PROJECT REPORT

*on*

**Handwritten Digit Recognition using ANN**

*Submitted in partial fulfilment for award of the degree*

*Of*

**BACHELOR OF TECHNOLOGY**

*In*

**COMPUTER SCIENCE AND ENGINEERING**

*by*

**Aditya Singh (1031310069)**

*Under the guidance of*

**Dr. S. S. Sridhar**

**Professor of Computer Science and Engineering**



**DEPARTMENT OF COMPUTERSCIENCE AND ENGINEERING**

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**SRM UNIVERSITY**

**(Under section 3 of UGC Act 1956)**

SRM Nagar, Kattankulathur – 603203

Kancheepuram District

**APRIL 2015­­­­­**

PROJECT REPORT

*on*

**Handwritten Digit Recognition using ANN**

*Submitted in partial fulfilment for award of the degree*

*Of*

**BACHELOR OF TECHNOLOGY**

*In*

**COMPUTER SCIENCE AND ENGINEERING**

*by*

**Aditya Singh (1031310069)**

*Under the guidance of*

**Dr. S. S. Sridhar**

**Professor of Computer Science and Engineering**



**DEPARTMENT OF COMPUTERSCIENCE AND ENGINEERING**

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**SRM UNIVERSITY**

**(Under section 3 of UGC Act 1956)**

SRM Nagar, Kattankulathur – 603203

Kancheepuram District

**APRIL 2015­­­­­**

**BONAFIDE CERTIFICATE**

Certified that this project report titled “Common Fund Management System” is the bonafide work of Aditya Singh (Reg.No:1031310069), who carried out the project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on any other candidate.

|  |  |
| --- | --- |
| **Signature of the Guide**  **DR. S. S. SRIDHAR**  Professor  Department of Computer Science and  Engineering  SRM University  Kattakulathur - 603203 | **Signature of the Head of Dept.**  **DR. E. POOVAMMAL**  Professor & Head  Department of Computer Science and  Engineering  SRM University  Kattakulathur - 603203 |

Submitted for Project Work Viva-voce Examination held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Place: Kattankulathur

Date

|  |  |
| --- | --- |
| **INTERNAL EXAMINER** | **EXTERNAL EXAMINER** |

**CONTENTS**

**Acknowledgment**

**Introduction**

**Objective**

**Artificial Neural Networks**

**Implementation**

**Sample Code**

**Screenshots**

**Conclusion**

**Future Enhancement**

**References**

**ACKNOWLEDGEMENT**

Apart from the effort put in by myself, the success of the project depend largely on encouragement, guidance and support of many others. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of the project.

I would like to thank the Director of Engineering& Technology Dr. C. Muthamizhchelvan for providing the technical aid and support needed throughout the project. I am also very grateful to my Head of the Department Dr. E. Poovammal (Head of Department) for allowing me to access every possible need and help.

I am highly indebted to Class In Charge Mrs. Vathana D.(Assistant Professor) for my guidance and constant supervision as well as providing necessary information regarding the project topic and also for her support in completing the project

I would like to show my greatest appreciation to my Guide Dr. S. S. Sridhar (Assistant Professor). He gave me moral support and guided me in different matters regarding the topic. He has been very kind and patient, whilst suggesting to me the outlines of this project and clarifying my doubts. There is no way that I could thank him for all he has done for us.

**INTRODUCTION**

The need to process handwritten documents still exists today despite growing adoption of electronic communication. For example, the postal service accepts packages and envelopes with handwritten addresses, and it is both impractical and expensive to have humans sort all of the mail. Additionally, many companies continue to provide means by which people can submit handwritten forms and other documentation. As the population and industrial sector continue to grow, so does the amount of handwritten data that needs to be processed. The expenses associated with paying human employees to sort through the data and enter the information manually into a computer becomes increasingly more time-consuming and expensive. Knowing this, it is easy to see that it would be beneficial to automate part or all of the process. Many techniques for automatically recognizing handwritten data have already been developed and tested over the past several years.

