PROJECT 1: HAND WRITEN DIGITS RECOGNITION PROGRAM

ALI NEHRANI

Abstract. In this project, I implemented a neural network technique in order to recognize hand written digits. I used simple multiple layer neural network to address this problem an dtested it with min square error and cross entropy objective functions. Data hand written digits data is very important for training the network so I created more data with suported hand written digit creation function. In network I used around 20 to 30 hidden layers. Results are reported through figures.

1. Preprocessing

There are some data provided for this project with special structure 1×200. This amount of data is not enough to get 90% accuracy. This amount of accuracy requires more proper data. For generating more data provided Matlab function `getUserTraj.m` is applied. I write a function `DatasetMake.m` which request user to draw a digit at the given area (with mouse) then it added that to the dataset. Using this function I created 4300 data. I the following I will discuss the structure and the coverage of the dreated data.
Figure 1 presents the process of preparing training data. The training data is saved as a Matlab data structure `Data09.mat` and provided alonge the programs.
The structure of the `Data09` is a matrix 4300×201 where rows are training data for all digits $0-9$ and first 100 columns are the training datum's x-axis and columns $101-200$ are it's y-axis and column 201 is the target. The following simple Matlab function creats training data set:

```
dt=0.01
fig = figure(1)
if ~ exist('DataN');
   DataN = [];
end
target = input('enter the target you want to collect data: ');
for i=1:189
    [x,y] = getUserTraj(dt,fig);
    DataN = [DataN; [x y] target];
    y = input('Do you want to continue? ');
    if y == 'n'
        break
    end
end
```

Date: Dec. 2017.
Thanks for your time to read my report.

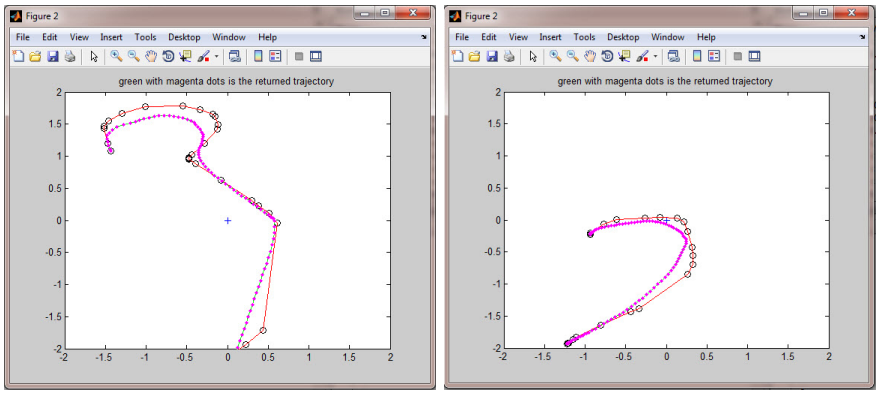

FIGURE 1. Building training data for hand written digits;Training digit "3" left, Training digit "7" right

FIGURE 2. Testing hand written data for scale adaptation - digit "6"

1.1. **Training Data.** Data is very important in increasing the accuracy of the method. The network should be scale and rotation invariant as well as ability to recognize digits that not recognizible by humans somehow (up to 90%). Theoreticaly there is a solution for scalling by making all data standard. This standardization is also supported by providing scalled data (scalled inputs) as it is shown in Figure 2. The following Matlab function is implemented for making data standard:

```
function [SData, mean_X, std_X] = standardize(varargin)
switch nargin
    case 1
        mean_X = mean(varargin{1});
        std_X = std(varargin{1});

        SData = varargin{1} - repmat(mean_X, [size(varargin{1}, 1) 1]);

        for i = 1:size(SData, 2)
            SData(:, i) = SData(:, i) / std(SData(:, i));
        end
    case 3
        mean_X = varargin{2};
        std_X = varargin{3};
        SData = varargin{1} - repmat(mean_X, [size(varargin{1}, 1) 1]);
        for i = 1:size(SData, 2)
            SData(:, i) = SData(:, i) / std_X(:, i);
        end
end
```

For Rotation, I used two tricks; first I applied mathematical rotation technique and rotated the data randomly and added those rotated data to the existing data (by adding these data the total amount of data became twice and also more). The Matlab function `rand_rotation.m` does the rotation

```
function r_data = rand_rotation(Data, rtheta)
   Data_x = Data(:,1:100);
   Data_y = Data(:,101:200);

   sinus   = sin(rtheta*pi/180);
   cosinus = cos(rtheta*pi/180);
   r_data =[Data_x*cosinus + Data_y*sinus - Data_x*sinus + Data_y *cosinus
      ...
            Data_x * cosinus + Data_y*sinus - Data_x*sinus + Data_y *
    cosinus];
end
```

The other technique that I applied to improve the ability of recognizing rotation is to providing ritated hand-written digits. This approach is also useful in training network to recognize bad written digits. In Figure 3 it was shown that what kind of data is feeded to the network.

