ECE 559 – Neural Networks

Character Recognition using Backpropagation Neural Network (BPNN)

Assignment – I

Arindam Bose
UIN: 665387232
09-17-2015

1. *Abstract*

Advancement in Artificial Intelligence has led to the developments of various 'smart' devices. Character recognition is one of the most widely used biometric traits for authentication of person as well as document. In this assignment a Back propagation Neural network (BPNN) is designed to model the way in which a few characters are recognized. Each image character is comprised of 8×8 pixels. These inputs pixels are given to a back propagation neural network with a hidden layer and output layer. I have used the BPNN for efficient recognition where the errors were corrected through back propagation and rectified neuron values were transmitted by feed-forward method in the neural network of multiple layers.

2. *Problem Statement*

Use Back Propagation neural network (BPNN) for character recognition to recognize 3 characters of your choice (for example, 'A', 'C', and 'N'). Use 8x8 grid to define your characters. The network should also be able to identify 'other' character i.e. any character other than the ones used for training should be classified as 'other'. So, in all, your BPNN should be able to classify any 8x8 character into one of the four groups ('A','C','N' or 'other').

Test your network for characters with 0-bit, 1-bit, 2-bit and 3-bit noise.

Also, test it for characters other than the 3 characters used for training.

3. *Description of Design of the BPNN*

a. Network Structure: There are 4 layers: Input Layer, 2 Hidden Layers and Output Layer

- The Input Layer consists of the 64 Neurons which takes 8x8 grid of characters as input.
- Both the Hidden Layers consists of 2 Neurons each.
- Also the Output Layer consists of 2 neurons which can perfectly represent 3 different characters viz. [0,1] as letter 'A', [1,0] as letter 'C' and [1,1] for letter 'N'. We refrain from choosing large number of Neurons to avoid the unnecessary complexity in calculation.

There are 64 inputs to the network. In this particular case the sigmoid function: $y = \frac{1}{1+\exp(-z)}$ is chosen as a nonlinear neuron activation function. Bias terms (equal to 1) with trainable weights were also included in the network structure.

