Kathlene Ngo
Melanie Zhang
Adam Kagel
Yunxuan He

MAT 128B Project 2: Using algebraic methods for optimization: Backpropagation Neural Networks

i-Load the dataset

ii- Plot digits. Tryout the first example using the code in 17(a). We can get exactly what on the text book.

File: Project2Part2Ex1.m

File: MNISTData.m ImageExample#2.png ImageFor#2.png

Contributors: Yunxuan He

iii- A neuron. Implement a neuron where F is given as the logistic function. Take the derivative and analyze the function.

File: Neuron.m Project2Part3.m Also see attached paper.

Contributors: Yunxuan He

iv- Multiplayer Network. Implement a network with a variable number of hidden networks.

File: MultiLayerNetwork.m (for both part 4 and 5)

Contributors: Melanie Zhang

v.- Initializing the network

File: Project2Part5.m (for both part 4 and 5)

Contributors: Melanie Zhang, Adam Kagel

vi.- Training the network

File: MultiLayerNetworkTest.m MultiLayerNetworkTrain.m generateInsOuts.m generateTests.m initializeWeights.m

Contributors: Adam Kagel, Kathlene Ngo, Melanie Zhang

vii.- Dependence on parameters

test_train.m

Contributors: Kathlene Ngo, Adam Kagel, Yunxuan He

Summary Report: Kathlene Ngo, Yunxuan He

REPORT SUMMARY

i. Download *MNIST_all.mat* from Greenbaum and Chartier textbook website. Read pages 179-180.

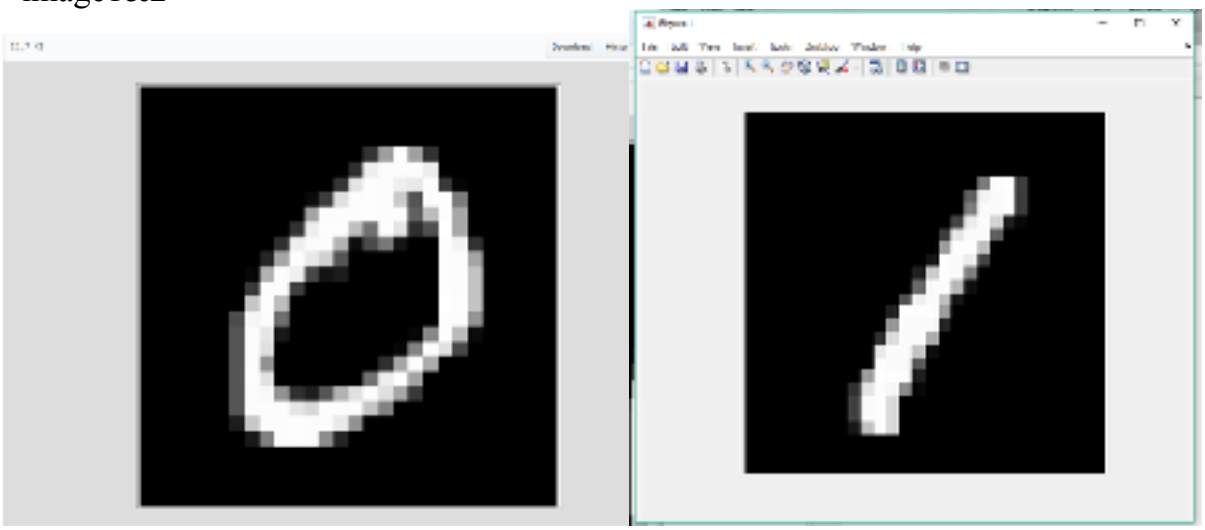
ii. Plotting digits

- **code**
- File: Project2Part2Ex1.m

%% Here we are implementing a program that reads digits from the data base given by the textbook's website (with MNIST_all.mat)

```
digit = train0(1,:);
digitImage = reshape(digit, 28, 28);
image(rot90(flipud(digitImage),-1));
colormap(gray(256)), axis square tight off;

digit = train1(1,:);
digitImage = reshape(digit, 28, 28);
image(rot90(flipud(digitImage),-1));
colormap(gray(256)), axis square tight off;
```