**OBJECTIVE**

In this project, I developed a machine learning program which is able to recognize human’s handwritten digit from pictures. The learning algorithm I used is “artificial neural network”, which is a computational model inspired by animals’ central nervous systems. The input pictures should be processed to only black-white color with a fixed number of pixels. The predict function will be able to output the prediction of any pictures of human’s handwritten digit based on weight vectors, “Theta(l)”, of the neural network. We will obtain the weight vectors by training the neural network with a set of pictures (training set) of human’s handwritten digit.

**ARTIFICIAL NEURAL NETWORKS**

Here is a brief review of ANN algorithm. An ANN model is composed by three parts, input layer, output layer and hidden layer. The size of input layer, N, is the number of element in each data vector, X(i). Each node of the input layer represent one element in X(i). The output layer represent the results of learning function. The number of nodes in output layer, M, is the number of classes we want to category the input data. Thus, if M > 1, we will have a multiclass classification problem. The hidden layers can be any size as long as greater than M and smaller than N. They are the most important part of ANN model. The role of hidden layers are to transform the data of the input layer to the scale/size that output layer can use. One can apply any function in the hidden layers. In ANN, every layer is connected. For a ANN of layer total L, all nodes in layer l (1<l<L) is the input to every node in layer l-1 with weight Theta(l).

**MODEL REPRESENTATION**

In the following ANN model, I apply logistic regression in the hidden layers. Each node in hidden layer has a sigmoid function to transfer the data from previous layer. In this project, I chose L = 3, where l = 1 is input layer, l = 2 is hidden layer, and l = 3 is output layer. Thus, Theta(1) is the weight from l = 1 to l = 2, and Theta(2) is the weight from l = 2 to l = 3. The cost function is similar to logistic regression’s except we need to consider all the nodes.

The input data X, is a set of black-white picture of 20X20 pixels in each. Thus the size of X(i) is 400. Each element stores a scale of grey from 0 to 1 (0 is white and 1 is black). The size of hidden layer is 25 and output layer is 10 (digits 0 – 9). The size of Theta(1) is 25X401 and Theta(2) is 10X26, noted that I considered to add the bias term.

**COST FUNCTION**

The cost function, J(Theta(l)), determine the “cost” of current weight vectors. If we can build such a cost function, we can apply gradient descent algorithm to determine the best weight vectors for our training set. We can obtain the gradient by calculating the partial derivative of weight vector, Theta(l), of each layer.

**BACK PROPAGATION ALGORITHM**

We use this algorithm to compute the gradient for the cost function. Before implementing the algorithm, we may need to calculate the gradient of Sigmoid function, g'(z) = d/dz(g(z)) = g(z)(1-g(z)). gradient of Sigmoid function is important for calculating the “error term”, delta(l,j), which measures how much that node(l,j) was “responsible” for any errors in our output. In backpropagation algorithm, we calculate delta(l=1), delta(l=2), delta(l=3). Note that we need to first calculate delta(l=3), and then use delta(l=3) to obtain delta(l=2). Intuitively, delta(l) is related to the step size of the gradient decedent applying to the partial derivative of J in respect to Theta(l-1). Each step size of d/d(Theta(l))(J) is delta(l+1)(a(l))^T, where a is the set of elements of layer l.

Similarly, we also need to compute the regularization term in our gradient function to prevent the over-fitting issue. Except the bias term, we need to have a regularized term, lambda/mTheta(l)(i,j) in each node.

