ECE 559 – Neural Networks

Character Recognition using LAMSTAR1 and LAMSTAR2 Neural Networks

Assignment – V

Arindam Bose
UIN: 665387232
10-29-2015

1. *Problem Statement*

Use Large Scale Memory Storage and Retrival Neural network (LAMSTAR) for character recognition. 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 LAMSTAR should be able to classify any 8x8 characters.

2. *Introduction*

In this project we recognize the characters '0', '1' and '3', to demonstrate the use of a LAMSTAR network. The LAMSTAR neural network is specifically designed for application to retrieval, diagnosis, classification, prediction and decision problems which involve a very large number of categories. The resulting LAMSTAR neural network [Graupe, 1997, Graupe and Kordylewski, 1998] is designed to store and retrieve patterns in a computationally efficient manner, using tools of neural networks, especially Kohonen's SOM (Self Organizing Map)-based network modules [Kohonen, 1988], combined with statistical decision tools. The basic storage modules of the LAMSTAR network are modified Kohonen networks. SOM modules are Associate-Memory-based WTA. The information is stored and processed via correlation links between individual neurons in separate SOM modules in the LAMSTAR network. Its ability to deal with a large number of categories is partly due to its use of simple calculation of link weights. The link weights are the main engine of the network, connecting many layers of SOM modules such that the emphasis is on correlation of link weights between atoms of memory, not on the memory atoms themselves. Like this, the design becomes closer to knowledge processing in the biological CNS. The forgetting feature is a basic feature of biological networks whose efficiency depends on it, as is the ability to deal with incomplete data sets. Its elementary neural unit or cell (neuron) is the one employed in all neural networks. Accordingly, if the p inputs into a given neuron at the jth SOM layer are denoted as $x_{ij};\ i = 1,2,\dots,p$, and if the (single) output of that neuron is denoted as y, then the neuron's output y satisfies:

$$y = f\left[\sum_{i=1}^{p} w_{ij}x_{ij}\right]$$

where $f[\cdot]$ is a nonlinear function denoted as Activation function.

The Winner-Take-All (WTA) principle, is employed such that an output is produced only at the winning neuron, namely, at the output of the neuron whose storage weights w_{ij} are closest to vector x_j when a best-matching memory is sought at the jth SOM module. By using a link weights structure for its decision and

browsing, the LAMSTAR network utilizes not just the stored memory values w_{ij} as in other neural networks, but also the interrelations these memories to the decision module and between the memories themselves.

These relations are fundamental to its operation. By Hebb's Law link weights adjust and serve to establish flow of neuronal signal traffic between groups of neurons, such that when a certain neuron fires very often in close time, then the interconnecting link-weights relating to that traffic, increase as compared to other interconnections. Link weights serve as Hebbian intersynaptic weights and adjust accordingly.

Also I tried to improve the results using a normalized LAMSTAR network. It improves the recognition rate but it's a bit slower than LAMSTAR 1.

3. Design of the network

The LAMSTAR network has the following components:

a. Input word and its subwords:
The input word (in this case, the character) is divided into a number of subwords. Each subword represents an attribute of the input word. The subword division in the character recognition problem was done by considering every row and every column as a subword hence resulting in a total of 16 subwords for a given character.

b. SOM modules for storing input subwords:
For every subword there is an associated Self Organizing Map (SOM) module with neurons that are designed to function as Kohonen 'Winner Take All' neurons where the winning neuron has an output of 1 while all other neurons in that SOM module have a zero output.
In this project, the SOM modules are built dynamically in the sense that instead of setting the number of neurons at some fixed value arbitrarily, the network was built to have neurons depending on the class to which a given input to a particular subword might belong. For example if there are two subwords that have all their pixels as '1's, then these would fire the same neuron in their SOM layer and hence all they need is 1 neuron in the place of 2 neurons. This way the network is designed with lesser number of neurons and the time taken to fire a particular neuron at the classification stage is reduced considerably. A generalized block diagram of LAMSTAR can be found at Fig.1.