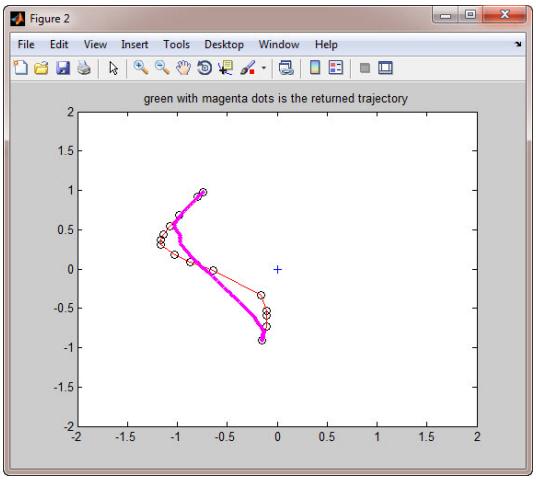

FIGURE 3. Training hand written data for scale adaptation - digit "7"

1.2. **Testing Data.** For creating test data I prepared a Matlab function `TestData.m` which gets the number of testing data and for that number; first it asks for the target (the number that use wants to write), second loads the `getUserTraj.m` for creating handwritten digit by the user, then adds this data to the testing data set and predicts the handwritten digit, compares it with the given target and returns the accuracy percentage. Figure 4 shows a process of creating test datat and the output of the program.
I created 4300 data in order to cover any kind of rotation and bad writing cases. The codes are provided in the following:

```
acprt=0;
if ~ exist('Tdata')
    Tdata = [];
end
for j = 1:n_test
    y_true = input('Enter true y: ')
```

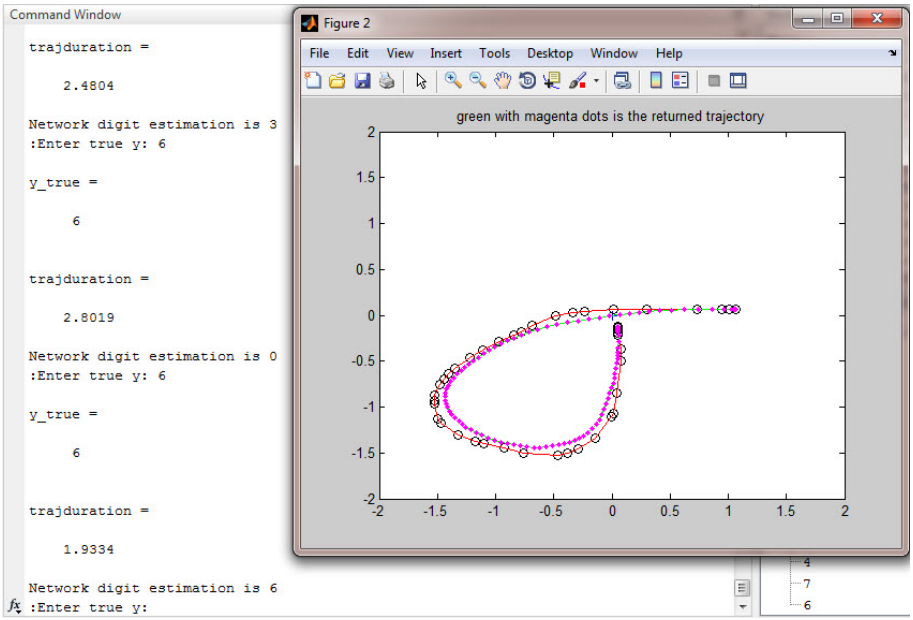

FIGURE 4. Testing hand written data and the prediction result of the program- digit "6"

```
    [x,y] = getUserTraj(dt, fig);
    test_data = [x y];
    Tdata=[Tdata; x y y_true];
    [O,Oh,V,Vh] = forwardPass(test_data,w,W);
    [o_m,O_i]=max(O);
    y_pred = O_i-1;
    fprintf('Network digit estimation is %d\n:', y_pred);

    if y_pred == y_true
      acprt = acprt + 1;
    end
end
fprintf('Testing accuracy equals to: %f\n',acprt*100/j );
```

In applying Cross-validation I separate 10% of the data for testing process. the cross validation results are provided in Table 1.

