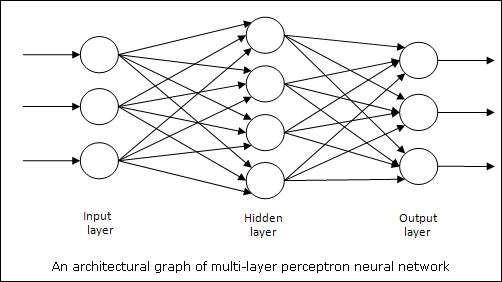
**Sayan Banerjee**

**UWM**

EE:890 Mini Project#4 Report

# Mini Project #4: Hand-Written Digit Recognition Using a Multi-Layer Perceptron



**Architecture:**

Input Neuron: 784

Hidden Neurons: 25

Output Neuron: 10

Sigmoid function has been used to calculate the activation value at each neuron at Hidden Layer and Output Layer. At Input Layer the input values serves as activation values for each neuron.

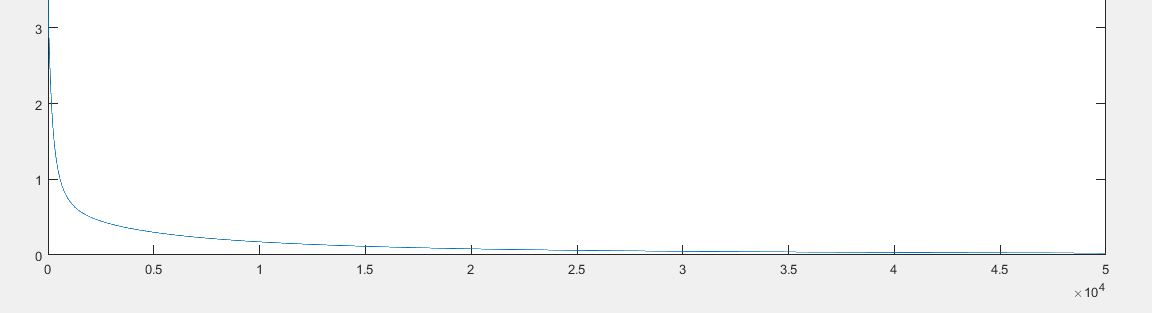
Instead of squared error to calculate the cost/loss of the network, I have used **Cross Entropy** to calculate the cost/loss at each iteration for training and batch gradient descent has been used as optimization technique to get the optimized weights.

**Cross Entropy**:

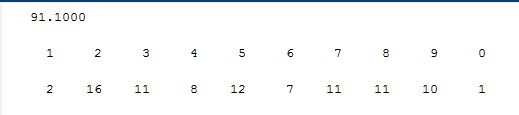
**Learning Rate:** 0.13

**Total Number of Iterations:** 50000

**Accuracy Achieved:** 91.1%



**Graph of Training Cost with No. of Iterations**



**Total Accuracy and Misclassified Class with Counts when Testing**

**MATLAB Code:**

clear;

clc;

iteration=50000;

LR=0.13;

input\_layer=784;

hidden\_layer=25;

output\_layer=10;

N=5000;

error=0;

Train\_ErrorClass=[1 2 3 4 5 6 7 8 9 0];

Test\_ErrorClass=[1 2 3 4 5 6 7 8 9 0];

Train\_ErrorClassCount=zeros(1,10);

Test\_ErrorClassCount=zeros(1,10);

load digits.mat

for k = 1:5000

dummy = train(:,k) ;

for i = 1:28

for j = 1:28

%x(i,j,k) = double(dummy((i-1)\*28 + j)) ;

end

end

% imagesc(x(:,:,k)')

% colormap('gray')

% colorbar

% pbaspect([1 1 1])

% pause(5.0)

end

train\_T=double(train');

test\_T=double(test');

[q,e]=size(train\_T);

Train\_Normalized\_Features=train\_T/255;

Train\_Normalized\_Features=[ones(q,1) Train\_Normalized\_Features];