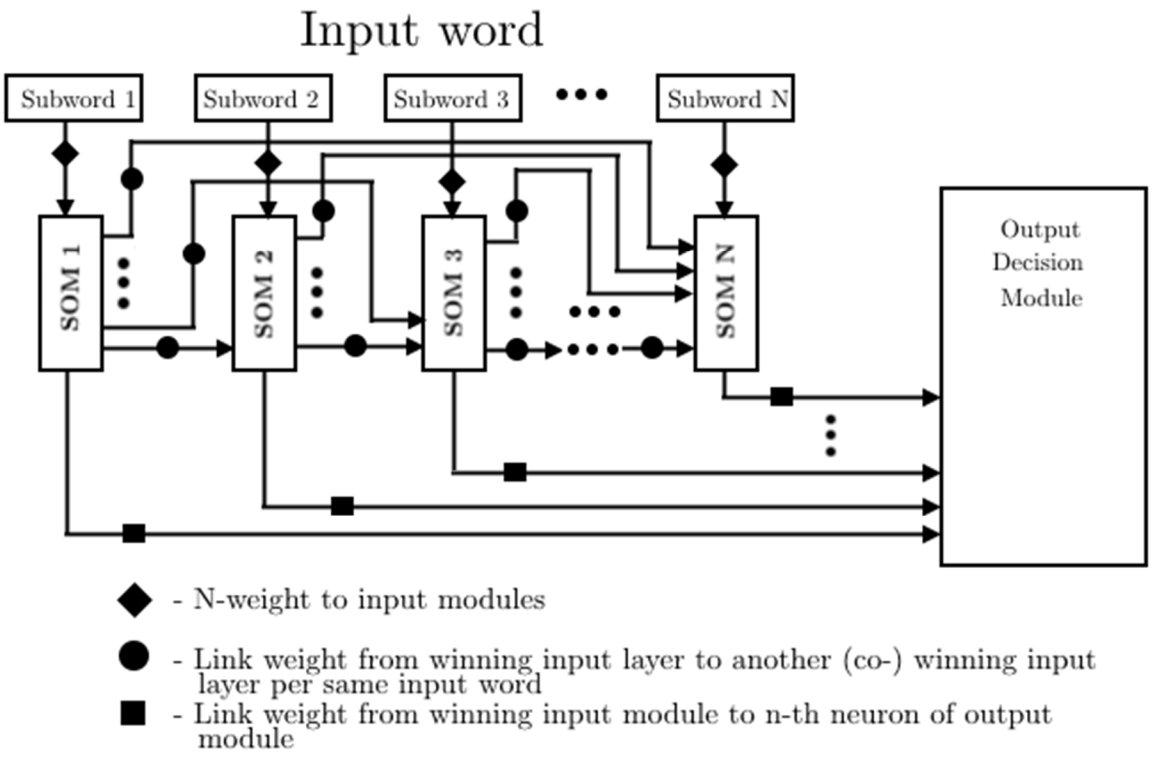

Fig. 1: A generalized LAMSTAR block-diagram

c. Output layer:

The present output layer is designed to have two layers, which have the following neuron firing patterns:

Pattern	Output Neuron 1	Output Neuron 2
0	Not fired	Not fired
1	Not fired	Fired
3	Fired	Not fired
Rest Of The World	Fired	Fired

Table 1: Firing order of the output neurons.

The link-weights from the input SOM modules to the output decision layer are adjusted during training on a reward/punishment principle. Furthermore, they continue being trained during normal operational runs.

4. Fundamental Principles

a. Dynamic SOM layer design:

The number of neurons in every SOM module is not fixed. The network is designed to grow dynamically. At the beginning there are no neurons in any of the modules. So when the training character is sent to the network, the first neuron in every subword is built. Its output is made 1 by adjusting the weights based on the 'Winner Take All' principle.

b. Winner Take All principle:

The SOM modules are designed to be Kohonen layer neurons, which act in accordance to the 'Winner Take All' Principle. This layer is a competitive layer wherein the Eucledian distance between the weights at every Kohonen layer and the input pattern is measured and the neuron that has the least distance if declared to be the winner.