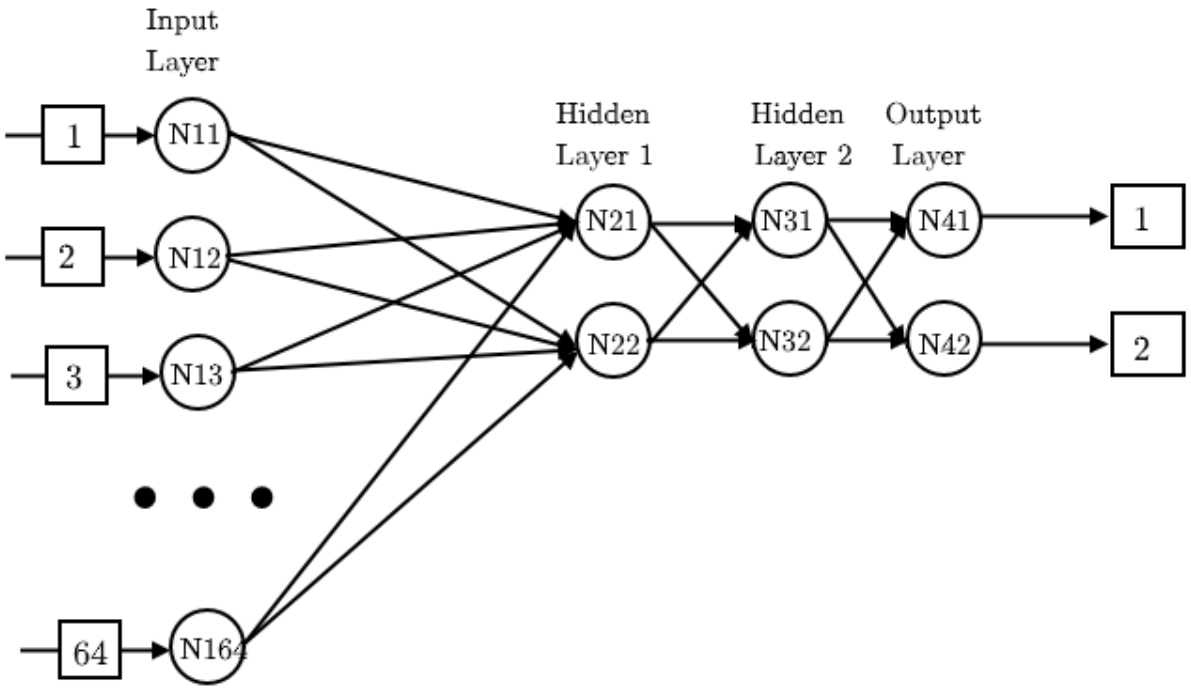

Fig. 1. Schematic design of the back-propagation neural network

b. Dataset Design: In this BPNN we teach the network to recognize characters 'A', 'C' and 'N'. To train the network to produce error signal we will use another 3 characters 'X', 'Y', 'Z'. We are interested in checking the response of the network to errors on the characters which were not involved in the training procedure. The characters to be recognized are given on an 8×8 grid. Each of the 64 pixels is set to either 0 (white pixels) or 1 (black pixels) as in Fig. 2.

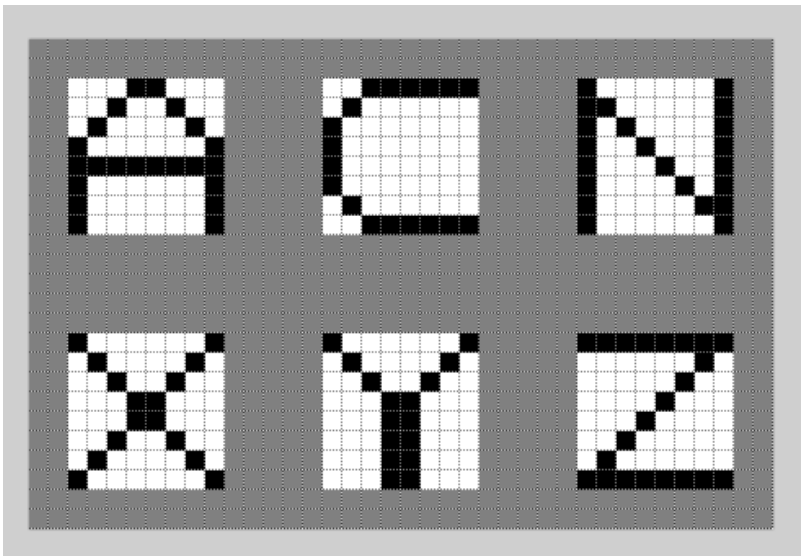

Fig. 2. Dataset: 'A', 'C', 'N', 'X', 'Y', 'Z'

4. *Results*

a. Network Training: To train the network to recognize the above characters we applied the corresponding 8×8 grids in the form of 1×64 vectors to the input of the network. The character was considered recognized if both outputs of the network were no more than 0.1 off their respective desired values. The initial learning rate η was experimentally set to 1.5 and was decreased by a factor of 2 after each 100^{th} iteration. It was also reset to its initial value after 400^{th} iteration. After around 1000 iterations I was able to converge to zero error and correctly recognize all the characters. 1000 iterations took place in about 0.61019 secs.

At this point the output of the last layer was as below:

```
Training Set: 1:[0.041467][0.99936]: Passed
Training Set: 2:[0.99959][0.044494]: Passed
Training Set: 3:[0.96031][0.95829]: Passed
```

Training set 1, 2, 3 corresponds to letter 'A', 'C', 'N' respectively. This program optimizes the error function in each iteration. The more is the number of iteration, the less is the error as depicted in Fig.3. This is how the weights converge after a large learning time.

