SMAI Assignment 1 Report

201401074

Problem 1

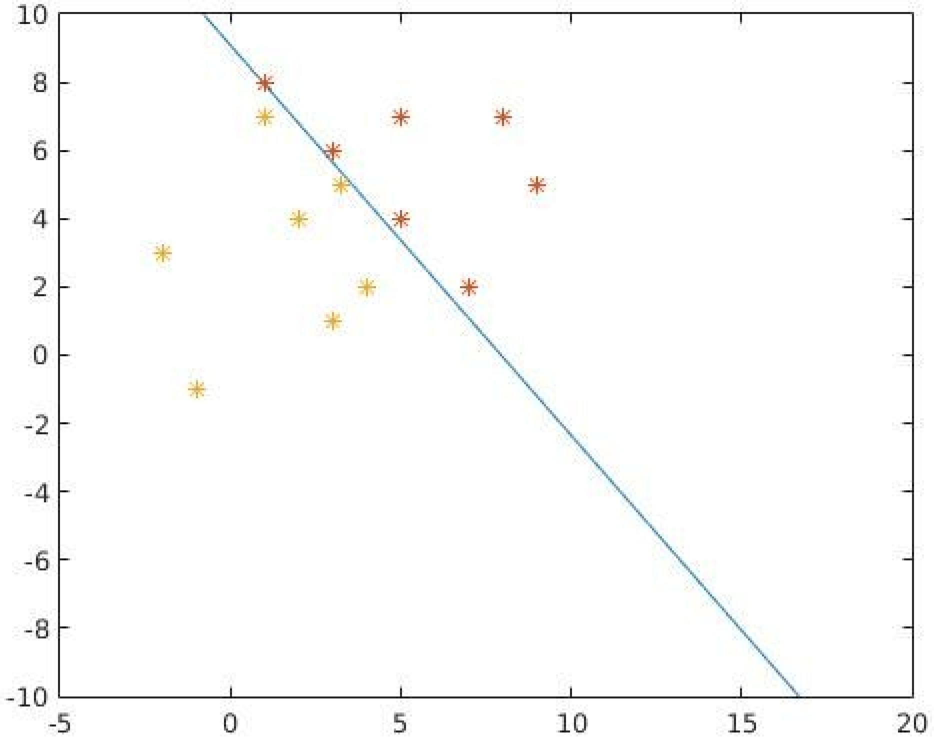
A.

- MATLAB code

```
  %% Single-sample perceptron
 a = rand(3,1);
 w1 = [[2,7];[8,1];[7,5];[6,3];[7,8];[5,9];[4,5]];
 w2 = [[4,2];[-1,-1];[1,3];[3,-2];[5,3.25];[2,4];[7,1]];
 py = [ones(size(w1,1),1),w1];
 ny = [-ones(size(w2,1),1),-w2];
 y = [py;ny]';
 k = 1;
 n = size(y,2);
 it = 0;
 eta = 1;
 while 1
     dist = a'*y;
     if size(find(dist<0),2) == 0
         break
     end
     if a'*y(:,k) < 0
         a = a + eta*y(:,k);
     end
     k = mod(k+1,n);
     if k == 0
         k = 1;
     end
    it = it + 1;
end
disp(it);
```

- There is variation in convergence of algorithm on random initialisation of weight vector
- Eg : a = transpose(1,1,1), converges in 853 iterations
- Eg : a = transpose(0.7209,0.0186,0.6748) converges in 970 iterations
- For random initialisation of a between 0-1 converges in average 1210 iterations
- For random initialisation of a between 0-10 converges in average 2671 iterations

- For random initialisation of a between 0-100 converges in average 6599 iterations
- Learning rate was kept fixed as 1 for above cases
- On decreasing learning rate to 0.01 reduces average convergence of algorithm to 876, 1516, 5842 iterations respectively. (for 0-1, 0-10, 0-100)
- Initialisation the weight vector in and around the dimensions of the data samples helps in convergence. Very high value makes convergence slow on the other hand,a very low value makes the descent unstable in the start.
- Plot : Red - Class 1, Yellow - Class 2



