ECE 559 – Neural Networks

Character Recognition using Hopfield Network

Assignment – II

Arindam Bose
UIN: 665387232
10-1-2015

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 Hopfield Neural network (HNN) 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 HNN 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 Hopfield network 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 HNN 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 different bit noise.

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

3. *Description of Design of the HNN*

The goal of this case study is to recognize three characters 'A', 'C', 'N'. To do this, one-layer Hopfield network is created, it is trained with standard data sets (8×8); make the algorithm converge and it is tested the network with a set of test data other than trained set and also trained data having varying bit noise.

a. Network Structure: There are 64 inputs to the network. So 64 neurons are used to form the input layer as depicted in Fig. 1

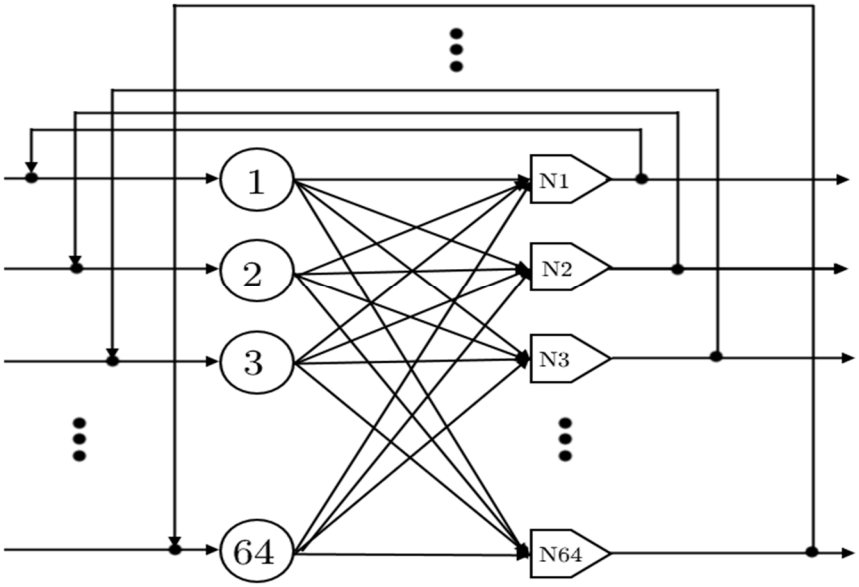

Fig. 1. Schematic design of the Hopfield neural network

b. Dataset Design: In this HNN 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 -1 (white pixels) or 1 (black pixels) as in Fig. 2.

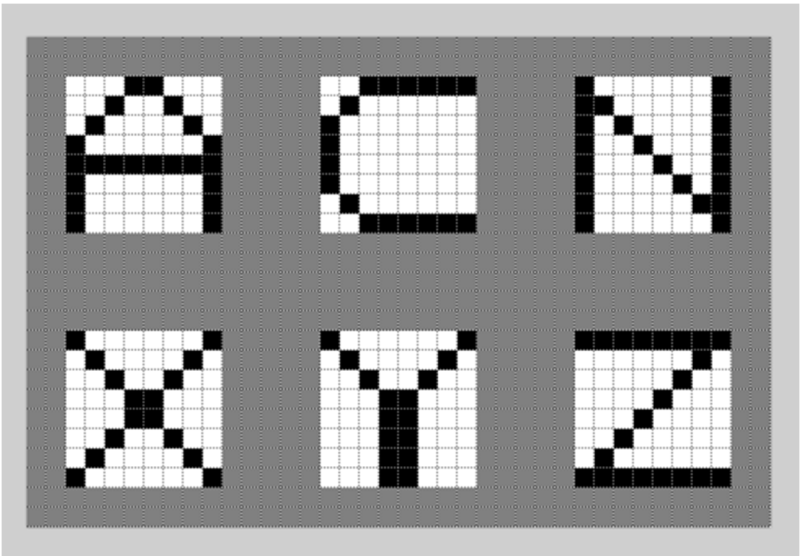

Fig. 2. Dataset: 'A', 'C', 'N', 'X', 'Y', 'Z'

c. Initial Weights:

1. Get all training data vectors $X_i, i = 1,2,3 \ldots L$.
2. Compute the weight matrix $W = \sum X_i X_i^T$ over L vectors.
3. Set $w_{ii} = 0, \forall i$, where w_{ii} is the i-th diagonal element of W.
4. Assign the j-th row vector of W to the j-th neuron as its initial weights.

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

```
Training set: 1: Pass
Training set: 2: Pass
Training set: 3: Pass
```

Training set 1, 2, 3 corresponds to letter 'A', 'C', 'N' respectively. The system successfully recognized all the trained exemplars.