**CODE**

**Trainng Function:**

function TrainFunc(wd\_coefficient, n\_hid, n\_iters, learning\_rate, momentum\_multiplier, do\_early\_stopping, mini\_batch\_size)

% wd\_coefficient Weight decay coefficient

% n\_hid Number of hidden units

% n\_iters Number of iterations

% learing\_rate

% momentum\_multiplier

% do\_early\_stopping

% mini\_batch\_size

warning('error', 'Octave:broadcast');

if exist('page\_output\_immediately'), page\_output\_immediately(1); end

more off;

model = initial\_model(n\_hid); %Initialize the weights

from\_data\_file = load('data.mat');

datas = from\_data\_file.data;

n\_training\_cases = size(datas.training.inputs, 2); %size(data.training.inputs)=(256 1000)

%if n\_iters ~= 0, test\_gradient(model, datas.training, wd\_coefficient); end %test if the code for the gradient is ok

% optimization

theta = model\_to\_theta(model);

momentum\_speed = theta \* 0;

training\_data\_losses = [];

validation\_data\_losses = [];

if do\_early\_stopping,

best\_so\_far.theta = -1; % this will be overwritten soon

best\_so\_far.validation\_loss = inf;

best\_so\_far.after\_n\_iters = -1;

end

for optimization\_iteration\_i = 1:n\_iters,

fprintf('%d ',optimization\_iteration\_i);

model = theta\_to\_model(theta);

%Prepare the batcth for learning

training\_batch\_start = mod((optimization\_iteration\_i-1) \* mini\_batch\_size, n\_training\_cases)+1;

training\_batch.inputs = datas.training.inputs(:, training\_batch\_start : training\_batch\_start + mini\_batch\_size - 1);

training\_batch.outputs = datas.training.outputs(:, training\_batch\_start : training\_batch\_start + mini\_batch\_size - 1);

%Compute the gradient and update the weights

gradient = model\_to\_theta(d\_loss\_by\_d\_model(model, training\_batch, wd\_coefficient));

momentum\_speed = momentum\_speed \* momentum\_multiplier - gradient;

theta = theta + momentum\_speed \* learning\_rate;

model = theta\_to\_model(theta);

training\_data\_losses = [training\_data\_losses, loss(model, datas.training, wd\_coefficient)];

validation\_data\_losses = [validation\_data\_losses, loss(model, datas.validation, wd\_coefficient)];

if do\_early\_stopping && validation\_data\_losses(end) < best\_so\_far.validation\_loss,

best\_so\_far.theta = theta; % this will be overwritten soon

best\_so\_far.validation\_loss = validation\_data\_losses(end);

best\_so\_far.after\_n\_iters = optimization\_iteration\_i;

end

if mod(optimization\_iteration\_i, round(n\_iters/10)) == 0,

fprintf('\nAfter %d optimization iterations, training data loss is %f, and validation data loss is %f\n', optimization\_iteration\_i, training\_data\_losses(end), validation\_data\_losses(end));

end

end

if n\_iters ~= 0, test\_gradient(model, datas.training, wd\_coefficient); end % check again, this time with more typical parameters

if do\_early\_stopping,

fprintf('Early stopping: validation loss was lowest after %d iterations. We chose the model that we had then.\n', best\_so\_far.after\_n\_iters);

theta = best\_so\_far.theta;

end

% the optimization is finished. Now do some reporting.

model = theta\_to\_model(theta);

clf; %Clear current figure window

hold on;

plot(training\_data\_losses, 'b');

plot(validation\_data\_losses, 'r');

legend('training', 'validation');

ylabel('loss');

xlabel('iteration number');

hold off;

datas2 = {datas.training, datas.validation, datas.test};

data\_names = {'training', 'validation','test'};

for data\_i = 1:3,

data = datas2{data\_i};

data\_name = data\_names{data\_i};

fprintf('\nThe loss on the %s data is %f\n', data\_name, loss(model, data, wd\_coefficient));

if wd\_coefficient~=0,

fprintf('The classification loss (i.e. without weight decay) on the %s data is %f\n', data\_name, loss(model, data, 0));

end

fprintf('The classification error rate on the %s data is %f\n', data\_name, classification\_performance(model, data));

end

save -mat7-binary model.mat model;

end

% test\_gradient is a function used to test our implementation of the

% gradient calculation

function test\_gradient(model, data, wd\_coefficient)

base\_theta = model\_to\_theta(model);

h = 1e-2;

correctness\_threshold = 1e-5;

analytic\_gradient = model\_to\_theta(d\_loss\_by\_d\_model(model, data, wd\_coefficient));