2. Neural Network

The structure Neural Network was the same as the provided multi layer perceptron as shown in the Figure 5. It consists of three layers (like all other networks) and the hidden layer itself has two layer. For trainig the hand written digits I applied 15 unites for the hidden layers. The input has 200 units and output has 10 units. For this there is two weight matrices; input to hidden layer w with size 200$\times$ and hidden to output layer weight W with size 30×10.
In the final part `Softmax` layer is included to improve the recognition. The softmax

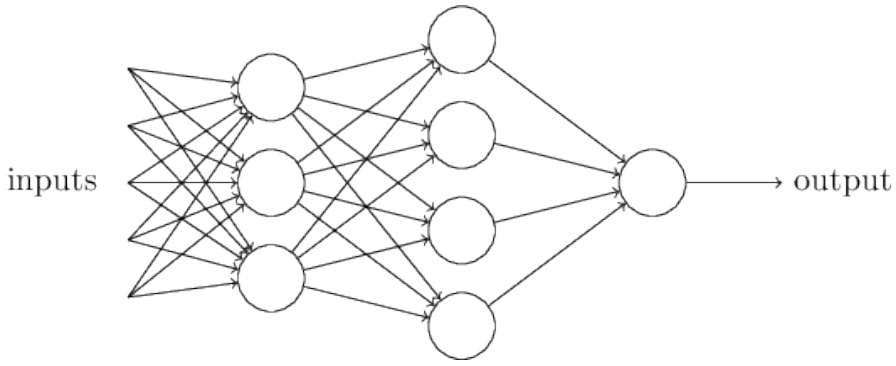

FIGURE 5. Structure of the Applied Network: $200 \times 15 \times 15 \times 10$

is formulated as:

$$O_i = \frac{\exp l_i}{\sum_i \exp l_i}$$

where l_i is the output of the *hidden-out* layer.

2.1. **Hyper Parameters of the Neural Network.** Neural networks have some hyperparameters which should be set befor training. There are strategies for some of them and some other are set based on experience. In applied neural network there are some hyper parameters in learning, perceptrons and also type of cost function. I listed the hyperparameters with their value in the following:

Number of hidden layers:	25
Maximum iteration:	2000000
learning rate eta (η):	0.000013
slack learning rate nu (ν):	0.0000107
Sigmoid parameter B:	1.8
Lealy RELU parameter α:	0.15
Random noise coefficient beta (β):	10^{-10}

I applied Nestrov version of the accelerated gradient diecent method which itself has two hyperparameters for the learning process which has two hyper parameters η and ν. η weights the gradient effect on the updating of the new weight and ν weights the guiding vector. I studied the effect of some parameters in learning.

2.2. **Activation Functions.** One of the most importan t part of the neural networks is activation functions. I applied three different activation functions in this project. Sigmoid activation function which is represented in Figure **??** with it's derivative. Sigmoid function is represented as

$$y = \frac{1}{1+\exp(-z)}, \qquad y' = y(1-y)$$

The Matlab implementation of these function is provided in the following

```
% activation function
function ret = sigmoid(x)
B = .8;
ret = 1./(1+exp(-B*x));
end
```

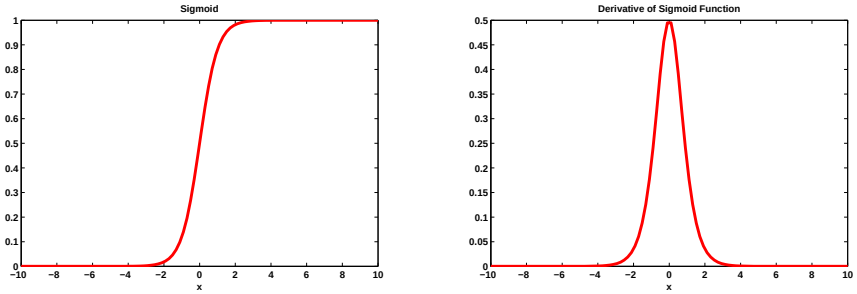

FIGURE 6. Sigmoid activation function (left) and it's derivative (right)

```
% derivative of sigmoid function
function ret = sigder(x)
B = .8;
ret = B*sigmoid(x).*(1-sigmoid(x));
end
```

The next activation function that I applied is Leaky RELU which is represented with it's derivative in Figure **??**. The mathematical formula of this activation function is

$$y = x \text{ if } x \geq 0, \quad y = \alpha x \text{ if } x < 0,$$

The Matlab implementation of these function is provided in the following

```
% leaky relu
function ret =lrel(x)
ret = x;
alpha = 0.19;
ret(find(x<0)) = alpha .* x(find(x<0));
end
% leaky relu der
function ret = lrelder(x)
ret = ones(size(x));
alpha = 0.19;
ret(find(x<0)) = alpha;
end
```