- image1&2

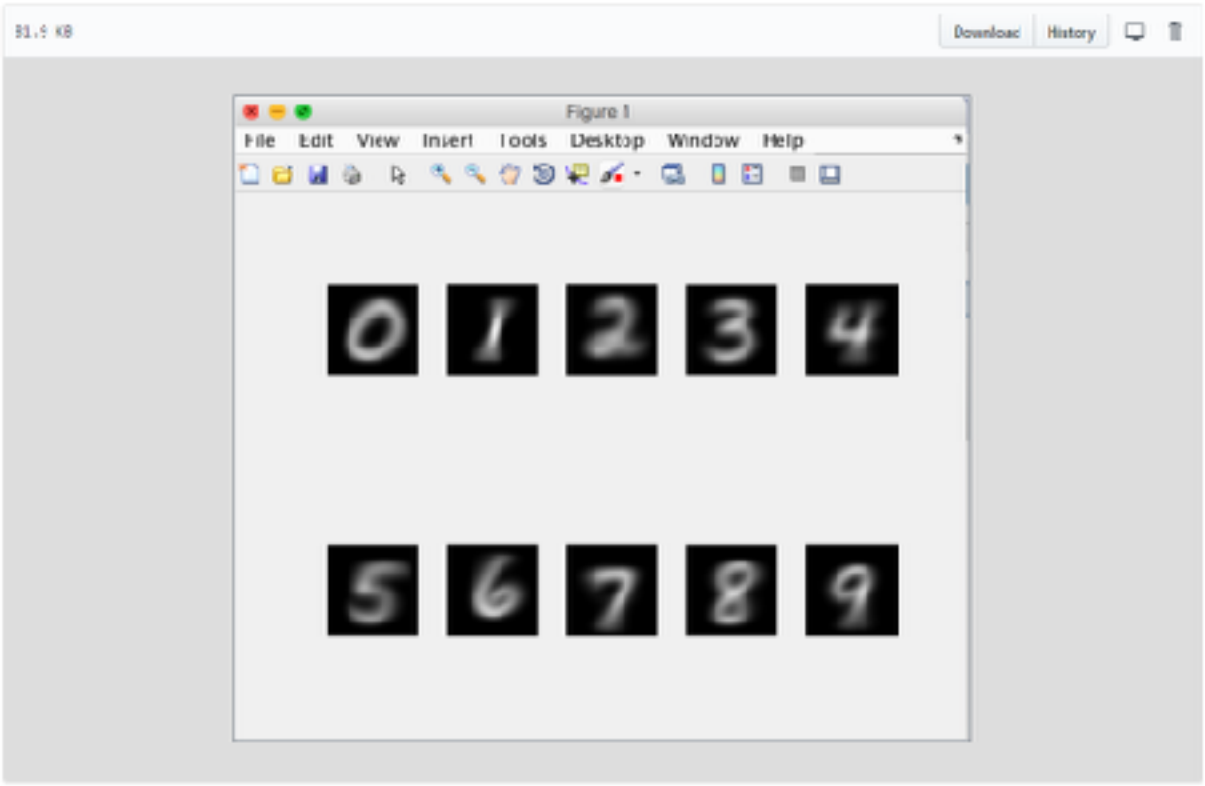


- Second Part

%% Then, we compute the average digit using command T(1,:) = mean(train0), T(2,:) = mean(train1);etc and plot them, we get the image shown on the book.

```
T(1,:) = mean(train0);
T(2,:) = mean(train1);
T(3,:) = mean(train2);
T(4,:) = mean(train3);
T(5,:) = mean(train4);
T(6,:) = mean(train5);
T(7,:) = mean(train6);
T(8,:) = mean(train7);
T(9,:) = mean(train8);
T(10,:) = mean(train9);

for i = 1:10,
 subplot(2,5,i);
 j = T(i,:);
digitimage1 = reshape(j, 28, 28);
image(rot90(flipud(digitimage1),-1));
colormap(gray(256)), axis square tight off
end;
```

- image3



iii- A neuron. Implement a neuron where F is given as the logistic function. Take the derivative and analyze the function.