% Test the gradient not for every element of theta, because that's a lot of work. Test for only a few elements.

for i = 1:100,

test\_index = mod(i \* 1299721, size(base\_theta,1)) + 1; % 1299721 is prime and thus ensures a somewhat random-like selection of indices

analytic\_here = analytic\_gradient(test\_index);

theta\_step = base\_theta \* 0;

theta\_step(test\_index) = h;

contribution\_distances = [-4:-1, 1:4];

contribution\_weights = [1/280, -4/105, 1/5, -4/5, 4/5, -1/5, 4/105, -1/280];

temp = 0;

for contribution\_index = 1:8,

temp = temp + loss(theta\_to\_model(base\_theta + theta\_step \* contribution\_distances(contribution\_index)), data, wd\_coefficient) \* contribution\_weights(contribution\_index);

end

fd\_here = temp / h;

diff = abs(analytic\_here - fd\_here);

% fprintf('%d %e %e %e %e\n', test\_index, base\_theta(test\_index), diff, fd\_here, analytic\_here);

if diff < correctness\_threshold, continue; end

if diff / (abs(analytic\_here) + abs(fd\_here)) < correctness\_threshold, continue; end

error(sprintf('Theta element #%d, with value %e, has finite difference gradient %e but analytic gradient %e. That looks like an error.\n', test\_index, base\_theta(test\_index), fd\_here, analytic\_here));

end

fprintf('Gradient test passed. That means that the gradient that your code computed is within 0.001%% of the gradient that the finite difference approximation computed, so the gradient calculation procedure is probably correct (not certainly, but probably).\n');

end

function ret = logistic(input)

ret = 1 ./ (1 + exp(-input));

end

function ret = log\_sum\_exp\_over\_rows(a)

% This computes log(sum(exp(a), 1)) in a numerically stable way

maxs\_small = max(a, [], 1);

maxs\_big = repmat(maxs\_small, [size(a, 1), 1]);

ret = log(sum(exp(a - maxs\_big), 1)) + maxs\_small;

end

function ret = loss(model, data, wd\_coefficient)

% model.input\_to\_hid is a matrix of size (n\_hid,256)

% model.hid\_to\_class is a matrix of size (10,256)

% data.inputs is a matrix of size (256,<number of data cases>)

% data.outputs is a matrix of size (10,<number of data cases>)

% first, do the forward pass, i.e. calculate a variety of relevant values

hid\_in = model.input\_to\_hid \* data.inputs; % input to the hidden units, i.e. before the logistic. size: (n\_hid,<number of data cases>)

hid\_out = logistic(hid\_in); % output of the hidden units, i.e. after the logistic. size: (n\_hid,<number of data cases>)

class\_in = model.hid\_to\_class \* hid\_out; % input to the components of the softmax. size: (10, <number of data cases>)

class\_normalizer = log\_sum\_exp\_over\_rows(class\_in); % log(sum(exp)) is what we subtract to get normalized log class probabilities. size: (1,<number of data cases>)

log\_class\_prob = class\_in - repmat(class\_normalizer, [size(class\_in, 1), 1]); % log of probability of each class. size: (10, <number of data cases>)

class\_out = exp(log\_class\_prob); % probability of each class. Each column (i.e. each case) sums to 1. size: (10, <number of data cases>)

classification\_loss = -mean(sum(log\_class\_prob .\* data.outputs, 1)); % select the cross entropy right log class probability using that sum; then take the mean over all data cases.

wd\_loss = sum(model\_to\_theta(model).^2)/2\*wd\_coefficient; % very straightforward: E = 1/2 \* lambda \* theta^2

ret = classification\_loss + wd\_loss;

end

function ret = d\_loss\_by\_d\_model(model, data, wd\_coefficient)