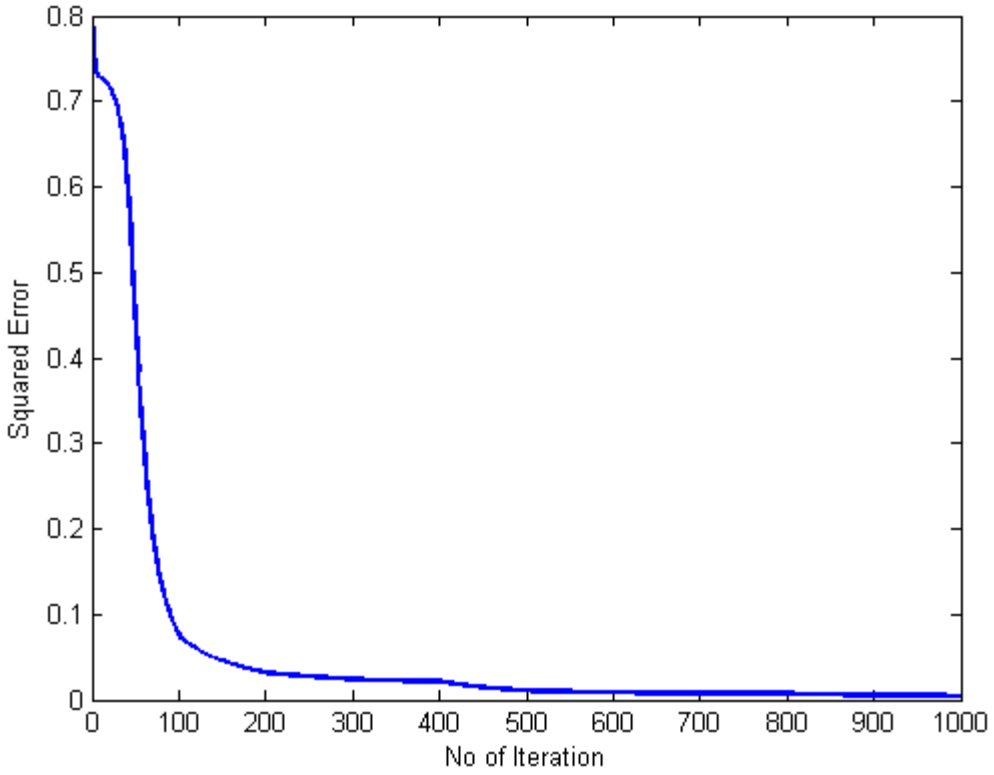

Fig. 3. Graph: Squared error vs Number of iteration.

b. Recognition Results: In order to determine if error detection is performed correctly, I tried to recognize other untrained characters with the same trained weights. I saved the obtained weights into a data file and then tested 'X', 'Y' and 'Z' instead of 'A', 'C', 'N'. I got the following results: Note that they are recognized as 'Other' character because they don't belong to any of the [{0, 1}, {1, 0}, {1, 1}] output category.

```
Test Character: X
N/W Output: [0.96076][0.88902]
Recognized Character: Other

Test Character: Y
N/W Output: [0.98896][0.22201]
Recognized Character: Other

Test Character: Y
N/W Output:[0.99289][0.72]
Recognized Character: Other
```

c. Robustness: To investigate the robustness of the neural network, I added several noise bits to the input and got the following results. I tested with 1, 2 and 3-bit noises:

```
Number of error bits (1-3)? 1
With 1 false bit per character:
Training Set 1: 64/64 recognitions (100%)
Training Set 2: 64/64 recognitions (100%)
Training Set 3: 64/64 recognitions (100%)
Total time elapsed : 0.020071 secs

Number of error bits (1-3)? 2
With 2 false bit per character:
Training Set 1: 4032/4032 recognitions (100%)
Training Set 2: 4032/4032 recognitions (100%)
Training Set 3: 4022/4032 recognitions (99.752%)
Total time elapsed : 0.9788 secs

Number of error bits (1-3)? 3
With 3 false bit per character:
Training Set 1: 249984/249984 recognitions (100%)
Training Set 2: 249984/249984 recognitions (100%)
Training Set 3: 246522/249984 recognitions (98.6151%)
Total time elapsed : 61.2535 secs
```

From the results it is evident that the network works fairly well even for 3-bit error. The poorest recognition rate it gives is 98.6151%.