5. Training Process

The training of the LAMSTAR network if performed as follows:

a. Subword Formation:

The input patterns are to be divided into subwords before training/testing the LAMSTAR network. In order to perform this, the every row of the input 8x8 character is read to make 8 subwords followed by every column to make another 8 subwords resulting in a total of 16 subwords.

b. Input Normalization:

Each subwords of every input pattern is normalized as follows:

$$x_i' = \frac{x_i}{\sqrt{\sum x_j^2}}$$

where, x — subword of an input pattern. During the process, those subwords, which are all zeros, are identified and their normalized values are manually set to zero.

c. Other character Patterns:

The network is also trained with the other character patterns. This is done by taking the average of these patterns and including the average as one of the training patterns.

d. Dynamic Neuron formation in the SOM modules:

The first neuron in all the SOM modules are constructed as Kohonen neurons as follows:

• As the first pattern is input to the system, one neuron is built with 6 inputs and random weights to start with initially and they are also normalized just like the input subwords. Then the weights are adjusted such that the output of

this neuron is made equal to 1 (with a tolerance of 10^{-5} according to the formula:

$$w(n+1) = w(n) + \alpha * (x - w(n))$$

where,

α — learning constant = 0.8

w — weight at the input of the neuron

x — subword

$$z = w * x$$

where, z — output of the neuron (in the case of the first neuron it is made equal to 1).

• When the subwords of the subsequent patterns is input to the respective modules, the output at any of the previously built neuron is checked to see if it is close to 1 (with a tolerance of 0.05). If one of the neurons satisfies the condition, then this is declared as the winning neuron, i.e., a neuron whose weights closely resemble the input pattern. Else another neuron is built with new sets of weights that are normalized and adjusted as above to resemble the input subword.

• During this process, if there is a subword with all zeros then this will not contribute to a change in the output and hence the output is made to zero and the process of finding a winning neuron is bypassed for such a case.

e. Desired neuron firing pattern:
The output neuron firing pattern for each character in the training set has been established as given in Table 1.

f. Link weights:
Link weightsare defined as the weights that come from the winning neuron at every module to the 2 output neurons. If in the desired firing, a neuron is to be fired, then its corresponding link weights are rewarded by adding a small positive value of 0.05 every iteration for 20 iterations. On the other hand, if a neuron should not be fired then its link weights are reduced 20 times by 0.05. This will result in the summed link weights at the output layer being a positive value indicating a fired neuron if the neuron has to be fired for the pattern and high negative value if it should not be fired.

g. The weights at the SOM neuron modules and the link weights are stored.

5.1. Training Dataset

a. Training Dataset:
The LAMSTAR network is trained to detect the characters '0', '1', '3' and 'other' characters. The training set consists of 16 training patterns 5 each for '0', '1' and '3' and one average of the 'rest of the world' characters.

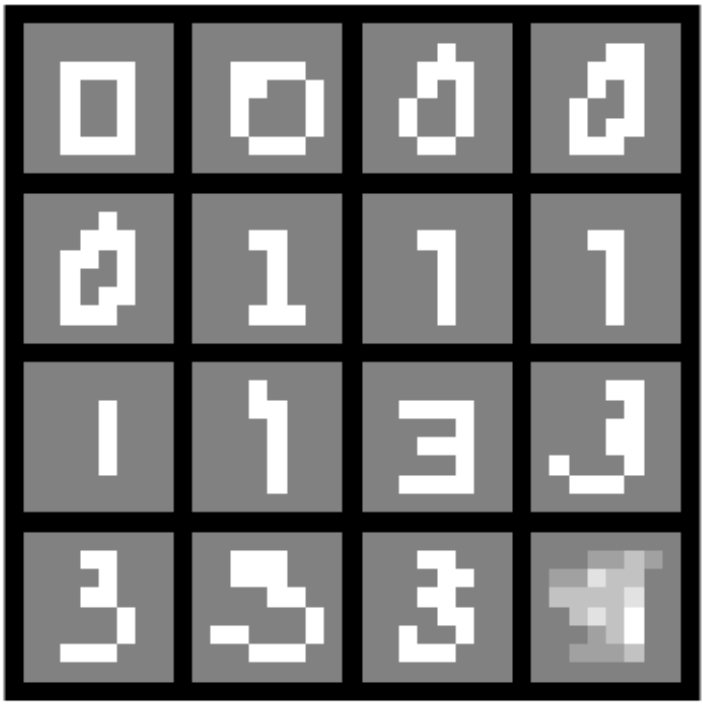

Fig. 2: Training Pattern Set for recognizing characters '0', '1', '3' and 'mean of rest of world' patterns '9', '4', '6', '5'.