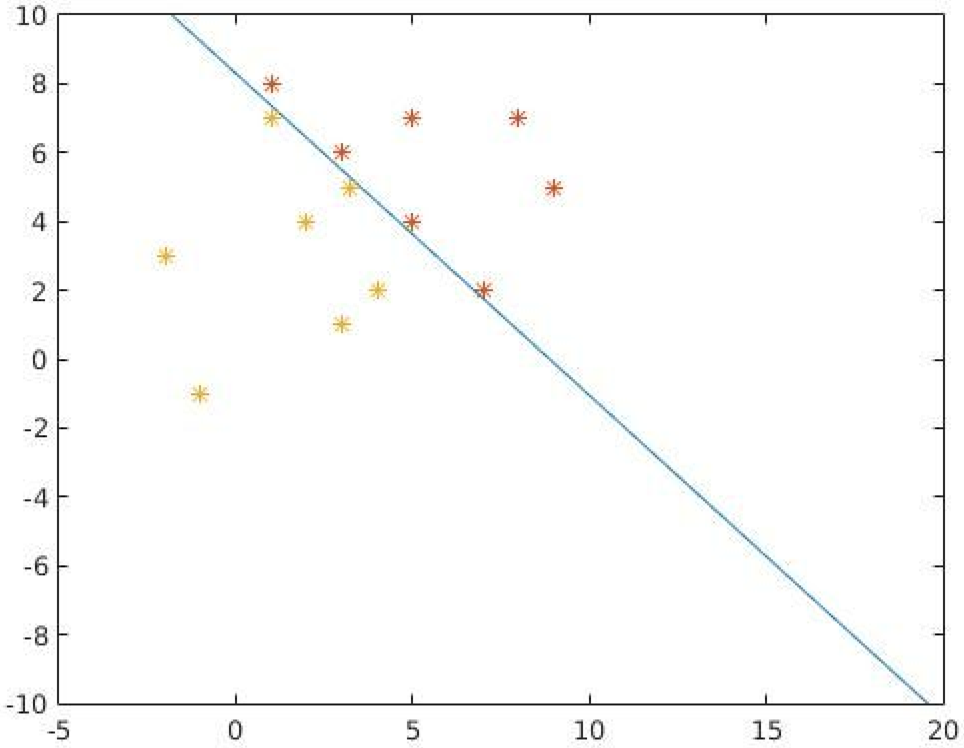
B.

- MATLAB code

```matlab
%% Single-sample perceptron with margin
a = rand(3,1);
w1 = [[2,7];[8,1];[7,5];[6,3];[7,8];[5,9];[4,5]];
w2 = [[4,2];[-1,-1];[1,3];[3,-2];[5,3.25];[2,4];[7,1]];
py = [ones(size(w1,1),1),w1];
ny = [-ones(size(w2,1),1),-w2];
y = [py;ny]';
k = 1;
n = size(y,2);
b = 100;
eta = 1;
while 1
    dist = a'*y;
    if size(find(dist<b),2) == 0
        break
    end
    if a'*y(:,k) < b
        a = a + eta*y(:,k);
    end
    k = mod(k+1,n);
    if k == 0
        k = 1;
    end
end
```

- Effect of convergence of algorithm on the initialisation of weights is similar to that of Single Sample Perceptron case. High values of a leads to slower convergence. Values of a similar ot the test data points leads to a faster convergence.
- Changing the value of the margin keeping the weight vector a the same also varies the iterations needed for algorithm to converge.
- For margin b = 0.01, 296 iterations to converge.
- For margin b = 0.1, 918 iterations to converge.
- For margin b = 1, 3562 iterations to converge.
- For margin b = 10, 20227 iterations to converge.
- For margin b = 100, 287650 iterations to converge.