- File: Neuron.m
- Also see attached paper.

```
function [ OUT ] = Neuron( InputList, InputWeight )
% Format for InputList [x1;x2;x3;etc]
% Format for InputWeight [y1 y2 y3 etc];
NET = InputWeight * InputList
OUT = 1/(1+exp(-NET));
End
```

- File: Project2Part3.m

```
% Give several pairs of InputList and InputWeight values
 InputList = [1; 2; 3];
 InputWeight = [1 2 3];
 Neuron(InputList,InputWeight)
% This gives the case where NET = 14 OUT = 9.999991684719722e-01
```

```
I = [.1;.1;.1];
O = [.2 .2 .2];
Neuron(I, O)
% This gives the case where NET = 6.000000000000001e-02 and the
% corresponding OUT = 5.149955016194100e-01

J = [.01;.01;.01];
P = [.02 .02 .02];
Neuron(J, P)
% This gives NET = 6.000000000000001e-04 and the OUT =
5.001499999954999e-01
```

- Conclusion

```
% From the above three cases we observe that for F = logistics function, as
% NET gets smaller, the value of OUT is getting smaller too. This implies
% that smaller value of NET gives small value of OUT and vice versa.
```

- We notice that the derivative of the sigmoidal (logistic) function from Figure 2 has a nice expression in terms of the OUT value.
  F(NET) = OUT = 1/(1+exp(-NET))
  F'(NET) = OUT(1-OUT) = (1/(1+exp(-NET)))(1-(1/(1+exp(-NET)))

- Using test values, we notice that the smaller value of the NET, then the smaller value of the OUT.
- We can also use the exponential function using b (constant as the base) such that F(NET)=OUT=1/(1+b^(-NET))

iv. Multilayer Network. Implement a network with a variable number of hidden networks.

File: MultiLayerNetwork.m

```
function [ O ] = MultiLayerNetwork( InputList, InputWeights)
% PART 4!!!
% Format for InputList [x1;x2;x3;etc]
% Format for InputWeights cell array [W1;W2;W3;etc] with each Wi as a
% weight matrix

O = InputList;
for i=1:length(InputWeights) % last layer is indexed at length+1
```

```
    NET = InputWeights{i} * O;  % "NET = Xw", where X is the ith matrix
                                % contained in InputWeights and w is the
                                % input vector
    O = 1/(1+exp(-NET));              % output is O = F(NET)
    O = O.';
end
```

- The last layer is NOT part of the network. It contains the values of the training set and comparing to output.
- INPUT LAYER does NOT do the calculations; Neurons in the HIDDEN and the OUTPUT layer contain NET and OUT.
- However many weight matrices = hidden layers
- The last layer is at length+1 and does not have a weight matrix.

v. Initializing the Network

File : Project2Part5.m

```
% PART 5!
% Initialize the network by assigning a random (small) number to each
% weight
n = 3;  % number of layers
WeightMatrices = cell(1,n);
for i=1:n
    WeightMatrices{i} = rand([4 4]);  % weights for each layer, initialized
                                      % to be random numbers between 0 and 1
end

input = [1; 2; 3; 4];

MultiLayerNetwork(input, WeightMatrices)
```

In this file, we are simply initializing network taking a random (small) number to each weight.

vi.- Training the network

A network will learn by iteratively adapting the values of wi,j
. Each input is associated to an output. These are called training pairs
Here is our algorithm (from PDF):

- Select next training pair (INPUT, OUTPUT) and apply the INPUT to the network ("forward pass")
- Calculate output using the network ("forward pass")
- Calculate error between the network's output and the desired output ("reverse pass")
    - ERROR (=|TARGET−OUT|)
- Adjust weights that minimizes the error ("reverse pass")
- Repeat steps for each training pair

Calculations are performed by layers, in the hidden layer(s) before any calculation is performed in the output layer.
Forward: Weights between neurons can be represented by matrix W and NET= XW (X being the input vector) (Output vector is input vector for next iteratin)
Reverse: ERROR signal when comparing OUTPUT with the TARGET value. (neuron p, hidden layer j to neuron q, output layer k)