5. *Conclusions*

The BPNN discussed here performed fairly well with both noise and no-noise dataset. Also, it was able to detect all the False Input correctly. Also the training time with 3 different training input for 1000 iterations was 0.61019 seconds which is fairly small. Testing time was also small, averaging 0.02169 seconds for each character. In the following assignments I would like to experiment using more number of neurons in the hidden layer and then compare my results.

6. *References*

[1] Graupe D., "Principles of Artificial Neural Networks – 3ed," ch 6, pp. 59-67, 2013.

7. *Source Code (MATLAB R2013b)*

a. main.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECE 599 Neural Networs                                                  %
% Name: Arindam Bose                                                      %
% UIN: 665387232                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Implementation of Character Recognition system using
Backpropagation Neural Network
%% Initialization
global NoOfExemplar NoOfInput NoOfOutput NoOfLayer Exemplar
TrueOutput config w z y Fi Rate Thresh;
Initialization();

%% Main menu
first = true;
while first
   clc;
   disp('------------- Main Menu -------------');
   disp('1: Apply weights and print output');
   disp('2: Train network');
   disp('3: Print current weights and parameters');
   disp('4: Save weights');
   disp('5: Load weights');
   disp('6: Recognition with noise');
   disp('7: Recognition with different input without noise');
   disp('0: Exit');
   ch = input('Your choice: ', 's');
   switch ch
        case '1'
            ApplyWeights(); % Apply weights and print output
        case '2'
            n = input('How many iteration? ');
            TrainNetwork(n); % Train network
            disp('Training Complete...');
        case '3'
```

```
            PrintWeights(); % Print current weights and parameters
        case '4'
            SaveWeights(); % Save weights to text file
            disp('Weights are saved to the file...');
        case '5'
            LoadWeights(); % Load weights from text file
            disp('Weights are loaded from the file...');
        case '6'
            n = input('Number of error bits (1-3)? ');
            RecognitionWithError(n); % Recognition with noise
        case '7'
            n = input('Test character (A/C/N/X/Y/Z)? ', 's');
            Testing(n); % Recognition with different input without
noise
        case '0'
            first = false;
            disp('Exiting...');
    end
    disp('Press any key to continue...');
    pause;
end
clear all;
```

b. Initialization.m

```
function Initialization()
% Initialization of global variables and initial weights
global NoOfExemplar NoOfInput NoOfOutput NoOfLayer Exemplar
TrueOutput config w Rate Thresh;

NoOfExemplar = 3;
NoOfInput = 64;
NoOfOutput = 2;
NoOfLayer = 3;
Thresh = 0.1;
Rate = 1.5;

config = [NoOfInput, 2, 2, NoOfOutput];
ExemplarA = [ 0, 0, 0, 1, 1, 0, 0, 0, ... % 'Letter A'
              0, 0, 1, 0, 0, 1, 0, 0, ...
              0, 1, 0, 0, 0, 0, 1, 0, ...
              1, 0, 0, 0, 0, 0, 0, 1, ...
              1, 1, 1, 1, 1, 1, 1, 1, ...
              1, 0, 0, 0, 0, 0, 0, 1, ...
              1, 0, 0, 0, 0, 0, 0, 1, ...
              1, 0, 0, 0, 0, 0, 0, 1];
ExemplarC = [ 0, 0, 1, 1, 1, 1, 1, 1,  ... % 'Letter C'
              0, 1, 0, 0, 0, 0, 0, 0, ...
              1, 0, 0, 0, 0, 0, 0, 0, ...
              1, 0, 0, 0, 0, 0, 0, 0, ...
              1, 0, 0, 0, 0, 0, 0, 0, ...
              1, 0, 0, 0, 0, 0, 0, 0, ...
              0, 1, 0, 0, 0, 0, 0, 0, ...
              0, 0, 1, 1, 1, 1, 1, 1];
ExemplarN = [ 1, 0, 0, 0, 0, 0, 0, 1,  ... % 'Letter N'
              1, 1, 0, 0, 0, 0, 0, 1, ...
              1, 0, 1, 0, 0, 0, 0, 1, ...
```