- Decreasing value of learning rate speeds up the convergence of the algorithm till a point and then the convergence increases, same with increasing the eta value.
- For eta = 1, 515 iterations to converge.
- For eta = 0.1, 1009 iterations to converge.
- For eta = 2, 296 iterations to converge.
- For eta = 3, 749 iterations to converge.
- Plot : Red - Class 1, Yellow - Class 2



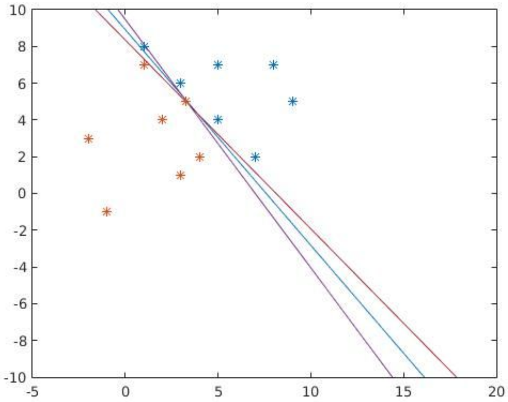
C.

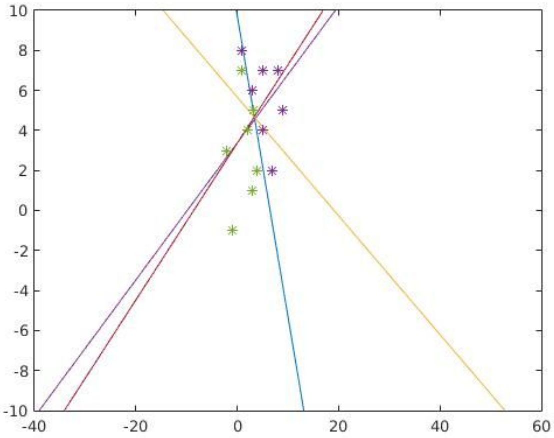
- MATLAB code

```
%% Relaxation algorithm with margin
a = randi([-100 100],3,1);
w1 = [[2,7];[8,1];[7,5];[6,3];[7,8];[5,9];[4,5]];
w2 = [[4,2];[-1,-1];[1,3];[3,-2];[5,3.25];[2,4];[7,1]];
py = [ones(size(w1,1),1),w1];
ny = [-ones(size(w2,1),1),-w2];
y = [py;ny]';
k = 1;
n = size(y,2);
learningRate = 0.01;
b = 10;
theta = 0.00001;
it = 0;
while 1
     dist = a'*y - b;
     ind = find(dist<0);
     miscfd = y(:,ind);
     miscfd_dist = dist(ind);
     miscfd_norm = diag(miscfd'*miscfd)';
     miscfd_err = repmat(miscfd_dist ./ miscfd_norm,size(y,1),1);
     miscfd_err = miscfd_err .* miscfd;
     final_err = sum(miscfd_err')';
     if abs(sum(learningRate * final_err)) < theta
          break
     end
     a = a - learningRate * final_err;
     it = it + 1;
end
disp(it);
```

- In many cases, we do not get a good decision boundary. This is mostly due to the values of the learningRate and the threshold (theta). Sometimes the algorithm breaks out due to high theta value so a proper decision boundary is not found as the algorithm is not let to converge.
- Margin has the same effect on convergence time as discussed above, larger b values considerable slowing down the convergence.

- Smaller value of theta / threshold improve the convergence of the algorithm to find a decent decision boundary but also increase the convergence time considerably.
- Eg : if theta = 0.1,0.01,0.001 there are many outliers but the no. of iterations is <500 but for theta = 0.0001,0.00001,0.000001 we find a very good decision boundary but the no. of iterations is 10629, 80143, 712056 respectively.
- The initialization of the weight vector has similar results on convergence time and giving a good decision boundary as discussed earlier.
- Plot : Red - Class 1, Yellow - Class 2, Fig 1 - Different values of a, Fig 2 - Different values of b





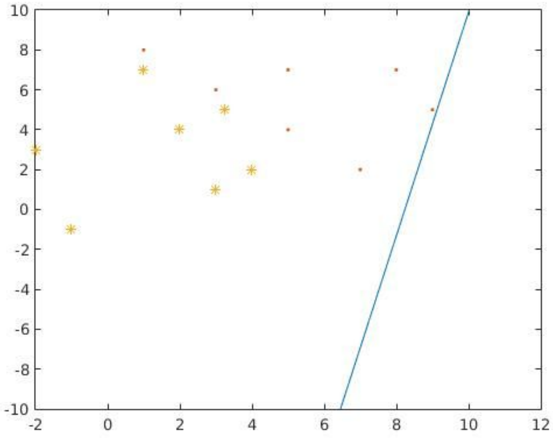
D.