File: MultiLayerNetworkTest.m

```
>> function [percentWrong, totalErrorRate] = MultiLayerNetworkTest(inputs, targets,
weights)

numLayers = length(weights);
errors = zeros(1, 10);
totalErrorRate = 0;
for i=1:length(inputs)
   O = inputs{i};
   OUT = cell(1, numLayers);
   % forward pass ----------------------------------------------------
   for j=1:numLayers
      NET =  O * weights{j};  % "NET = Xw", where X is the ith matrix
                              % contained in InputWeights and w is the
                              % input vector
      O = 1./(1 + exp(-NET));   % output is O = F(NET)
      OUT{j} = O;
   end
   % backward pass ---------------------------------------------------
   % for last layer
```

```
    output = OUT{numLayers};
    [Mout, Iout] = max(abs(output));
    [Mtarg, Itarg] = max(abs(targets{i}));
    errors(mod(i - 1,10) + 1) = errors(mod(i - 1,10) + 1) + (Iout ~= Itarg);
    totalErrorRate = totalErrorRate + (Iout ~= Itarg);
end
percentWrong = errors./(length(inputs)/10);
percentWrong = percentWrong * 100;
totalErrorRate = totalErrorRate * 100 / length(inputs) ;
end
```

- This function has the same inputs as the MultiLayerNetworkTrain.m; however, it does not include eta or implement any back propagation. It calculates the output and compares it to the target. The function returns a vector containing the percentage of times it gets each digit wrong and the total percentage of wrong digits.

File: MultiLayerNetworkTrain.m

```
>> function weights = MultiLayerNetworkTrain(inputs, targets, weights, eta)

% EXPECTED INPUT FORMAT:
% inputs: cell array of input COLUMN vectors
% targets: cell array of output COLUMN vectors
% ---- inputs & targets should be same length, they are our training pairs ----
% weights: cell array of weight matrices. make sure weights{1} has the same
%          number of columns as each input has entries, and
%          weights{length(weights)} has same number of rows as each target
%          has entries
% eta: a scalar between 0.01 and 0.1

numLayers = length(weights);
for i=1:length(inputs)
    O = inputs{i};
    OUT = cell(1, numLayers);
    % forward pass ----------------------------------------------------
    for j=1:numLayers
        NET =  O * weights{j};  % "NET = Xw", where X is the ith matrix
                                % contained in InputWeights and w is the
                                % input vector
        O = 1./(1 + exp(-NET));   % output is O = F(NET)
        OUT{j} = O;
```

```
    end
    % backward pass ----------------------------------------------------
    % for last layer
    output = OUT{numLayers};
    error = output - targets{i};  % dont know if abs should be there
    delta = output .* (1 - output) .* error;
    weightUpdate = eta * OUT{numLayers-1}' * delta;
    weights{numLayers} = weights{numLayers} - weightUpdate;

    % for other hidden layers
    for j=numLayers-1:-1:2
        delta = (delta * weights{j+1}') .* OUT{j} .* (1 - OUT{j});
        weightUpdate = eta * OUT{j-1}' * delta;
        weights{j} = weights{j} - weightUpdate;
    end
end
```

- This function has inputs of cell arrays (inputs, targets, weights) and a float input of eta. It returns the weights after training them on all input-target pairs. Weights is a cell array with each element being the weight matrix for one layer to the next.

File: generateInsOuts.m

```
>> function [inputs, targets] = generateInsOuts(datasetName, trainLength)

load(datasetName);

if(trainLength > 5421)
  trainLength = 5421;
end
rawTrainData = cell(1, 10);
rawTrainData{1} = train0;
rawTrainData{2} = train1;
```

```
rawTrainData{3} = train2;
rawTrainData{4} = train3;
rawTrainData{5} = train4;
rawTrainData{6} = train5;
rawTrainData{7} = train6;
rawTrainData{8} = train7;
rawTrainData{9} = train8;
rawTrainData{10} = train9;

inputs = cell(1, 10*trainLength);
targets = cell(1, 10*trainLength);

j = 1;
for i = 1:(10*trainLength)
    train = rawTrainData{j};
    digit = train(ceil(i/10),:);
    digitimage = reshape(digit, 28, 28);
    digitimage = rot90(flipud(digitimage),-1);
    digitimage = digitimage(:);
    digitimage = (digitimage > 0);
    inputs{i} = digitimage';

    targets{i} = 1:10;
    targets{i} = (targets{i} == j);

    j = j + 1;
    if j > 10
      j = j - 10;
    end
end
clear -regexp ^train ^test;
end
```