b. 'Rest of the world' patterns:
The rest of the world patterns used to train the network are as follows:

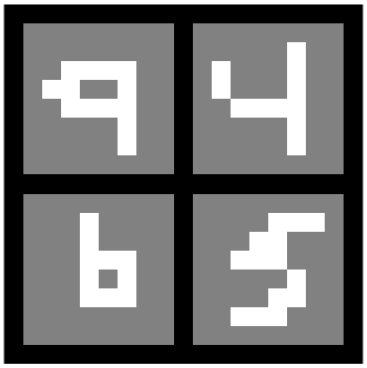

Fig. 3: 'Rest of the world patterns '9', '4', '6', '5'.

6. Testing Process

The LAMSTAR network was tested with 30 patterns as follows:

• The patterns are processed to get 16 subwords as before. Normalization is done for the subwords as explained in the training.

• The stored weights are loaded

• The subwords are propagated through the network and the neuron with the maximum output at the Kohonen layer is found and their link weights are sent to the output neurons.

• The output is a sum of all the link weights.

• All the patterns were successfully classified. There were subwords that were completely zero so that the pattern would be partially incorrect. Even these were correctly classified.

6.1. Testing Dataset

The network was tested with 30 characters consisting of 10 pattern each of '0', '1' and '3'. All the patterns are noisy, either distorted or a whole row/column removed to test the efficiency of the training. The following is the test pattern set.

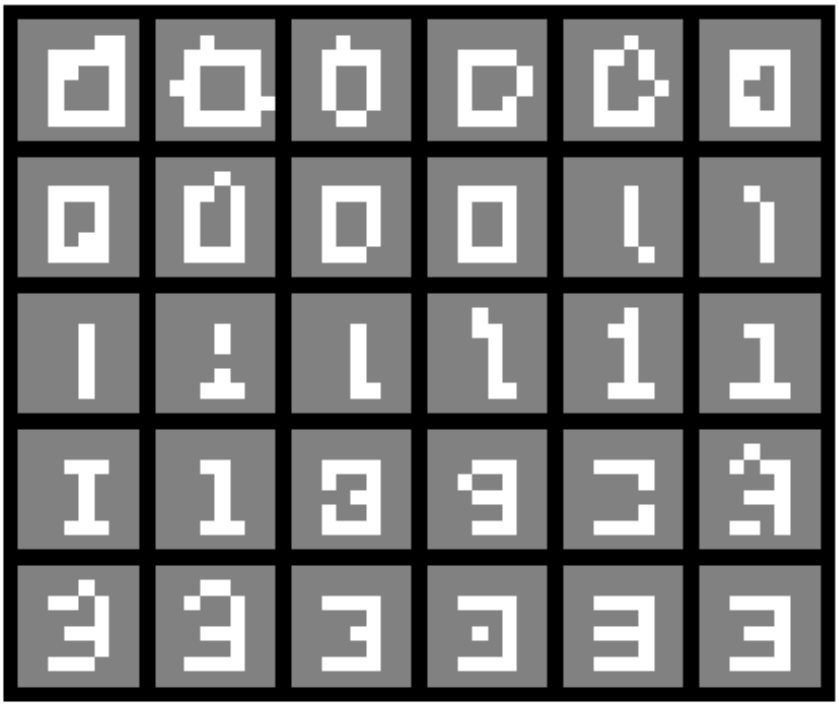

Fig. 4: Test pattern set consisting of 10 patterns each for '0', '1' and '3'.

