiddenOutput = self.sigmoid(self.lastHiddenOutput)
        self.lastHiddenOutput  = np.append([1], self.lastHiddenOutput)
        self.lastOutput = np.dot(self.WList[1], self.lastHiddenOutput)
        self.lastOutput = self.sigmoid(self.lastOutput)
        return self.lastOutput

class Backpropagation:

    def __init__(self, mlp, alpha, regularization):
        self.DeltaList = [np.zeros((mlp.nHidden, mlp.nInput+1)), np.zeros((mlp.nOutput,
mlp.nHidden+1))]

        self.alpha = alpha;
        self.regularization = regularization;
        self.mlp = mlp


    def iterate(self, inputValues, outputValues):
        for i in range(len(inputValues)):
            self.mlp.compute(inputValues[i])
            delta3 = np.subtract( self.mlp.lastOutput, outputValues[i]);

            Od3 = np.dot((self.mlp.WList[1].transpose()),delta3)
            gz2 = np.multiply(self.mlp.lastHiddenOutput, (1-self.mlp.lastHiddenOutput))
            delta2 = np.multiply(Od3, gz2)

            self.DeltaList[1] = np.add(self.DeltaList[1], np.outer(delta3, self.mlp.lastHiddenOutput))
            self.DeltaList[0] = np.add(self.DeltaList[0], np.delete(np.outer(delta2,
self.mlp.lastInput),0,0))

        self.DeltaList[0]   = self.DeltaList[0]/len(inputValues) + self.regularization*self.mlp.WList[0]
        self.DeltaList[1]   = self.DeltaList[1]/len(inputValues) + self.regularization*self.mlp.WList[1]
        self.mlp.WList[0] = self.mlp.WList[0]-self.alpha*self.DeltaList[0]
        self.mlp.WList[1] = self.mlp.WList[1]-self.alpha*self.DeltaList[1]
```

OUTPUT:



```
xelese@xelese-Lenovo-Y50-70: ~/Machine Learning/MLP
xelese@xelese-Lenovo-Y50-70:~/Machine Learning/MLP$ python main.py
Backpropagation:
[ 0.03467522]
[ 0.97009441]
[ 0.87853304]
[ 0.21487124]
------------------------------
```

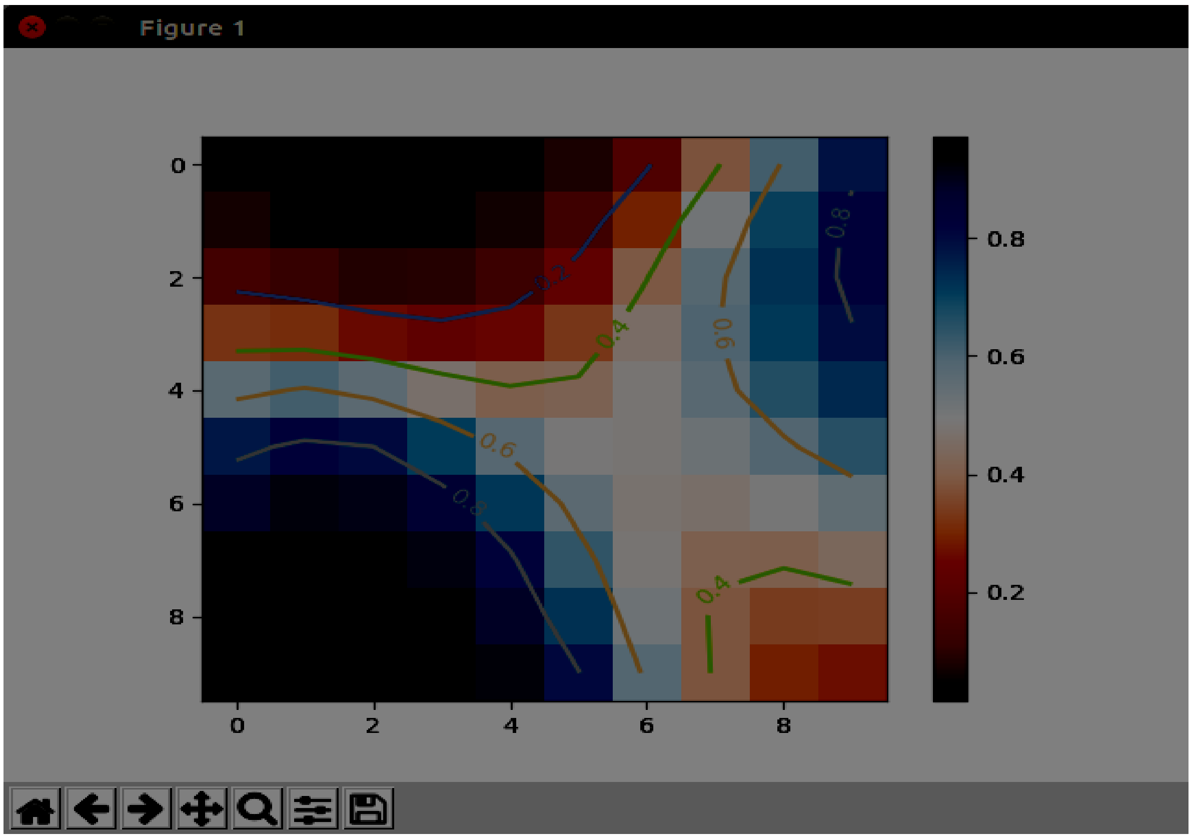


Figure 1