File: generateTests.m

```
>> function [inputs, targets] = generateTests(datasetName, numTests)

load(datasetName);

if(numTests > 892)
  numTests = 892;
end

rawTrainData = cell(1, 10);
rawTrainData{1} = test0;
rawTrainData{2} = test1;
rawTrainData{3} = test2;
rawTrainData{4} = test3;
rawTrainData{5} = test4;
rawTrainData{6} = test5;
rawTrainData{7} = test6;
rawTrainData{8} = test7;
rawTrainData{9} = test8;
rawTrainData{10} = test9;

inputs = cell(1, 10*numTests);
targets = cell(1, 10*numTests);

j = 1;
for i = 1:(10*numTests)
   train = rawTrainData{j};
   digit = train(ceil(i/10),:);
   digitimage = reshape(digit, 28, 28);
   digitimage = rot90(flipud(digitimage),-1);
   digitimage = digitimage(:);
   digitimage = (digitimage > 0);
   inputs{i} = digitimage';

   targets{i} = 1:10;
```

```
    targets{i} = (targets{i} == j);

    j = j + 1;
    if j > 10
      j = j - 10;
    end
end
clear -regexp ^train ^test;
end
```

- The generateTests and generateInsOuts functions take data from the MNIST_all.m data set and format the images into a cell array of 784x1 input values and a cell array of 10x1 target values.

File: InitializeWeights.m

```
>> % Assuming number of output nodes is 10 and input nodes is 9
function weights = initializeWeights(numLayers, nodesPerLayer)

weights = cell(1,numLayers);
weights{1} = -0.25 + 0.5*rand([784 nodesPerLayer]);
for i=2:(numLayers-1)
    weights{i} = -0.25 + 0.5*rand([nodesPerLayer nodesPerLayer]);
end
weights{numLayers} = -0.25 + 0.5*rand([nodesPerLayer 10]);

end
```

This is the modified version of function from Part 5.

---

vii.- Dependence on parameters
The learning of the network (i.e. the minimization
Of the error) will depend on the number of layers, the number of neurons per layer, and for fixed values of these two parameters the network will also depend on the size of the training set. Set up a study in which you change the values of these parameters and report the error(s) you obtain (you will obtain an error for the training set and another for the test test–which should be very similar to each other provided the test and training set are similar enough).