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'.

```
Training set: 4: Untrained
Training set: 5: Untrained
Training set: 6: Untrained
```

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 1-40 bit noises:

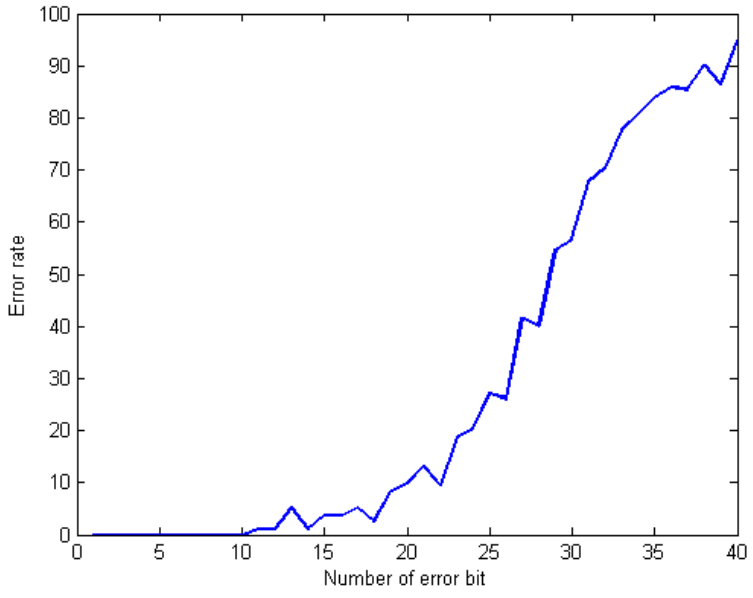

Fig. 3. Graph: Error vs Number of error-bit.

From the results it is evident that the network works fairly well even for 10-bit error. As the number of error bit increases from 10-bit, error rate also increases, but fairly slowly.

5. *Conclusions*

The HNN discussed here performed quiet 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 exemplar was 0.0032963 seconds which is very small. Testing time was also small, averaging 0.0069 seconds for each character. The Hopfield network is robust with high convergence rate and also it is seen that it has very high success rate even if in the case of large bit errors.

6. *References*

[1] Graupe D., "Principles of Artificial Neural Networks – 3ed," ch 7, pp. 123 - 146, 2013.

7. *Source Code (MATLAB R2013b)*

a. main.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECE 599 Neural Networs                                                      %
% Name: Arindam Bose                                                          %
% UIN: 665387232                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Implementation of Character Recognition system using Hopfield
Neural Network
%% Initialization
Initialization();

%% Main menu
first = true;
while first
    clc;
    disp('------------- Main Menu -------------');
    disp('1: Train network');
    disp('2: Test with trained exemplars');
    disp('3: Test with untrained exemplars');
    disp('4: Test with noise');
    disp('5: Generate graph for error rates');
    disp('0: Exit');
    ch = input('Your choice: ', 's');
    switch ch
        case '1'
            tic;
            TrainNetwork(); % Train network
            t = toc;
            disp(['Training complete in ' num2str(t) ' sec']);
        case '2'
```

```
            tic;
            TestWithExemplars('trained'); % Test with trained
exemplars A, C, N
            t = toc;
            disp(['Testing trained exemplar complete in ' num2str(t)
' sec']);
        case '3'
            tic;
            TestWithExemplars('untrained'); % Test with untrained
exemplars X, Y, Z
            t = toc;
            disp(['Testing untrained exemplar complete in '
num2str(t) ' sec']);
        case '4'
            n = input('Number of error bits ? ');
            RecognitionWithError(n); % Recognition with noise
        case '5'
            n = input('Number of final error bits ? ');
            GenerateGraph(n);
        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
clear all;
%% Global variabls
global HopfieldNetwork Exemplars Thresh Iteration
NumberPerTrainingSet;

NoOfNeuron = 64;
weightRange = [-5 5];
Thresh = 0;
Iteration = 20;
NumberPerTrainingSet = 64;

%% Hopfield network structure
HopfieldNetwork = [];
HopfieldNetwork.number = NoOfNeuron;
HopfieldNetwork.error = [];
HopfieldNetwork.Y = [];
HopfieldNetwork.neurons = [];
HopfieldNetwork.Z = [];
HopfieldNetwork.weights = [];

%% create a default layer
for i = 1: NoOfNeuron
    % create a default neuron
```

```
    offset = (weightRange(1) + weightRange(2))/2.0;
    range = abs(weightRange(2) - weightRange(1));
    weights = (rand(1, NoOfNeuron) - 0.5 )* range + offset;

    neuron.weights = weights;
    neuron.z = 0;
    neuron.y = 0;
    HopfieldNetwork.neurons = [HopfieldNetwork.neurons, neuron];
    HopfieldNetwork.weights = [HopfieldNetwork.weights; weights];
end