The next activation function that I applied is x-Sigmoid function which is represented in Figure **??**. This activation function is obtained by multiplying x to the sigmoid function to prevent ints dying effects. x-sigmoid is similar to the Leaky RELU. The Matlab implementation of these function is provided in the following

```
%x sigmoid
function ret = sigmoid(x)
B = 1.8;
ret = x./(1+exp(-B*x));
end
% x sig derivative
function ret = xsigder(x)
B = 1.8;
ret = x.*(B*sigmoid(x).*(1-sigmoid(x))) + 1./(1+exp(-B*x));
end
```

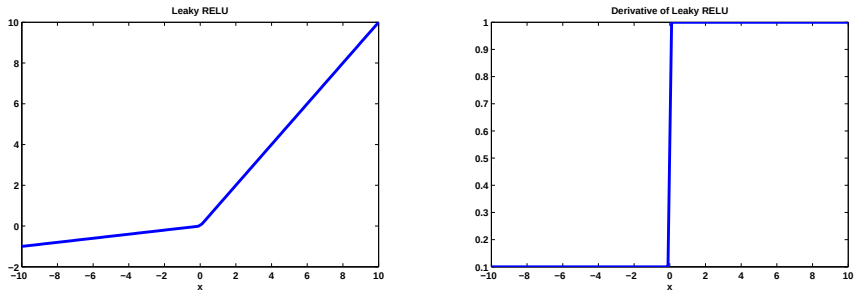

FIGURE 7. Leaky RELU activation function (left) and it's derivative (right) with parameter $\alpha = 0.2$

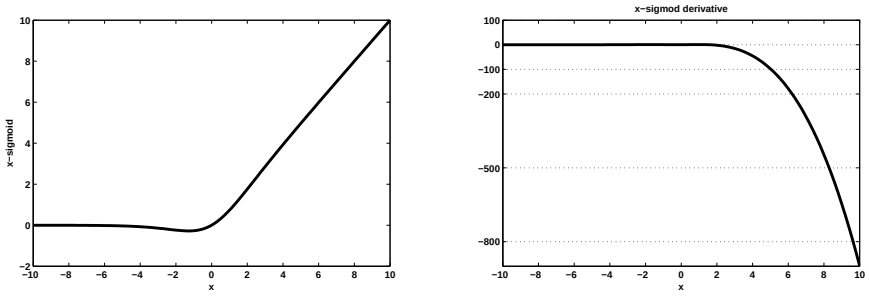

FIGURE 8. x-Sigmoid activation function (left) and it's derivative (right) with parameter $\alpha = 0.2$

3. Learning with Gradient Decent

For learning, we should consider to select cost function and optimization method. I introduced two type of cost functions: Cross entropy objective function and mean square error. Each of these cost function has propertiese and effective, for some problems cross entropy works better and for others mean square error. Mathematical formula of MSE is

$$MSE = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{2}\|y - \hat{y}\|_2^2$$

where y is target and $\hat{y}$ is prediction. In Figure 9 I plotted the responce of MSE for two random vectors. Matlab implementation of MSE and its derivative is in the following code box

```
if strcmp(cost, 'mse')
                E = 0.5*norm(T-O,2)^2;
                dW = -((T-O).*sigder(Oh))*V;
                dw = -((((T-O).*sigder(Oh))'*W).*sigder(Vh))'*x;
```

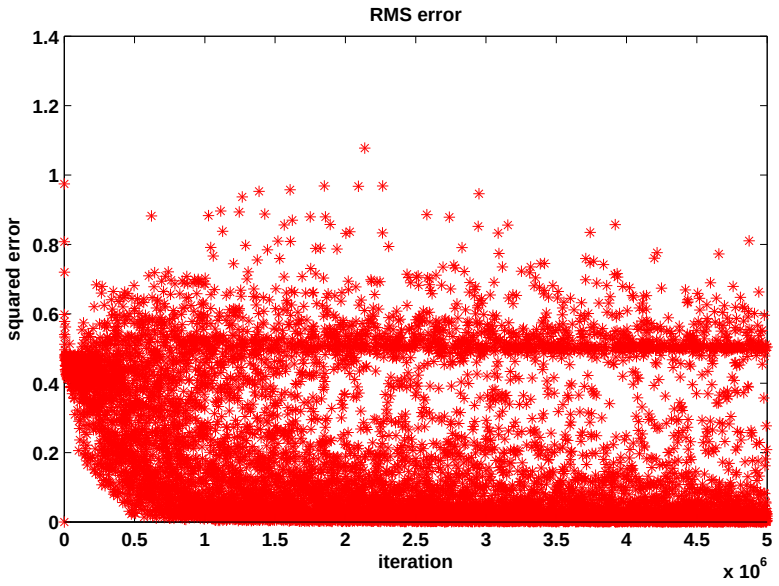

FIGURE 9. MSE function behaviour for random vectors uniformly distributed in [0 1]