7. *Results*

a. Network Training: The results obtained after training the network are presented in Table 2:

- Number of training patterns = 16
- Training efficiency = 100%
- Number of SOM modules = 16
- The number of neurons in the 16 SOM modules after dynamic neuron formation in are:

SOM Module Number	**Number Of Neurons**
1	1
2	5
3	6
4	9
5	6
6	5
7	4
8	1
9	1
10	2
11	10
12	7
13	9
14	6
15	3
16	1

Table 2: Number of neurons in the SOM modules.

b. Test results:

The result of testing the network are as in Table 3:

- Number of testing patterns = 30
- Neurons fired at the modules for the 16 test patterns
- Efficiency: For LAMSTAR 1: 28/30 correct: 93.33%

For normalized LAMSTAR 1: 30/30 correct: 100%

Pattern No	Test Pattern	Neuron 1	Neuron 2	Result
1	0	0 (Not Fired)	0 (Not Fired)	Correct
2	0	0 (Not Fired)	0 (Not Fired)	Correct
3	0	0 (Not Fired)	0 (Not Fired)	Correct
4	0	0 (Not Fired)	0 (Not Fired)	Correct
5	0	0 (Not Fired)	0 (Not Fired)	Correct
6	0	0 (Not Fired)	0 (Not Fired)	Correct
7	0	0 (Not Fired)	0 (Not Fired)	Correct
8	0	0 (Not Fired)	0 (Not Fired)	Correct
9	0	0 (Not Fired)	0 (Not Fired)	Correct
10	0	0 (Not Fired)	0 (Not Fired)	Correct
11	1	0 (Not Fired)	1 (Fired)	Correct
12	1	0 (Not Fired)	1 (Fired)	Correct
13	1	0 (Not Fired)	1 (Fired)	Correct
14	1	0 (Not Fired)	1 (Fired)	Correct
15	1	0 (Not Fired)	1 (Fired)	Correct
16	1	0 (Not Fired)	1 (Fired)	Correct
17	1	0 (Not Fired)	1 (Fired)	Correct
18	1	0 (Not Fired)	1 (Fired)	Correct
19	1	0 (Not Fired)	1 (Fired)	Correct
20	1	0 (Not Fired)	1 (Fired)	Correct
21	3	0 (Not Fired)	0 (Not Fired)	**Wrong**
22	3	1 (Fired)	0 (Not Fired)	Correct
23	3	0 (Not Fired)	0 (Not Fired)	**Wrong**
24	3	1 (Fired)	0 (Not Fired)	Correct
25	3	1 (Fired)	0 (Not Fired)	Correct
26	3	1 (Fired)	0 (Not Fired)	Correct
27	3	1 (Fired)	0 (Not Fired)	Correct
28	3	1 (Fired)	0 (Not Fired)	Correct
29	3	1 (Fired)	0 (Not Fired)	Correct
30	3	1 (Fired)	0 (Not Fired)	Correct

Table 3: Firing pattern for the test characters for LAMSTAR 1.

Pattern No	Test Pattern	Neuron 1	Neuron 2	Result
1	0	0 (Not Fired)	0 (Not Fired)	Correct
2	0	0 (Not Fired)	0 (Not Fired)	Correct
3	0	0 (Not Fired)	0 (Not Fired)	Correct
4	0	0 (Not Fired)	0 (Not Fired)	Correct
5	0	0 (Not Fired)	0 (Not Fired)	Correct
6	0	0 (Not Fired)	0 (Not Fired)	Correct
7	0	0 (Not Fired)	0 (Not Fired)	Correct
8	0	0 (Not Fired)	0 (Not Fired)	Correct
9	0	0 (Not Fired)	0 (Not Fired)	Correct
10	0	0 (Not Fired)	0 (Not Fired)	Correct
11	1	0 (Not Fired)	1 (Fired)	Correct
12	1	0 (Not Fired)	1 (Fired)	Correct
13	1	0 (Not Fired)	1 (Fired)	Correct
14	1	0 (Not Fired)	1 (Fired)	Correct
15	1	0 (Not Fired)	1 (Fired)	Correct
16	1	0 (Not Fired)	1 (Fired)	Correct
17	1	0 (Not Fired)	1 (Fired)	Correct
18	1	0 (Not Fired)	1 (Fired)	Correct
19	1	0 (Not Fired)	1 (Fired)	Correct
20	1	0 (Not Fired)	1 (Fired)	Correct
21	3	1 (Fired)	0 (Not Fired)	Correct
22	3	1 (Fired)	0 (Not Fired)	Correct
23	3	1 (Fired)	0 (Not Fired)	Correct
24	3	1 (Fired)	0 (Not Fired)	Correct
25	3	1 (Fired)	0 (Not Fired)	Correct
26	3	1 (Fired)	0 (Not Fired)	Correct
27	3	1 (Fired)	0 (Not Fired)	Correct
28	3	1 (Fired)	0 (Not Fired)	Correct
29	3	1 (Fired)	0 (Not Fired)	Correct
30	3	1 (Fired)	0 (Not Fired)	Correct