% model.input\_to\_hid is a matrix of size (n\_hid,256)

% model.hid\_to\_class is a matrix of size (10,n\_hid)

% data.inputs is a matrix of size (256,<number of data cases>)

% data.outputs is a matrix of size (10,<number of data cases>)

% The returned object is supposed to be exactly like parameter <model>, i.e. it has fields ret.input\_to\_hid and ret.hid\_to\_class. However, the contents of those matrices are gradients (d loss by d model parameter), instead of model parameters.

% This is the only function that you're expected to change. Right now, it just returns a lot of zeros, which is obviously not the correct output. Your job is to change that.

ret.input\_to\_hid = model.input\_to\_hid \* 0;

ret.hid\_to\_class = model.hid\_to\_class \* 0;

% first, do the forward pass, i.e. calculate a variety of relevant values

hid\_in = model.input\_to\_hid \* data.inputs; % input to the hidden units, i.e. before the logistic. size: (n\_hid,<number of data cases>)

hid\_out = logistic(hid\_in); % output of the hidden units, i.e. after the logistic. size: (n\_hid,<number of data cases>)

class\_in = model.hid\_to\_class \* hid\_out; % input to the components of the softmax. size: (10, <number of data cases>)

class\_normalizer = log\_sum\_exp\_over\_rows(class\_in); % log(sum(exp)) is what we subtract to get normalized log class probabilities. size: (1,<number of data cases>)

log\_class\_prob = class\_in - repmat(class\_normalizer, [size(class\_in, 1), 1]); % log of probability of each class. size: (10, <number of data cases>)

class\_out = exp(log\_class\_prob); % probability of each class. Each column (i.e. each case) sums to 1. size: (10, <number of data cases>)

error\_deriv = class\_out - data.outputs; %Error derivate w.r.t. zj size (10,<number of data cases>

hid\_to\_output\_weights\_gradient = [];

for i=1:size(error\_deriv,2),

hid\_to\_output\_weights\_gradient(:,:,i) = hid\_out(:,i)\*error\_deriv(:,i)'; %Gradient size must be(n\_hid,10,<number of data cases>

end

hid\_to\_output\_weights\_gradient = mean(hid\_to\_output\_weights\_gradient,3); %We need the mean

ret.hid\_to\_class = hid\_to\_output\_weights\_gradient'; %traspose to fit into the model

backpropagate\_error\_deriv = model.hid\_to\_class'\*error\_deriv; %size(n\_hid,<number of data cases>) %Not sure about this

input\_to\_hidden\_weights\_gradient = [];

for i=1:size(backpropagate\_error\_deriv,2),

input\_to\_hidden\_weights\_gradient(:,:,i) = data.inputs(:,i)\*((1-hid\_out(:,i)).\*hid\_out(:,i).\*backpropagate\_error\_deriv(:,i))'; %Gradient size must be(n\_hid,10,<number of data cases>

end

input\_to\_hidden\_weights\_gradient = mean (input\_to\_hidden\_weights\_gradient,3);

ret.input\_to\_hid=input\_to\_hidden\_weights\_gradient';

ret.input\_to\_hid = ret.input\_to\_hid + model.input\_to\_hid \* wd\_coefficient; %ret.input\_to\_hid Size(256, n\_hid)

ret.hid\_to\_class = ret.hid\_to\_class + model.hid\_to\_class \* wd\_coefficient; %ret.hid\_to\_class Size(n\_hid,10)

end

%Theta is a column vector that holds the weights

%Model contains two matrix (,) with the weights

function ret = theta\_to\_model(theta)

n\_hid = size(theta, 1) / (256+10);

ret.input\_to\_hid = transpose(reshape(theta(1: 256\*n\_hid), 256, n\_hid));

ret.hid\_to\_class = reshape(theta(256 \* n\_hid + 1 : size(theta,1)), n\_hid, 10).';