```
                1, 0, 0, 1, 0, 0, 0, 1, ...
                1, 0, 0, 0, 1, 0, 0, 1, ...
                1, 0, 0, 0, 0, 1, 0, 1, ...
                1, 0, 0, 0, 0, 0, 1, 1, ...
                1, 0, 0, 0, 0, 0, 0, 1];
ExemplarX = [ 1, 0, 0, 0, 0, 0, 0, 1,  ... % 'Letter X'
                0, 1, 0, 0, 0, 0, 1, 0, ...
                0, 0, 1, 0, 0, 1, 0, 0, ...
                0, 0, 0, 1, 1, 0, 0, 0, ...
                0, 0, 0, 1, 1, 0, 0, 0, ...
                0, 0, 1, 0, 0, 1, 0, 0, ...
                0, 1, 0, 0, 0, 0, 1, 0, ...
                1, 0, 0, 0, 0, 0, 0, 1];
ExemplarY = [ 1, 0, 0, 0, 0, 0, 0, 1,  ... % 'Letter Y'
                0, 1, 0, 0, 0, 0, 1, 0, ...
                0, 0, 1, 0, 0, 1, 0, 0, ...
                0, 0, 0, 1, 1, 0, 0, 0, ...
                0, 0, 0, 1, 1, 0, 0, 0, ...
                0, 0, 0, 1, 1, 0, 0, 0, ...
                0, 0, 0, 1, 1, 0, 0, 0, ...
                0, 0, 0, 1, 1, 0, 0, 0];
ExemplarZ = [ 1, 1, 1, 1, 1, 1, 1, 1,  ... % 'Letter Z'
                0, 0, 0, 0, 0, 0, 1, 0, ...
                0, 0, 0, 0, 0, 1, 0, 0, ...
                0, 0, 0, 0, 1, 0, 0, 0, ...
                0, 0, 0, 1, 0, 0, 0, 0, ...
                0, 0, 1, 0, 0, 0, 0, 0, ...
                0, 1, 0, 0, 0, 0, 0, 0, ...
                1, 1, 1, 1, 1, 1, 1, 1];

Exemplar = [ExemplarA; ExemplarC; ExemplarN; ...
            ExemplarX; ExemplarY; ExemplarZ];

TrueOutput = [[0,1]; [1,0]; [1,1]];

for i = 1:1:NoOfLayer
    for j = 1:1:config(i+1)
        for k = 1:1:config(i)+1
            w{i}(j,k) = rand(1) - 0.5;
        end
    end
end
end
```

c. ApplyInput.m

```
function ApplyInput(in)
%% Apply input to the network with existing weight vector
global NoOfLayer Exemplar config w z y;

for i = 1:1:NoOfLayer % Counting layers
    for j = 1:1:config(i+1) % Counting neurons in each layer
        z(i,j) = 0;
        for k = 1:1:config(i) % Counting input to each layer (= # of
neurons in the previous layer)
            if (i == 1) % If the layer is not the first one
```

```
                inputdata = Exemplar(in,k);
            else
                inputdata = y(i-1,k);
            end
            z(i,j) = z(i,j) + w{i}(j,k) * inputdata;
        end
        z(i,j) = z(i,j) + w{i}(j,config(i)+1); % Bias term
        y(i,j) = NonLinear(z(i,j));
    end
end
end
```

d. ApplyWeights.m