File: test_train.m

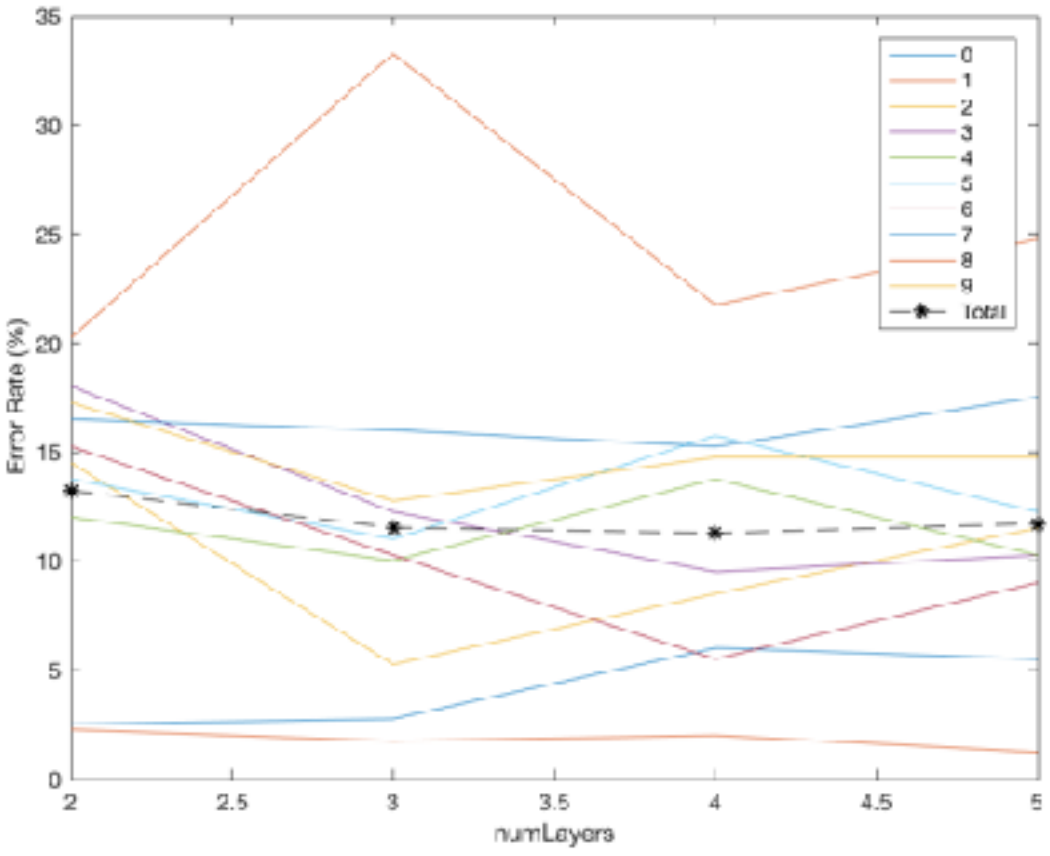
```
>> % testing MultiLayerNetworkTrain
%
% to actually extend this to a handwriting recognizer, each input will be % a vector of length 9, with each entry of the vector representing one
% grid section of the image, and each target will be a vector of length 10 % with zeroes in all entries EXCEPT the digit it represents. (exception:
% if it's 0 then the 10th entry will be 1)
% so for example, if input{1} is a vector representing the image of a
% handwritten 2, then target{1} will be [0; 1; 0; 0; 0; 0; 0; 0; 0; 0]
% also we need to make sure that the first weight matrix is n x 9 and the
% last weight matrix is 10 x m (for some n, m, doesn't matter as long as
% all the dimensions match up for adjacent matrices)
%
```

File: test_train.m

```
numLayers = 3;
nodesPerLayer = 400;
eta = 0.1;

setsOfDigitsToTrainOn = 1000;
trainingSessions = 100;

[inputs, targets] = generateInsOuts("mnist_all.mat", setsOfDigitsToTrainOn);
weights = initializeWeights(numLayers, nodesPerLayer);

% this function returns the set of weights that should have been updated
% through the training to maximize accuracy of prediction.
% i need to write more code to implement the actual testing that will come
```

```
% after the training. should b pretty straightforward
h = waitbar(0,sprintf('%.2f%% done', 0.0));
for i = 1:trainingSessions
    weights = MultiLayerNetworkTrain(inputs, targets, weights, eta);
    waitbar(i/trainingSessions,h,sprintf('%.2f%% done', i*100/trainingSessions));
end
close(h);

[testIns, testOuts] = generateTests("mnist_all.mat", 400);
[percentWrong, totalErrorRate] = MultiLayerNetworkTest(testIns, testOuts, weights);

disp(percentWrong);
disp(totalErrorRate);
```

- We set up a study to examine the changes in error by altering the parameters (number of layers, number of nodes per layer, and size of training size).
- First we change the number of layers and fix the other two parameters and we got the following image



- From that we conclude that the change in number of layers does not affect the error rate, i.e. the error rate fluctuate between 10% and 16% when number of layers is between 2 and 9. Further investigation is needed for greater number of layers.
- Then we change the number of nodes per layer and fix the other two variables to be the same. Using the conditions below and we get the conclusion that as the number of nodes per layer increases, the error gets slightly bigger. Comparing the two values below, as number of layers and the size of training sets stay unchanged, the error rate increases as the number of nodes per layer increases from 500 to 700.