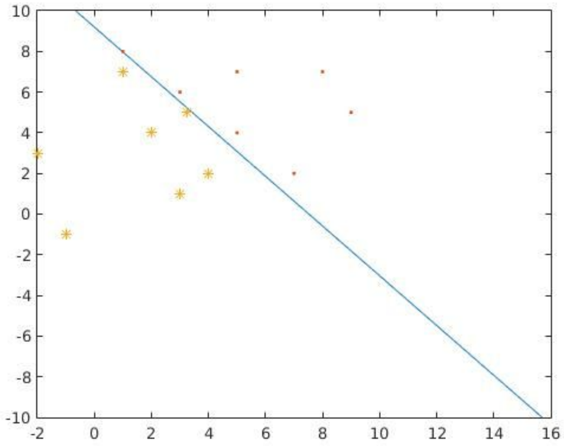
- MATLAB code

```
%% Widrow-Hoff or Least Mean Squared (LMS) Rule
a = randi([-100 100],3,1);
disp(a);
w1 = [[2,7];[8,1];[7,5];[6,3];[7,8];[5,9];[4,5]];
w2 = [[4,2];[-1,-1];[1,3];[3,-2];[5,3.25];[2,4];[7,1]];
b = 0.03*ones(1,size(w1,1)+size(w2,1));
py = [ones(size(w1,1),1),w1];
ny = [-ones(size(w2,1),1),-w2];
y = [py;ny]';
k = 1;
n = size(y,2);
it = 0;
learningRate = 0.01;
theta = 0.0000002;
while 1
     dist = (a'*y(:,k) - b(k));
     if abs( learningRate * y(:,k) * dist) < theta
          break
     end
     if a'*y(:,k) - b(k) < 0
          a = a - learningRate * y(:,k) * dist;
```

```
        end
        k = mod(k+1,n+1);
        if k == 0
            k = 1;
        end
        it = it + 1;
    end
    disp(it);
```

- Full classification may not be achieved but good classification boundary is achieved,with mis-classifieds samples ranging from 2-3 at max.
- The effect of initialization and margin are the same as discussed earlier.
- Smaller value of theta / threshold improve the convergence of the algorithm to find a decent decision boundary but also increase the convergence time considerably. Experiments show similar results as in Relaxation with margin case.
- Plot : Red - Class 1, Yellow - Class 2, Fig 1 - Unclassified case, Fig 2 - Classified case





Problem 2

- Layer class file :

```
import numpy as np

class layer:

    def __init__ (self, layerType, nUnits, nUintsPrev):
        '''
             NN layer constructor
        '''
        self.layerType = layerType
        if self.layerType == 'input':
             self.weight = np.asmatrix(np.identity(nUnits))
             self.bias = np.asmatrix(np.zeros((nUnits,1), np.float))
        else:
             self.weight = np.asmatrix(np.random.normal(0,
np.sqrt(1.0/nUintsPrev), (nUnits, nUintsPrev)))
             self.bias = np.asmatrix(np.random.rand(nUnits, 1))
        self.netActiv = np.asmatrix(np.zeros((nUnits, 1), np.float))
        self.outputVal = np.asmatrix(np.zeros((nUnits, 1), np.float))
        self.inputVal = np.asmatrix(np.zeros((nUnits, 1), np.float))
        self.delta = np.asmatrix(np.zeros((nUnits, 1), np.float))

    def feedForward (self, inputVal): # inputVal is same as output of
previous layer
        '''
             calculates netActiv and outputVal of the layer
        '''
        self.inputVal = inputVal
        self.netActiv = np.add(np.dot(self.weight, inputVal), self.bias)
        if self.layerType == 'input':
             self.outputVal = self.netActiv
        else:
             self.outputVal = sigmoid(self.netActiv)

    def backProp (self, nextVal, learningRate, nextWeight = None):
```

```
      '''
            calculates delta, updates weight and bias of layer
      '''
      # calculate delta
      if self.layerType == 'output':
            desiredOutputVal = nextVal
            costDerivative = np.subtract(self.outputVal,
desiredOutputVal)
            self.delta = np.multiply(costDerivative,
sigmoidDerivative(self.netActiv))
      elif self.layerType == 'hidden':
            nextDelta = nextVal
            self.delta = np.multiply(np.dot(np.transpose(nextWeight),
nextDelta), sigmoidDerivative(self.netActiv))
      # update wight
      weightPartialDerivative = np.dot(self.delta,
np.transpose(self.inputVal))
      self.weight = np.subtract(self.weight, np.multiply(learningRate,
weightPartialDerivative))
      # update bias
      biasPartialDerivative = self.delta
      self.bias = np.subtract(self.bias, np.multiply(learningRate,
biasPartialDerivative))

def sigmoid (x):
    '''
      sigmoid function y = 1 / (1 + exp(-x))
    '''
    y = np.divide(1.0, np.add(1.0, np.exp(np.multiply(-1, x))))
    return y

def sigmoidDerivative (x):
    '''
      y' = sigmoid(x) * (1 - sigmoid(x))
    '''
    y = sigmoid(x)
    yDash = np.multiply(y, np.subtract(1.0, y))
    return yDash
```