Table 4: Firing pattern for the test characters for LAMSTAR 2.

8. *Conclusions*

The network is much faster than Back Propagation network for the same character recognition problem. The neurons are built dynamically in the SOM modules by which the computation is largely reduced as the search time is reduced to a minimum number of neurons. The training procedure of 16 samples takes only **0.00396 sec** for LAMSTAR1 and **0.007896 sec** for normalzed LAMSTAR and testing of 30 samples takes only **0.006764 sec** and **0.005596 sec** respectively. Also, in cases where the inputs are all 0's the recognition efficiency is 100% due to the link weights. The NN learns as it goes even if untrained. The output for the test pattern as is 93.33% efficient. Thus, the LAMSTAR network is trained for three characters '0', '1' and '3'. Also the number of neurons that function were reduced to a minimum quantity which also reduced the search time.

9. *References*

[1] Graupe D., "Principles of Artificial Neural Networks – 3ed," ch 9, pp. 203 - 235, 2013

10. *Appendix: Source Code (MATLAB R2015b)*

a. main.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ECE 599 Neural Networs                                                  %
% Name: Arindam Bose                                                      %
% UIN: 665387232                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Implementation of Character Recognition system using LAMSTAR
Neural Network

% File for setting up the LAMSTAR Network

clear all;close all; clc
load data.mat

X_train = [X(1:5,:)' X(51:55,:)'  X(101:105,:)'
(sum(X_negative)./4)'];
[row, col] = size(X_train);
numSubWords = sqrt(row)*2;

flag = zeros(1,numSubWords);
% Forming Sub Words
for i = 1:size(X_train,2)
    tempX = reshape(X_train(:,i),8,8);
    for j = 1:numSubWords
       if j<=8
           X_in{i}(j,:) = tempX(j,:);
```

```
        else
            X_in{i}(j,:) = tempX(:,j-8)';
        end
    end
end

% Normalizing the inputs
check(1,:) = zeros(1,8);
for i = 1:size(X_train,2)
    for j = 1:numSubWords
        if (sum(X_in{i}(j,:)) ~= sum(check(1,:)))
            X_norm{i}(j,:) = X_in{i}(j,:) ./
sqrt(sum(X_in{i}(j,:).^2));
        else
            X_norm{i}(j,:) = zeros(1,8);
        end
    end
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dynamic Building of neurons
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Building of the first neuron is done as Kohonen Layer neuron
%(this is for all the subwords in the first input pattern for all
SOM modules
i = 1;
ct = 1;
while (i<=numSubWords),
    cl = 0;
    for t = 1:8,
        if (X_norm{ct}(i,t)==0),
            cl = cl+1;
        end
    end
    if (cl == 8),
        Z{ct}(i) = 0;
    elseif (flag(i) == 0),
        W{i}(:,ct) = rand(8,1);
        flag(i) = ct;
        W_norm{i}(:,ct) = W{i}(:,ct)/sqrt(sum(W{i}(:,ct).^2));
        Z{ct}(i)= X_norm{ct}(i,:)*W_norm{i};
        alpha =0.7;
        tol = 1e-5;
        while(Z{ct}(i) <= (1-tol)),
            W_norm{i}(:,ct) = W_norm{i}(:,ct) +
alpha*(X_norm{ct}(i,:)' -...
            W_norm{i}(:,ct));
            Z{ct}(i) = X_norm{ct}(i,:)*W_norm{i}(:,ct);
        end
    end
    r(ct,i) = 1;
    i = i+1;
end