```
function ApplyWeights()
%% Apply existing weights and prints the output of the network
global NoOfExemplar NoOfLayer TrueOutput config y Thresh;

for in = 1:1:NoOfExemplar % Counting number of datasets
    ApplyInput(in);
    str = ['Training Set: ' num2str(in) ':'];
    for j = 1:1:config(4) % Counting neurons in the output layer
        str = strcat(str, '[', num2str(y(NoOfLayer,j)), ']');
    end
    if(y(NoOfLayer,1) > (TrueOutput(in,1) - Thresh) ...
    && y(NoOfLayer,1) < (TrueOutput(in,1) + Thresh) ...
    && y(NoOfLayer,2) > (TrueOutput(in,2) - Thresh) ...
    && y(NoOfLayer,2) < (TrueOutput(in,2) + Thresh))
            decision = 'Passed';
        else
            decision = 'Failed';
    end
    disp([str ': ' decision]);
end
end
```

e. NonLinear.m

```
function [y] = NonLinear(z)
%% Activation function
    y = 1/(1 + exp(-z));
end
```

f. TrainNetwork.m

```
function TrainNetwork(num)
%% Training of network num # of iteration
global NoOfExemplar NoOfLayer Exemplar TrueOutput config w y Fi Rate;

fileID = fopen('error.txt','wt');
tic;
RateTemp = Rate; % Starting learning rate
```

```
for it = 1:1:num % Going through all tr training runs
    sqErr = 0; % Each training run consists of runs through each
training set
    for in = 1:1:NoOfExemplar
        ApplyInput(in);
        for i = NoOfLayer:-1:1 % Counting the layers down
            for j = 1:1:config(i+1) % Counting neurons in the layer
                if(i == NoOfLayer) % If it is the output layer
                    multi = TrueOutput(in,j) - y(i,j);
                else
                    multi = 0;
                    for k = 1:1:config(i+2) % Counting neurons in the
following layer
                        multi = multi + Fi(i+1,k) * w{i+1}(k,j);
                    end
                end
                Fi(i,j) = y(i,j) * (1-y(i,j)) * multi;
                for k = 1:1:config(i) % Counting input to each layer
(= # of neurons in the previous layer)
                   if (i == 1) % If it is a first layer
                       prevOutput = Exemplar(in,k);
                   else
                       prevOutput = y(i-1,k);
                   end
                   w{i}(j,k) = w{i}(j,k) + Rate * Fi(i,j)*
prevOutput;
                end
                % Bias weight correction
                w{i}(j,config(i)+1) = w{i}(j,config(i)+1) + Rate *
Fi(i,j);
            end
        end
        sqErr = sqErr + (y(NoOfLayer,1) - TrueOutput(in,1))^2 + ...
               (y(NoOfLayer,2) - TrueOutput(in,2))^2;
    end
    fprintf(fileID, '%f\n' , 0.5*sqErr);
    if(~rem(it,100)) % Decrease learning rate every 100th iteration
         Rate = Rate/2;
    end
    if(~rem(it,400)) % Go back to original learning rate every 400th
iteration
         Rate = RateTemp;
    end
end
t=toc;
disp(['Total training time elapsed : ' num2str(t) ' secs']);
fclose(fileID);
end
```

g. LoadWeights.m

```
function LoadWeights()
%% Loads weithts from a file
global NoOfLayer config w;
```

```
fileID = fopen('weights.txt','r');
in = fscanf(fileID,'%f');
p = 1;
for i = 1:1:NoOfLayer % Counting layers
    for j = 1:1:config(i+1) % Counting neurons in each layer
        for k = 1:1:config(i) % Counting input to each layer (= # of
neurons in the previous layer)
            w{i}(j,k) = in(p);
            p = p + 1;
        end
    end
end
fclose(fileID);
end
```

h. SaveWeights.m