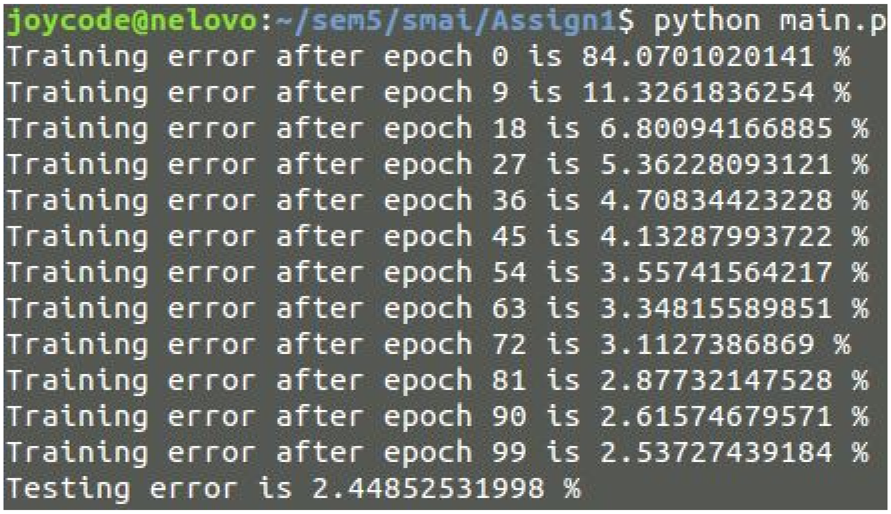
- Main file for NN :

```
from layer import *
import numpy as np
import math

layerType = {0:'input', 1:'hidden', 2:'output'}
nUnits = {0:64, 1:20, 2:10}
nUnitsPrev = {0:64, 1:64, 2:20}

def process():
    '''
      processing stuff
    '''
    TEST_FILE = 'optdigits.tes'
    TRAIN_FILE = 'optdigits.tra'
    # read training data
    with open(TRAIN_FILE) as f:
     tra_raw = f.readlines()
    # process training data
    tra_list = [map(int, item.strip().split(',')) for item in tra_raw]
    tra_data = np.transpose(np.asmatrix([item[:-1] for item in
tra_list]))
    tra_gt = [item[-1] for item in tra_list]
    # normalise training data
    mean_sample = np.divide(tra_data.sum(axis = 1), tra_data.shape[1])
    tra_data = np.subtract(tra_data, mean_sample)
    # read testing data
    with open(TEST_FILE) as f:
     tes_raw = f.readlines()
    # process training data
    tes_list = [map(int, item.strip().split(',')) for item in tes_raw]
    tes_data = np.transpose(np.asmatrix([item[:-1] for item in
tes_list]))
    tes_gt = [item[-1] for item in tes_list]
    # normalise training data
    mean_sample = np.divide(tes_data.sum(axis = 1), tes_data.shape[1])
    tes_data = np.subtract(tes_data, mean_sample)
```

```
    return tra_data,tra_gt,tes_data,tes_gt

def main():
    '''
      runs and trains NN
    '''
    #********************************************* TRAINING PART
***********************************
    tra_data, tra_gt, tes_data, tes_gt = process()

    learningRate = 0.001
    nLayers = 3
    nEpochs = 100
    nSamples = tra_data.shape[1]
    layers = []

    for i in range(nLayers):
      tmpLayer = layer(layerType[i], nUnits[i], nUnitsPrev[i])
      layers.append(tmpLayer)

    for epoch in range(nEpochs):

      nCorrect = 0

      for i in range(nSamples):
           # feed forward
           layers[0].feedForward(tra_data[:,i])
           layers[1].feedForward(layers[0].outputVal)
           layers[2].feedForward(layers[1].outputVal)

           # check correctly classified
           OutputVec = layers[2].outputVal
           if np.argmax(OutputVec) == tra_gt[i]:
                nCorrect += 1

           # desired output
           desiredOutputVec = np.asmatrix(np.zeros((nUnits[2], 1),
np.float))
           desiredOutputVec[tra_gt[i]] = 1.0
```