Cross Entropy cost function is also one good measure that alternatively is used in data processing and machine learning. Mathematical formula of cross entropy is

$$CE = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

where N here is number of classes. The behaviour of cross entropy cost function for two random vectors and two classes is presented in Figure 10. Matlab implementation of cross entropy for two classes case and its derivative is in the following code box

```
% Cross validation objective function
E = -sum(T.*log10(O) + (1-T).*log10(1-O));
dW = -(1/(log2(10)))*((T.*sigder(Oh)./O)*V +((1-T).*sigder(Oh)./(1-O))*V);
      % my Tuning
dw = -(1/(log2(10)))*(((T.*sigder(Oh)./O)'*W).*sigder(Vh))'*x  + (1/(log2
    (10)))*((((1-T).*sigder(Oh)./(1-O))'*W).*sigder(Vh))'*x;
```

3.1. **Optimization and Backward propagation.** My optimization algorithm which Nestrov's accelerated gradient decent is implemented as:

```
if it==1
            vect_p = -eta*dw;
end;
% Accelerated Gradient
vect = mu*vect_p - eta*dw + beta*rand(nV,tr_n-1); %
w = w - mu * vect_p + (1+mu)*vect;
vect_p = vect;
```

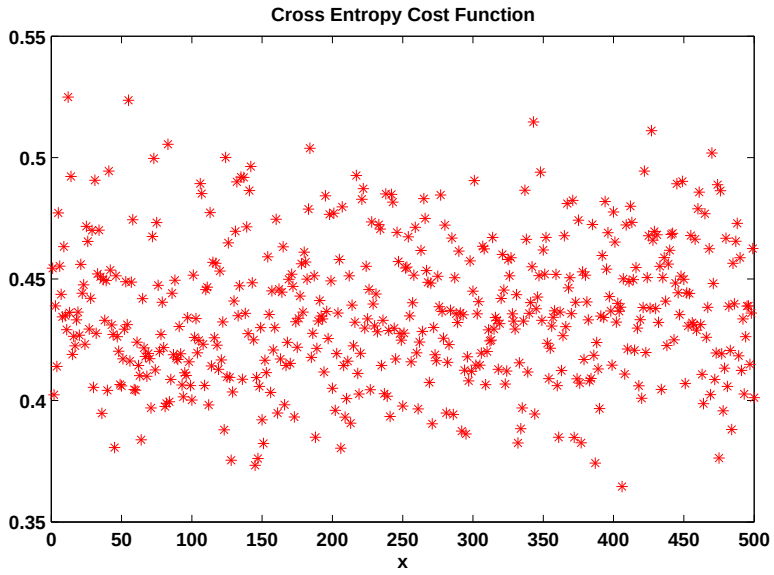

FIGURE 10. Cross entropy cost function behaviour for random vectors uniformly distributed in [0 1]

In this implementation, `eta` and `nu` are hyperparameters, `vect` and `vect_p` are guiding vectors and `nV` is number of hidden layers. As it is well known, Gradient decent based methods most of the times fall into local minima and do not converge for large data. Without convergence we can not reach to proper accuracy (especially the accuracy that is requested). To resolve this problem I usde rerunning trick which worked well.
In this trick after for example 10^6 iteration which no convergency is achieved, I saved the weights so that in the next run I used those trained weights. I did this many times untile I reached very small errors.

3.2. **Feedforward.** Feadforward process which is the same as lms code. In the feedforward we apply computed weights to predict the target.

```
% forward pass of neural network
function [O,Oh,V,Vh] = forwardPass(x,w,W)
Vh = x*w';
V = sigmoid(Vh);
    V(end) = 1;
Oh = W*V';
O = sigmoid(Oh);
end
```

4. Evaluating with Cross Validation

In order to evaluate the network properly, we should apply machine learning tools, at least the techniques that we learned from the course. Cross validation helps to evaluate the performance of the network. I applied 10 fold Cross validation.

(1) Data is divided into 10 parts (randomly saved in different mat files)

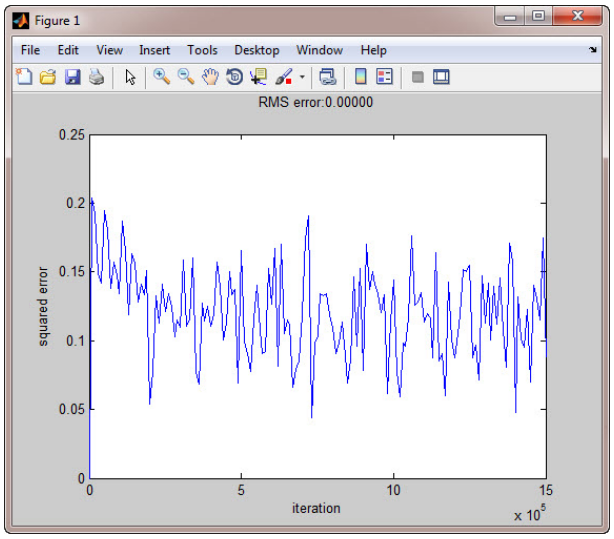

FIGURE 11. Error for multiple training- several times of training with 4300 data

(2) Each time 9 parts are applied for training and 1 is used for testing

In the following paper I provided accuracy results for each of 10 times train-test process:

Table 1: Cross-validation results with optimal number of layers and many iterations

Training	1	2	3	4	5	6	7	8	9	10
Achieved accuracy	95%	92%	85%	80%	91%	96%	89%	80%	87%	94%

How ever the program is written so that manuly I can change number of folds. Results of Table 1 the results are shown for more that 100×10^6 and using 25 hidden layers (which is optimal number of hidden layers according to cross validation). The out put of the program is

```
Error in fold 1
is equal to 95.220141 percent
cost =mse
Error in fold 2
 is equal to 92.580796 percent
cost =mse
Error in fold 3
 is equal to 84.840749 percent
cost =mse
Error in fold 4
 is equal to 80.435597 percent
cost =mse
Error in fold 5
 is equal to 91.346604 percent
cost = mse
Error in fold 6
 is equal to 96.409836 percent
cost =mse
Error in fold 7
```

```
 is equal to 89.960187 percent
cost = mse
Error in fold 8
 is equal to 80.814988 percent
cost = mse
Error in fold 9
 is equal to 87.112412 percent
cost =mse
Error in fold 10
 is equal to 94.346604 percent
Average Error in 10 folds is: 89.3068
```

In the report I reduced number of iterations into 10000 in order to reduce running time. In this amount of iterations I can not reach acceptable accuracy, but in order to demonstrate how cross validation works for evaluation the model I applied this low amount of iterations. I applied cross validation in order to obtain optimal amount of hidden layers. The Results of cross validation with small number of iterations (10000) is shown in Figure 12. According to this figure optimal number of hidden layers are about 25.

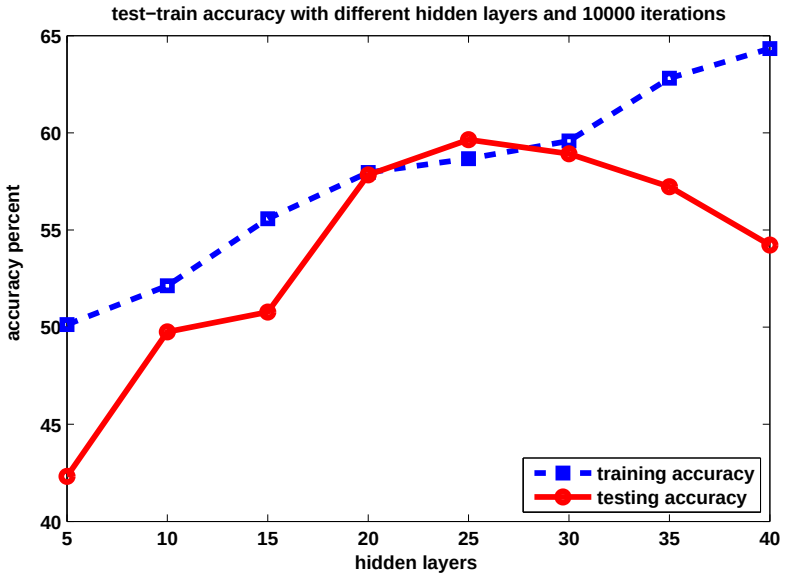

FIGURE 12. Cross entropy results for obtaining optimal number of hidden layers

5. CONCLUSION

This project was challenging to get more that 90% accuracy with small amount of data 6000 and it requires to apply many techniques. Beside If we increase number of data, convergence become problematics and using gradient decent type methods most of the times fall in local minimas. So we have some technical challenges from this type for which I applied tricks to resolve them.