```
function SaveWeights()
%% Saves weiths to a file
global NoOfLayer config w;
fileID = fopen('weights.txt','wt');
for i = 1:1:NoOfLayer % Counting layers
    for j = 1:1:config(i+1) % Counting neurons in each layer
        for k = 1:1:config(i) % Counting input to each layer (= # of
neurons in the previous layer)
            out = w{i}(j,k);
            fprintf(fileID, '%f\n', out);
        end
    end
end
fclose(fileID);
end
```

i. PrintWeights.m

```
function PrintWeights()
%% Prints complete information about the network
global NoOfLayer config w z y;

for i = 1:1:NoOfLayer % Counting layers
    disp(['Layer ' num2str(i)]);
    for j = 1:1:config(i+1) % Counting neurons in each layer
        disp(['Neuron ' num2str(i)]);
        for k = 1:1:config(i) % Counting input to each layer (= # of
neurons in the previous layer)
            disp(['w[' num2str(i) '][' num2str(j) '][' num2str(k)
']=' num2str(w{i}(j,k))]);
        end
        disp(['w[' num2str(i) '][' num2str(j) '][BIAS]='
num2str(w{i}(j, config(i)+1))]);
        disp(['z[' num2str(i) '][' num2str(j) ']=' num2str(z(i,j))]);
        disp(['y[' num2str(i) '][' num2str(j) ']=' num2str(y(i,j))]);
    end
end
```

```
end
```

j. RecognitionWithError.m

```
function RecognitionWithError(num)
%% Recognition statistics for 1, 2 and 3 false bit cases
global NoOfExemplar NoOfLayer Exemplar TrueOutput config y Thresh;

switch num
    case 1
        tic
        disp('With 1 false bit per character:');
        TotalCases = config(1);
        for in = 1:1:NoOfExemplar % Looking at each dataset
            count = 0;
            for j = 1:1:config(1) % Looking at each bit in a dataset
                if(Exemplar(in,j))
                    Exemplar(in,j) = 0;
                else
                    Exemplar(in,j) = 1;
                end
                ApplyInput(in);
                if(y(NoOfLayer,1) > (TrueOutput(in,1) - Thresh) ...
                && y(NoOfLayer,1) < (TrueOutput(in,1) + Thresh) ...
                && y(NoOfLayer,2) > (TrueOutput(in,2) - Thresh) ...
                && y(NoOfLayer,2) < (TrueOutput(in,2) + Thresh))
                        count = count + 1;
                end
                if(Exemplar(in,j)) % Switching back
                    Exemplar(in,j) = 0;
                else
                    Exemplar(in,j) = 1;
                end
            end
            disp(['Training Set ' num2str(in) ': ' num2str(count) '/'
num2str(TotalCases) ' recognitions (' num2str(count / TotalCases *
100) '%)']);
        end
        t=toc;
        disp(['Total time elapsed : ' num2str(t) ' secs']);
    case 2
        tic
        disp('With 2 false bit per character:');
        TotalCases = config(1)*(config(1)-1);
        for in = 1:1:NoOfExemplar % Looking at each dataset
            count = 0;
            for j = 1:1:config(1) % Looking at each bit in a dataset
                for k = 1:1:config(1)
                    if j==k
                        continue;
                    end
                    if(Exemplar(in,j)) % Switching back
                        Exemplar(in,j) = 0;
                    else
                        Exemplar(in,j) = 1;
                    end
                    if(Exemplar(in,k))
```