Train\_Normalized\_Features=Train\_Normalized\_Features';

train\_label=oneHotEncoding(trainlabels);

Test\_Normalized\_Features=test\_T/255 ;

[q,e]=size(test\_T);

Test\_Normalized\_Features=[ones(q,1) Test\_Normalized\_Features];

Test\_Normalized\_Features=Test\_Normalized\_Features';

test\_label=oneHotEncoding(testlabels);

W1=weightInitialization(hidden\_layer,input\_layer+1);

W2=weightInitialization(output\_layer,hidden\_layer+1);

%%====================TRAINING==================================%%

cost=double(zeros(iteration,1));

for iter=1:iteration

Z2=W1\*Train\_Normalized\_Features;

A2=sigmoid(Z2);

A2=[ones(1,N); A2];

Z3=W2\*A2;

A3=sigmoid(Z3);

prediction=A3';

cost(iter)=costCalc(N, train\_label,prediction);

delta\_3=delta\_3\_calc(N, prediction, train\_label,Z3);

delta\_2=delta\_2\_calc(W2, delta\_3, Z2);

big\_delta\_1=Train\_Normalized\_Features\*delta\_2;

big\_delta\_1=big\_delta\_1./N;

W1\_grad=big\_delta\_1';

big\_delta\_2=A2\*delta\_3;

big\_delta\_2=big\_delta\_2./N;

W2\_grad=big\_delta\_2';

W1=W1+(LR.\*W1\_grad);

W2=W2+(LR.\*W2\_grad);

end

plot(1:iteration,cost);

%%%=========================TESTING=====================================%%%

T\_N=1000;

T\_Z2=W1\*Test\_Normalized\_Features;

T\_A2=sigmoid(T\_Z2);

T\_A2=[ones(1,T\_N); T\_A2];

T\_Z3=W2\*T\_A2;

T\_A3=sigmoid(T\_Z3);

T\_prediction=T\_A3';

[dummy, p] = max(T\_prediction, [], 2);

for i=1:T\_N

if p(i,1)==10

p(i,1)=0;

end

if double(p(i,1))~=double(testlabels(i,1))

error=error+1;

if testlabels(i,1)==0

Test\_ErrorClassCount(1,10)=Test\_ErrorClassCount(1,10)+1;

else

y=testlabels(i,1);

Test\_ErrorClassCount(1,y)=Test\_ErrorClassCount(1,y)+1;

end

end

end

accuracy=((1000-error)/1000)\*100;

disp(accuracy);

disp(Test\_ErrorClass);

disp(Test\_ErrorClassCount);

%%%======================================================================%%%

function J = costCalc(N,actual\_label,prediction)

cost = (-actual\_label.\*log(prediction))-((1-actual\_label).\*log(1-prediction));

J= sum(cost(:));

J =J/N;

End

function delta2=delta\_2\_calc(W2, delta\_3, Z2)

sg\_g=sigmoidGradient(Z2);

W=W2(:,2:end);

delta2=(delta\_3\*W).\*sg\_g';

end

function delta3=delta\_3\_calc(N, prediction, train\_label,Z3)

diff=train\_label-prediction;

delta3=diff;

end

function [x] = oneHotEncoding(labels)

[i,j]=size(labels);

x=zeros(i,10);

for m=1:i

p=labels(m,1);

if (p==0)

x(m,10)=1;

else

x(m,p)=1;

end

end

end

function g = sigmoid(z)

%SIGMOID Compute sigmoid functoon

% J = SIGMOID(z) computes the sigmoid of z.

g = 1.0 ./ (1.0 + exp(-z));

end

function g = sigmoidGradient(z)

g = double(zeros(size(z)));

sigm = 1.0 ./ (1.0 + exp(-z));

g = sigm.\*(1-sigm);

end

function [x] = weightInitialization(row, col)

epsilon=sqrt(6)/(sqrt(row+col));

x=rand(row,col)\*(2\*epsilon)-epsilon;

end