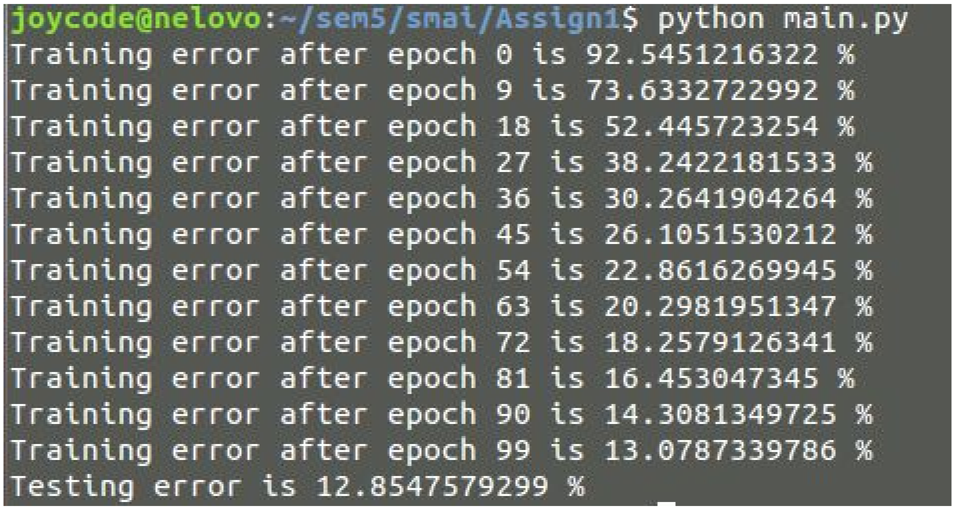
```
            # back prop
            layers[2].backProp(desiredOutputVec, learningRate)
            layers[1].backProp(layers[2].delta, learningRate,
layers[2].weight)

        if epoch % 9 == 0:
            print "Training error after epoch", epoch, "is",
(float(nSamples - nCorrect)/nSamples)*100, '%'

    #********************************************* TESTING PART
************************************

    nTestingSamples = tes_data.shape[1]
    nCorrect = 0

    for i in range(nTestingSamples):
        # feed forward
        layers[0].feedForward(tes_data[:,i])
        layers[1].feedForward(layers[0].outputVal)
        layers[2].feedForward(layers[1].outputVal)

        # check correctly classified
        OutputVec = layers[2].outputVal
        if np.argmax(OutputVec) == tes_gt[i]:
            nCorrect += 1

    print "Testing error is", (float(nTestingSamples -
nCorrect)/nTestingSamples)*100, '%'

if __name__ == "__main__":
    main()
```

- Normalise the data by mean centering the data.
- Making sure that the output of every layer is distributed normally by zero mean and unit variance using the fan in for every neuron.
- Took a constant learningRate = 0.001