end

function ret = model\_to\_theta(model)

input\_to\_hid\_transpose = transpose(model.input\_to\_hid);

hid\_to\_class\_transpose = transpose(model.hid\_to\_class);

ret = [input\_to\_hid\_transpose(:); hid\_to\_class\_transpose(:)];

end

function ret = initial\_model(n\_hid)

n\_params = (256+10) \* n\_hid;

as\_row\_vector = cos(0:(n\_params-1));

ret = theta\_to\_model(as\_row\_vector(:) \* 0.1); % We don't use random initialization, for this assignment. This way, everybody will get the same results.

end

function ret = classification\_performance(model, data)

% This returns the fraction of data cases that is incorrectly classified by the model.

hid\_in = model.input\_to\_hid \* data.inputs; % input to the hidden units, i.e. before the logistic. size: <number of hidden units> by <number of data cases>

hid\_out = logistic(hid\_in); % output of the hidden units, i.e. after the logistic. size: <number of hidden units> by <number of data cases>

class\_in = model.hid\_to\_class \* hid\_out; % input to the components of the softmax. size: <number of classes, i.e. 10> by <number of data cases>

[dump, choices] = max(class\_in); % choices is integer: the chosen class, plus 1.

[dump, targets] = max(data.outputs); % targets is integer: the target class, plus 1.

ret = mean(choices ~= targets);

end

**Testing Function:**

function TestFunc(imageName)

%TestFunc gets the name of the image as an input and displays the

%probabilities of each number

%imageName name of the image to load

close all; %close all figures

%Load the model

load model; %load the model of the network

%Load the image and transform it

number=imread(imageName); %read the image file

imshow(number); %show the image

movegui('northwest');

number=im2double(number); %transform the format from 8bit to double

number = number'; %this is just to adapt to USPS database

data=number(:); %finally transform the matrix into a vector

%Compute the probabilities

probabilities = computeProbabilities(model, data);

%Show the probabilities in a graph

figure;

numbers = [ 0 1 2 3 4 5 6 7 8 9];

bar(numbers,probabilities,'FaceColor',[.36 .63 .28]);

grid on

movegui('northeast');

end

function ret = logistic(input)

ret = 1 ./ (1 + exp(-input));

end

function ret = log\_sum\_exp\_over\_rows(a)

% This computes log(sum(exp(a), 1)) in a numerically stable way

maxs\_small = max(a, [], 1);

maxs\_big = repmat(maxs\_small, [size(a, 1), 1]);

ret = log(sum(exp(a - maxs\_big), 1)) + maxs\_small;

end

function ret = computeProbabilities(model, data, wd\_coefficient)

% model.input\_to\_hid is a matrix of size (n\_hid,256)

% model.hid\_to\_class is a matrix of size (10,256)

% data.inputs is a matrix of size (256,<number of data cases>)

% data.outputs is a matrix of size (10,<number of data cases>)

% first, do the forward pass, i.e. calculate a variety of relevant values

hid\_in = model.input\_to\_hid \* data; % input to the hidden units, i.e. before the logistic. size: (n\_hid,<number of data cases>)

hid\_out = logistic(hid\_in); % output of the hidden units, i.e. after the logistic. size: (n\_hid,<number of data cases>)

class\_in = model.hid\_to\_class \* hid\_out; % input to the components of the softmax. size: (10, <number of data cases>)

class\_normalizer = log\_sum\_exp\_over\_rows(class\_in); % log(sum(exp)) is what we subtract to get normalized log class probabilities. size: (1,<number of data cases>)

log\_class\_prob = class\_in - repmat(class\_normalizer, [size(class\_in, 1), 1]); % log of probability of each class. size: (10, <number of data cases>)

class\_out = exp(log\_class\_prob); % probability of each class. Each column (i.e. each case) sums to 1. size: (10, <number of data cases>)

ret = class\_out;