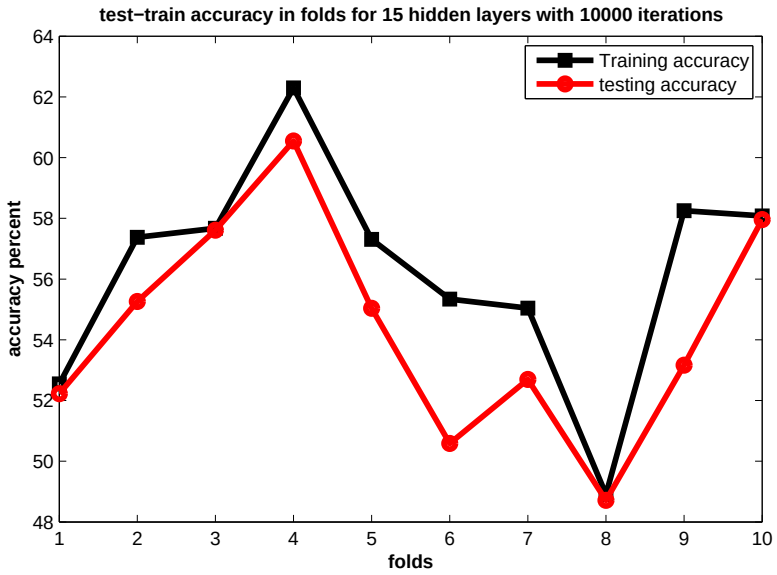

FIGURE 13. Results of accuracy in different folds with 15 hidden layers

Another important issue that we should notice is creating data carefully to cover many possible cases. Putting all toghether helped me to achieve this amount of accuracy.
There are other technical methods, including application of SVM method or other data processing techniques also can help to improve the accuracy even more, but with this amount of data I dont think that they can improve the significantly and for my case results are obtained after many times training.

6. APPENDIX: MATLAB IMPLEMENTATIONS

Main Part of the Code

```
%
% main  This  program do cross validation
%
clear all
close all
clc

%% Define parameters
MAXIT = 200000;      % maximum iteration of training
nV = 25;               % number of hidden units
batch_size = 10;  % batch size
beta = 1e-10;  % randomizer
mu = 0.0001;  % optimization parameter
eta =0.0001;      % learning rate
fold = 10;   % number of folds
%% Cross Validation data splitting
```

```
%try
%    data = [struct2cell(load('Data09.mat')){:}]; % IMPORTANT please Uncomment this when using OCTAVE
%catch
    tr_tg = load('Data09.mat');              % IMPORTANT please Uncomment this when using MATLAB
    data = struct2array(tr_tg); clear tr_tg % IMPORTANT please Uncomment this when using MATLAB
%end
%
%
%load weight1.mat  % In the case of testing ( This loads Trained weights)
%%
[m_data, n_data] = size(data);
% ------------------------------------    using 10-fold cross validation    ---------------------
total = linspace(1,m_data,m_data);
test_sel = total;

Train_data_ad = {}; Test_data_ad = {};
% number of folds
test_num = floor(m_data/fold);
% Cross validation spliting data - the address of data stored only
for n_fold = 1:fold
    T_sample = randi(size(test_sel,2),test_num,1); % selecting 10% of the data randomly
    Test_data_ad{n_fold} = T_sample;
    Train_data_ad{n_fold} = setdiff(total,T_sample);
    test_sel = setdiff(test_sel,T_sample);
end
clear  T_sample test_num
%% Iteration on the folds
E_fold = [];
for n_fold = 1:fold
%% training process
% select type of the cost function:
%cost = 'cross'
cost = 'mse'
%cost = input('which kind of cost function do you prefer: enter "cross" for cross entropy and "mse" for mean square error: ')
[W,w] = NN_train(data(Train_data_ad{n_fold},:), cost, nV, eta, batch_size, beta, mu, MAXIT);

%% Testing Process
    n_true = 0;
    test_data = data(Test_data_ad{n_fold},:);  % data is loaded from test fold
    for i=1:size(test_data,1)
        [O,Oh,V,Vh] = forwardPass(test_data(i,1:end-1),w,W);
        [o_m,O_i]=max(O);
        y_pred = O_i-1;
        if y_pred == test_data(i,end)
            n_true = n_true + 1;
```

```
        end
    end
    E_fold = [E_fold n_true/size(test_data,1)*100];
    fprintf('Test accuracy in fold %d is equal to: %f percent \n', n_fold,
     n_true/size(test_data,1)*100);

% clculating training accuracy
train_data = data(Train_data_ad{n_fold},:);
E_train = [];
n_true = 0;
    for i=1:size(train_data,1)
        [O,Oh,V,Vh] = forwardPass(train_data(i,1:end-1),w,W);
        [o_m,O_i]=max(O);
        y_pred = O_i-1;
        if y_pred == train_data(i,end)
            n_true = n_true + 1;
        end
    end
    E_train = [E_train n_true/size(train_data,1)*100];
    fprintf('Train accuracy in fold %d is equal to: %f percent \n', n_fold
    , n_true/size(train_data,1)*100);
clear W w  test_data
end
fprintf('Average Test accuracy in %d folds is: %f \n', fold, mean(E_fold))
    ;
fprintf('Average Train accuracy in %d folds is: %f \n', fold, mean(E_train
    ));
```