```
                        Exemplar(in,k) = 0;
                    else
                        Exemplar(in,k) = 1;
                    end
                    ApplyInput(in);
                    if(y(NoOfLayer,1) > (TrueOutput(in,1) - Thresh)
...
                    && y(NoOfLayer,1) < (TrueOutput(in,1) + Thresh)
...
                    && y(NoOfLayer,2) > (TrueOutput(in,2) - Thresh)
...
                    && y(NoOfLayer,2) < (TrueOutput(in,2) + Thresh))
                            count = count + 1;
                    end
                    if(Exemplar(in,j))
                        Exemplar(in,j) = 0;
                    else
                        Exemplar(in,j) = 1;
                    end
                    if(Exemplar(in,k))
                        Exemplar(in,k) = 0;
                    else
                        Exemplar(in,k) = 1;
                    end
                end
            end
            disp(['Training Set ' num2str(in) ': ' num2str(count) '/'
num2str(TotalCases) ' recognitions (' num2str(count / TotalCases *
100) '%)']);
        end
        t=toc;
        disp(['Total time elapsed : ' num2str(t) ' secs']);
    case 3
        tic
        disp('With 3 false bit per character:');
        TotalCases = config(1) * (config(1)-1) * (config(1)-2);
        for in = 1:1:NoOfExemplar % Looking at each dataset
            count = 0;
            for j = 1:1:config(1) % Looking at each bit in a dataset
                for k = 1:1:config(1)
                    if j==k
                        continue;
                    end
                    for l = 1:1:config(1)
                        if k==l || j==l
                            continue;
                        end
                        if(Exemplar(in,j))
                            Exemplar(in,j) = 0;
                        else
                            Exemplar(in,j) = 1;
                        end
                        if(Exemplar(in,k))
                            Exemplar(in,k) = 0;
                        else
                            Exemplar(in,k) = 1;
                        end
                        if(Exemplar(in,l))
```

```
                            Exemplar(in,l) = 0;
                        else
                            Exemplar(in,l) = 1;
                        end
                        ApplyInput(in);
                        if(y(NoOfLayer,1) > (TrueOutput(in,1) -
Thresh) ...
                        && y(NoOfLayer,1) < (TrueOutput(in,1) +
Thresh) ...
                        && y(NoOfLayer,2) > (TrueOutput(in,2) -
Thresh) ...
                        && y(NoOfLayer,2) < (TrueOutput(in,2) +
Thresh))
                                count = count + 1;
                        end
                        if(Exemplar(in,j)) % Switching back
                            Exemplar(in,j) = 0;
                        else
                            Exemplar(in,j) = 1;
                        end
                        if(Exemplar(in,k))
                            Exemplar(in,k) = 0;
                        else
                            Exemplar(in,k) = 1;
                        end
                        if(Exemplar(in,l))
                            Exemplar(in,l) = 0;
                        else
                            Exemplar(in,l) = 1;
                        end
                    end
                end
            end
            disp(['Training Set ' num2str(in) ': ' num2str(count) '/'
num2str(TotalCases) ' recognitions (' num2str(count / TotalCases *
100) '%)']);
        end
        t=toc;
        disp(['Total time elapsed : ' num2str(t) ' secs']);
end
end
```

k. Testing.m

```
function Testing(sym)
%% Recognition with different input without noise
global NoOfLayer config y Thresh;

tic;
switch sym % Choose the input
    case 'A'
        in = 1;
    case 'C'
        in = 2;
    case 'N'
        in = 3;
```

```
    case 'X'
        in = 4;
    case 'Y'
        in = 5;
    case 'Z'
        in = 6;
    otherwise
        disp('Wrong selection...');
        in = 0;
end
if (in > 0)
    ApplyInput(in); % Apply input to the network with existing
weights
    disp(['Test Character: ' sym]);
    str = 'N/W Output: ';
    for j = 1:1:config(4)
        str = strcat(str, '[', num2str(y(NoOfLayer,j)), ']');
        if (y(NoOfLayer,j) > (1 - Thresh)) % Threshold output layer
            k = 1;
        elseif (y(NoOfLayer,j) < Thresh)
            k = 0;
        else
            k = 2;
        end
        match(j) = k;
    end
    disp(str);
    if match(1)==0 && match(2)==1 % Recognition of output
        found = 'A';
    elseif match(1)==1 && match(2)==0
        found = 'C';
    elseif match(1)==1 && match(2)==1
        found = 'N';
    else
        found = 'Other';
    end
    disp(['Recognized Character: ' found]);
end
t=toc;
disp(['Total testing time elapsed : ' num2str(t) ' secs']);
end
```