%% Exemplar definition
Exemplars = [];
Exemplars(1).input = [ -1, -1, -1,  1,  1, -1, -1, -1; ... % 'Letter
A'
                      -1, -1,  1, -1, -1,  1, -1, -1; ...
                      -1,  1, -1, -1, -1, -1,  1, -1; ...
                       1, -1, -1, -1, -1, -1, -1,  1; ...
                       1,  1,  1,  1,  1,  1,  1,  1; ...
                       1, -1, -1, -1, -1, -1, -1,  1; ...
                       1, -1, -1, -1, -1, -1, -1,  1; ...
                       1, -1, -1, -1, -1, -1, -1,  1];
Exemplars(1).name = 'A';
Exemplars(2).input = [ -1, -1,  1,  1,  1,  1,  1,  1;  ... %
'Letter C'
                      -1,  1, -1, -1, -1, -1, -1, -1; ...
                       1, -1, -1, -1, -1, -1, -1, -1; ...
                       1, -1, -1, -1, -1, -1, -1, -1; ...
                       1, -1, -1, -1, -1, -1, -1, -1; ...
                       1, -1, -1, -1, -1, -1, -1, -1; ...
                      -1,  1, -1, -1, -1, -1, -1, -1; ...
                      -1, -1,  1,  1,  1,  1,  1,  1];
Exemplars(2).name = 'C';
Exemplars(3).input = [  1, -1, -1, -1, -1, -1, -1, 1;  ... % 'Letter
N'
                        1,  1, -1, -1, -1, -1, -1, 1; ...
                        1, -1,  1, -1, -1, -1, -1, 1; ...
                        1, -1, -1,  1, -1, -1, -1, 1; ...
                        1, -1, -1, -1,  1, -1, -1, 1; ...
                        1, -1, -1, -1, -1,  1, -1, 1; ...
                        1, -1, -1, -1, -1, -1,  1, 1; ...
                        1, -1, -1, -1, -1, -1, -1, 1];
Exemplars(3).name = 'N';
Exemplars(4).input = [  1, -1, -1, -1, -1, -1, -1,  1;  ... %
'Letter X'
                      -1,  1, -1, -1, -1, -1,  1, -1; ...
                      -1, -1,  1, -1, -1,  1, -1, -1; ...
                      -1, -1, -1,  1,  1, -1, -1, -1; ...
                      -1, -1, -1,  1,  1, -1, -1, -1; ...
                      -1, -1,  1, -1, -1,  1, -1, -1; ...
                      -1,  1, -1, -1, -1, -1,  1, -1; ...
                       1, -1, -1, -1, -1, -1, -1,  1];
Exemplars(4).name = 'X';
Exemplars(5).input = [  1, -1, -1, -1, -1, -1, -1,  1;  ... %
'Letter Y'
                      -1,  1, -1, -1, -1, -1,  1, -1; ...
                      -1, -1,  1, -1, -1,  1, -1, -1; ...
                      -1, -1, -1,  1,  1, -1, -1, -1; ...
```

```
                        -1, -1, -1,  1,  1, -1, -1, -1; ...
                        -1, -1, -1,  1,  1, -1, -1, -1; ...
                        -1, -1, -1,  1,  1, -1, -1, -1; ...
                        -1, -1, -1,  1,  1, -1, -1, -1];
Exemplars(5).name = 'Y';
Exemplars(6).input = [  1,  1,  1,  1,  1,  1,  1,  1;  ... %
'Letter Z'
                        -1, -1, -1, -1, -1, -1,  1, -1; ...
                        -1, -1, -1, -1, -1,  1, -1, -1; ...
                        -1, -1, -1, -1,  1, -1, -1, -1; ...
                        -1, -1, -1,  1, -1, -1, -1, -1; ...
                        -1, -1,  1, -1, -1, -1, -1, -1; ...
                        -1,  1, -1, -1, -1, -1, -1, -1; ...
                         1,  1,  1,  1,  1,  1,  1,  1];
Exemplars(6).name = 'Z';
end
```

c. TrainNetwork.m

```
function TrainNetwork()
% Function for training network with training exemplars
global HopfieldNetwork Exemplars;

weights = zeros;
for i = 1 : 3 % Count training exemplars and applying weights
    mIn = Exemplars(i).input(:);
    weights = weights + mIn * mIn';
end
wSize = size(weights);
for i = 1: wSize(1) % store weights
    weights(i, i) = 0;
    HopfieldNetwork.neurons(i).weights = weights(i,:);
end
HopfieldNetwork.weights = weights;
end
```

d. TestWithExemplars.m