Training Neural Network

```
%Handwritten Recognition Project
% I did the simple one just for zero, next time for other digits too
% text time testing to see the performance if it was qualified
% thinking about possible impovements
% thinking about the demo
%
%
function [W, w] = NN_train(tr_data, cost, nV, eta, batch_size, beta, mu,
    MAXIT)
% ENTER YOUR NAME HERE:
%% preparing the data
% preprocessing the data
% determine which data to load

[tr_m, tr_n] = size(tr_data);
tr_data = [standardize(tr_data(:,1:end-1),0.5*ones(1,tr_n-1),0.5*ones(1,
    tr_n-1)) tr_data(:,end)];

%% initial weights
    ....................................................
    if ~(exist('W') && exist('w'))
        W = (rand(10,nV)-0.5)*0.05;    % hidden->output weights
        w = (rand(nV,tr_n-1)-0.5)*0.01;  % input->hidden weights
    end;
```

```
22
23 %% initial parameters
   ..................................................
24     x = zeros(1,tr_n-1);   % input layer
25     V = zeros(nV,tr_n-1);  % hidden layer
26 %    O = 0;                 % output layer
27 %% setting the parameters
28     figure(1); clf;
29     Mean_E = 0;
30     err = zeros(MAXIT,1);
31     for it = 1:MAXIT,
32         tr_T = randi(tr_m,batch_size,1);
33         x = tr_data(tr_T,1:tr_n-1);
34         T = zeros(10,size(tr_T,1));
35         % design target
36         for i=1:batch_size
37          T(tr_data(tr_T(i),tr_n)+1,i)=1;     % training target, temporary
   for now
38         end
39         [O, Oh, V, Vh] = forwardPass(x,w,W);
40         %
41         % reshaping
42         Oh = reshape(Oh, 10*batch_size,1);
43         O = reshape(O, 10*batch_size,1);
44         T = reshape(T, 10*batch_size,1);
45         if strcmp(cost, 'mse')
46              E = 0.5*norm(T-O,2)^2 ;        % SUM SQUARED ERROR
47              % MSE derivative case
48              dW = -reshape((T-O).*relder(Oh),10,batch_size)*V;    % SUM
   SQUARED ERROR BASED dW
49              dw = -((reshape(((T-O).*relder(Oh)),batch_size,10)*W).*relder
   (Vh))'*x ;%+ 0.04*w; % SUM SQUARED ERROR BASED dw
50
51          elseif strcmp(cost, 'cross')
52 %         % Cross validation objective function
53 %              E = -sum(T.*log10(O) + (1-T).*log10(1-O));
54 %              dW = -(1/(log2(10)))*((T.*sigder(Oh)./O)*V +((1-T).*sigder(
   Oh)./(1-O))*V);  % my Tuning
55 %              dw = -(1/(log2(10)))*(((T.*sigder(Oh)./O)'*W).*sigder(Vh))
   '*x  + (1/(log2(10)))*((((1-T).*sigder(Oh)./(1-O))'*W).*sigder(Vh))'*x
   ; % My tuning
56         end
57         Mean_E =  Mean_E + ((sqrt(E)/batch_size - Mean_E)/MAXIT);
58         err(it+1)= sqrt(Mean_E)/batch_size;
59         if (mod(it,10000)==0)    % show learnig progress
60             cla;
61             plot(0:10000:it,err(1:10000:it+1));
62             xlabel('iteration');
63             ylabel('squared error');
64             drawnow;
65         end;
66 %
67   if it==1
```

```
68          vect_p1 = -eta*dW;
69    end;
70        vect1 = mu*vect_p1 - eta*dW ; %
71       W = W - mu * vect_p1 + (1+mu)*vect1;
72        vect_p1 = vect1;
73 %
74  if it==1
75      vect_p = -eta*dw;
76  end;
77     % Accelerated Gradient
78          vect = mu*vect_p - eta*dw + beta*rand(nV,tr_n-1); %
79          w = w - mu * vect_p + (1+mu)*vect;
80          vect_p = vect;
81
82     end %iteration
83    if (it > 9000)
84     plot((0:10000:length(err)-1),err(1:10000:end));
85     xlabel('iteration');
86     ylabel('squared error');
87     drawnow;
88    end
89     ms_err = 0;
90     ms_err = ms_err/tr_m;
91     rms_err = ms_err^0.5;
92     if mod(it,5000)==0
93        %fprintf('RMS error over data points:%3.5f\n',rms_err);
94         title(sprintf('RMS error:%3.5f\n',rms_err));
95
96     end
97 %%
98 end % end for trainer
99 %end  % main
```

Department of Computer Science, Ozyegin University
E-mail address: ali.nehrani@ozu.edu.tr