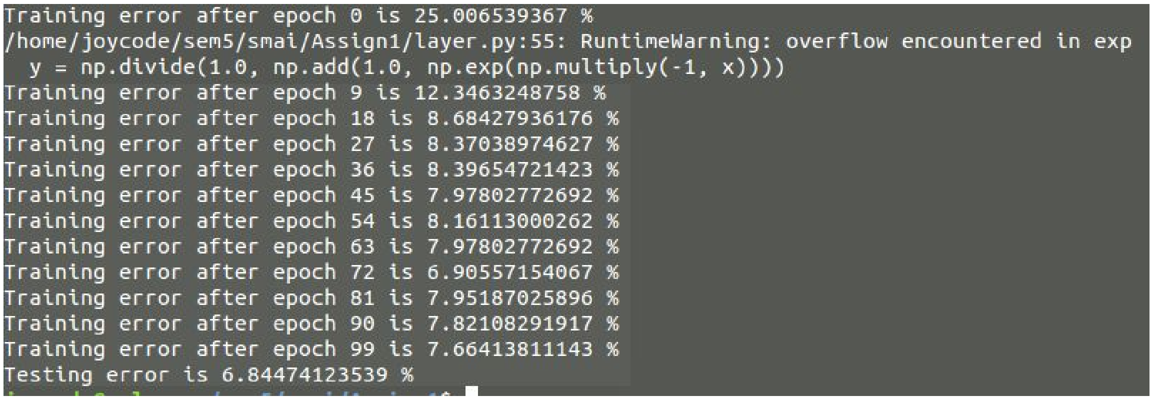
```
joycode@nelovo:~/sem5/smai/Assign1$ python main.p
Training error after epoch 0 is 84.0701020141 %
Training error after epoch 9 is 11.3261836254 %
Training error after epoch 18 is 6.80094166885 %
Training error after epoch 27 is 5.36228093121 %
Training error after epoch 36 is 4.70834423228 %
Training error after epoch 45 is 4.13287993722 %
Training error after epoch 54 is 3.55741564217 %
Training error after epoch 63 is 3.34815589851 %
Training error after epoch 72 is 3.1127386869 %
Training error after epoch 81 is 2.87732147528 %
Training error after epoch 90 is 2.61574679571 %
Training error after epoch 99 is 2.53727439184 %
Testing error is 2.44852531998 %
```

- Took a constant learningRate = 0.0001



```
joycode@nelovo:~/sem5/smai/Assign1$ python main.py
Training error after epoch 0 is 92.5451216322 %
Training error after epoch 9 is 73.6332722992 %
Training error after epoch 18 is 52.445723254 %
Training error after epoch 27 is 38.2422181533 %
Training error after epoch 36 is 30.2641904264 %
Training error after epoch 45 is 26.1051530212 %
Training error after epoch 54 is 22.8616269945 %
Training error after epoch 63 is 20.2981951347 %
Training error after epoch 72 is 18.2579126341 %
Training error after epoch 81 is 16.453047345 %
Training error after epoch 90 is 14.3081349725 %
Training error after epoch 99 is 13.0787339786 %
Testing error is 12.8547579299 %
```

- Took a decaying learningRate = A * itration ^ -p, a and p being constants



```
Training error after epoch 0 is 25.006539367 %
/home/joycode/sem5/smai/Assign1/layer.py:55: RuntimeWarning: overflow encountered in exp
  y = np.divide(1.0, np.add(1.0, np.exp(np.multiply(-1, x))))
Training error after epoch 9 is 12.3463248758 %
Training error after epoch 18 is 8.68427936176 %
Training error after epoch 27 is 8.37038974627 %
Training error after epoch 36 is 8.39654721423 %
Training error after epoch 45 is 7.97802772692 %
Training error after epoch 54 is 8.16113000262 %
Training error after epoch 63 is 7.97802772692 %
Training error after epoch 72 is 6.90557154067 %
Training error after epoch 81 is 7.95187025896 %
Training error after epoch 90 is 7.82108291917 %
Training error after epoch 99 is 7.66413811143 %
Testing error is 6.84474123539 %
```

- Hidden Layer Output and Weights are given in the following link.

https://drive.google.com/open?id=0B6aMBqXsxQfgT3h4cUV2QVE3eWc