```
function TestWithExemplars(exemplarType)
% funtion for test with trained/untrained exemplar
global Exemplars;

str = [];
for i = 1 : 3 % Count exemplar
    switch exemplarType
        case 'trained'
            j = i;
        case 'untrained'
            j = i + 3;
    end
    Propagation(Exemplars(j).input);
    [output] = Classification();
    if strcmp(output.Name, Exemplars(j).name)
```

```
        astr = ['Training set: ' num2str(j), ': Pass'];
    else
        astr = ['Training set: ' num2str(j), ': Untrained'];
    end
    str = char(str, astr);
end
disp(str);
end
```

e. Propagation.m

```
function [nIter] = Propagation(inputData)
% Function for propagating the Hopfield
global HopfieldNetwork Thresh Iteration;

% get inputs
nnInputs = inputData(:)';
nIter = 0;
delta = 2* Thresh + 1;
while delta > Thresh
    nIter = nIter + 1;
    if nIter > Iteration
        break;
    end
    % interation here
    delta = 0;
    Y = [];
    Z = [];
    for ele = 1:HopfieldNetwork.number
        % retrieve one neuron
        neuron = HopfieldNetwork.neurons(ele);
        % get analog outputs
        z = neuron.weights * nnInputs';
        neuron.z = z;
        Z = [Z, z];
        % get output
        Th = 0;
        if z > Th
            y = 1;
        elseif z < Th
            y = -1;
        else
            y = z;
        end
        neuron.y = y;
        Y = [Y, neuron.y];
        % update the structure
        HopfieldNetwork.neurons(ele) = neuron;
        % get the error
        newError = (y - nnInputs(ele)) * (y - nnInputs(ele));
        delta = delta + newError;
    end
    HopfieldNetwork.Y = Y;
    HopfieldNetwork.Z = Z;
    HopfieldNetwork.error = delta;
    % feedback
```

```
    nnInputs = Y;
end
return;
```

f. Classification.m

```
function [output] = Classification()
% Function for Hopfield classification
global HopfieldNetwork Exemplars;

delta = [];
Y = HopfieldNetwork.Y';
for i = 1 : 3 % Count training exemplars
    aSet = Exemplars(i).input(:);
    diff = abs(aSet - Y);
    diff = diff.^2;
    newError = sum(diff);
    delta = [delta, newError];
end
[eMin, eInd] = min(delta);
output.Name = Exemplars(eInd).name;
output.Vector = Exemplars(eInd).input;
output.Error = eMin;
end
```

g. RecognitionWithError.m

```
function RecognitionWithError(nBitError)
% Function for recognition with error

testData = GenerateTestData(nBitError);
sizeT = size(testData);
str = [];
success = 0;
for i = 1: sizeT(2) % count number of testdata
    [nIter] = Propagation(testData(i).input);
    [output] = Classification();
    strFormat = ' ';
    vstr = char(strFormat,num2str(i),num2str(nIter));
    if strcmp(output.Name, testData(i).name)
        success = success + 1;
        astr = [vstr(2,:), ': Success -> Iternation: ', vstr(3,:),'
Error: ', num2str(output.Error)];
    else
        astr = [vstr(2,:), ': Fail -> Iternation: ', vstr(3,:),];
    end
    str = char(str, astr);
end
astr = ['Success rate: ', num2str(success * 100/ sizeT(2)),'%'];
str = char(str, astr);
disp(str);
end
```

h. GenerateTestData.m

```
function testData = GenerateTestData(nBitError)
% Function for generating test inputs with error bit
global Exemplars NumberPerTrainingSet;

testData = [];
id = 1;
for i = 1 : 3 % Count training exemplars
    input = Exemplars(i).input;
    name = Exemplars(i).name;
    sizeI = size(input);
    for j = 1: NumberPerTrainingSet
        row = [];
        col = [];
        flag = ones(size(input));
        bitErrorNum = 0;
        while bitErrorNum < nBitError
            x = ceil(rand(1) * sizeI(1));
            y = ceil(rand(1) * sizeI(2));
            if x <= 0
                x = 1;
            end
            if y <= 0
                y = 1;
            end
            if flag(x, y) ~= -1
                bitErrorNum = bitErrorNum + 1;
                flag(x, y) = -1;
                row = [row, x];
                col = [col, y];
            end
        end
        newInput = input;
        for p = 1:nBitError
            newInput(row(p), col(p)) = newInput(row(p), col(p)) * (-
1);
        end
        testData(id).input = newInput;
        testData(id).name = name;
        id = id + 1;
    end
end
```

i. GenerateGraph.m

```
function GenerateGraph(nBitError)
for i = 1 : nBitError
    errorate(i) = RecognitionWithError(i);
end
figure;
plot(errorate, 'LineWidth', 2);
xlabel('Number of error bit');
ylabel('Error rate');
end
```