end

%Theta is a column vector that holds the weights

%Model contains two matrix (,) with the weights

function ret = theta\_to\_model(theta)

n\_hid = size(theta, 1) / (256+10);

ret.input\_to\_hid = transpose(reshape(theta(1: 256\*n\_hid), 256, n\_hid));

ret.hid\_to\_class = reshape(theta(256 \* n\_hid + 1 : size(theta,1)), n\_hid, 10).';

end

function ret = model\_to\_theta(model)

input\_to\_hid\_transpose = transpose(model.input\_to\_hid);

hid\_to\_class\_transpose = transpose(model.hid\_to\_class);

ret = [input\_to\_hid\_transpose(:); hid\_to\_class\_transpose(:)];

end

**SCREENSHOTS**

Image used for Testing:

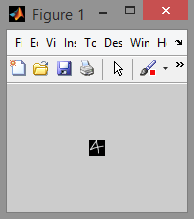
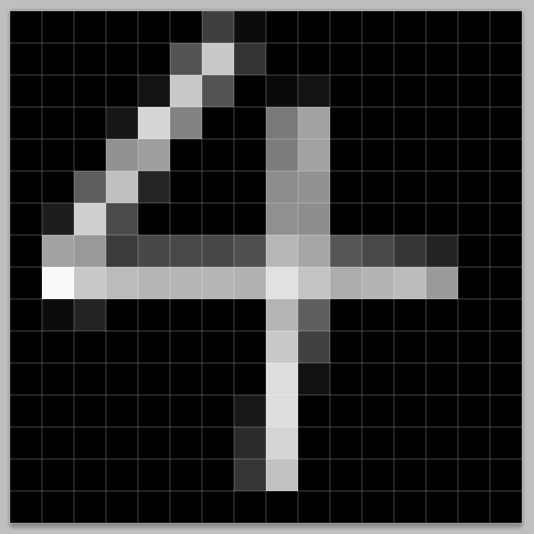
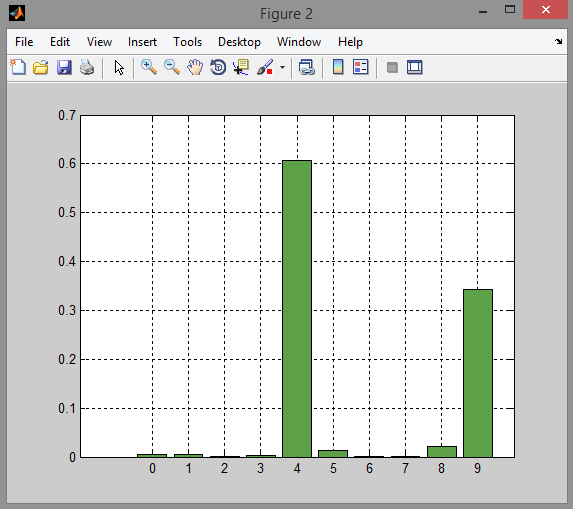


Image written in Photoshop:



Testing Results:



**CONCLUSION**

This experiment provided us with an interesting introduction to the problem of handwritten character recognition on a smaller and more feasible scale by attempting to recognize digits, which is simply a subset of possible characters.

**FUTURE ENHANCEMENTS**

One area of future work is to see what changes would have to be made to produce similar results for recognizing letters as well as digits. There would certainly have to be more nodes in the final layer, because each node represents a possible classification. As a result, a larger number of hidden nodes may need to be used in the layer before the output layer.

**REFERNCES**

* Programming Exercises IX. Neural Networks: Learning (Week 5) by Dr. Andrew Ng (<https://class.coursera.org/ml-004/assignment/view?assignment_id=6>)
* Fundamental of Neural Networks: Architecture, Algorithms and Applications by Laurene Fausette
* Discussion threads. Neural Networks by Geoffrey Hinton (<https://class.coursera.org/neuralnets-2012-001/forum/thread?thread_id=765>)