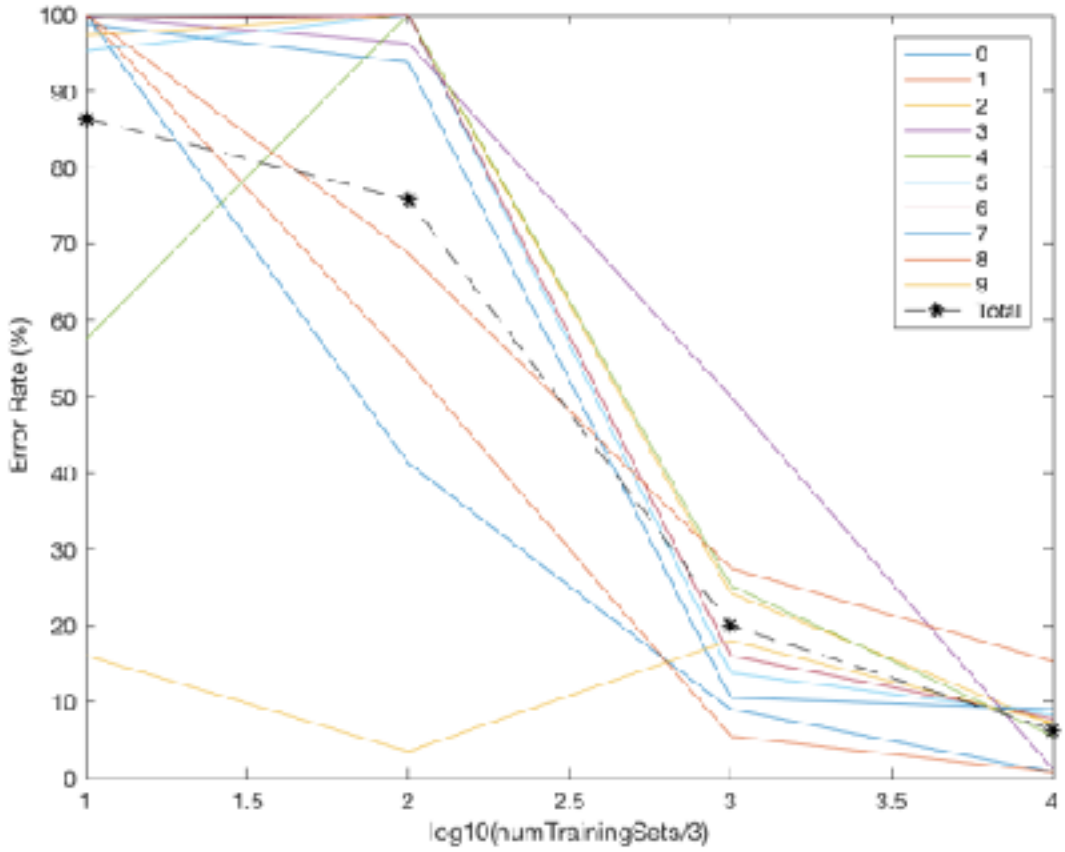
numLayers = 3;
nodesPerLayer = 500;

```
>> LayerTest
44          weights{j} = weights{j} - weightUpdate;
  Columns 1 through 3

    1.2500222000022200e+22     1.5000220000220000e+22     3.5002200000220000e+20

  Columns 4 through 6

    4.0000222000022200e+22     4.5000220000220000e+22     3.5002200000220000e+20

  Columns 7 through 9

    3.5000222000022200e+22     6.7500220000220000e+22     6.7502200000220000e+20

  Column 10

    5.5000222000022200e+22

    4.0750222000022200e+22

>>
```

numLayers=3;
nodesPerLayer=700;

```
>> LayerTest
  Columns 1 through 3

    2.0220002000020002e+02     1.7500020000200002e+22     2.7500200002000020e+20

  Columns 4 through 6

    8.5220002000020002e+02     6.7500020000200022e+22     1.2500200002000020e+20

  Columns 7 through 9

    6.5220002000020002e+02     5.2200020000200022e+22     6.2000200002000020e+20

  Column 10

    7.2400999999999992e+02

    4.7750002000020002e+02

>>
```

- Finally we set the number of layers and number of nodes per layer to be the same and change the size of the training set. We get the following image



- So there is the conclusion that as the number of training set decreases the error rate also decreases greatly. One possible reason for that is the decrease in the size of training sets also decreases the possibility of going into reverse pass and doing adjustment to weights. thus decreasing the error rate.