r(ct,:) = 1;
ct = ct+1;
```

```
while (ct <= size(X_train,2)),
    for i = 1:numSubWords,
        cl = 0;
        for t = 1:8,
            if (X_norm{ct}(i,t)==0),
                cl = cl+1;
            end
        end
        if (cl == 8),
            Z{ct}(i) = 0;
        else
            r(ct,i) = flag(i);
            r_new=0;
            for k = 1:max(r(ct,i)),
                Z{ct}(i) = X_norm{ct}(i,:)*W_norm{i}(:,k);
                if Z{ct}(i)>=0.95,
                    r_new = k;
                    flag(i) = r_new;
                    r(ct,i) = flag(i);
                    break;
                end
            end
            if (r_new==0),
                flag(i) = flag(i)+1;
                r(ct,i) = flag(i);
                W{i}(:,r(ct,i)) = rand(8,1);
                %flag(i) = r
                W_norm{i}(:,r(ct,i)) =
W{i}(:,r(ct,i))/sqrt(sum(W{i}(:,r(ct,i)).^2));
                Z{ct}(i) = X_norm{ct}(i,:)*W_norm{i}(:,r(ct,i));
                alpha =0.8;
                tol = 1e-5;
                while(Z{ct}(i) <= (1-tol)),
                    W_norm{i}(:,r(ct,i)) = W_norm{i}(:,r(ct,i)) +
...
                    alpha*(X_norm{ct}(i,:)' - ...
                    W_norm{i}(:,r(ct,i)));
                    Z{ct}(i) = X_norm{ct}(i,:)*W_norm{i}(:,r(ct,i));
                end
            end
        end
    end
    ct = ct+1;
end
save W_norm W_norm

for i = 1:5,
    d(i,:) = [0 0];
    d(i+5,:) = [0 1];
    d(i+10,:) = [1 0];
end
d(16,:) = [1 1];

%%%%%%%%%%%%%%%
% Link Weights
%%%%%%%%%%%%%%%
ct = 1;
m_r = max(r);
```

```
for i = 1:numSubWords,
    L_w{i} = zeros(m_r(i),2);
end

ct = 1;
%%% Link weights and output calculations
Z_out = zeros(16,2);
while (ct <= 16)
    %for mn = 1:2
    L = zeros(16,2);
    % for count = 1:20,
    % count = size(find(r(:,i) == r(ct,i)) , 1); % For Norm LAMSTAR
    for i = 1:numSubWords,
        if (r(ct,i)~=0),
            for j = 1:2,
                if (d(ct,j)==0),
                    L_w{i}(r(ct,i),j) = L_w{i}(r(ct,i),j)-0.05*20;
%                   L_w{i}(r(ct,i),j) = L_w{i}(r(ct,i),j)/count -
0.05*20; % For Norm LAMSTAR
                else
                    L_w{i}(r(ct,i),j) = L_w{i}(r(ct,i),j)+0.05*20;
%                   L_w{i}(r(ct,i),j) = L_w{i}(r(ct,i),j)/count +
0.05*20; % For Norm LAMSTAR
                end
            end
            L(i,:) = L_w{i}(r(ct,i),:);
%           L(i,:) = L_w{i}(r(ct,i),:)/count; % For Norm LAMSTAR
        end
    end
    Z_out(ct,:) = sum(L);
    ct = ct+1;
end
save L_w L_w
```

b. <u>test.m</u>

```
clear all; clc;
% X = test_pattern;
load W_norm
load L_w
load data.mat

numSubWords = 16;
X_test = [X(31:40,:)' X(81:90,:)' X(131:140,:)'];

% To make 12 subwords
for i = 1:size(X_test,2),
    tempX = reshape(X_test(:,i),8,8);
    for j = 1:numSubWords
        if j<=8
            X_in{i}(j,:) = tempX(j,:);
        else
            X_in{i}(j,:) = tempX(:,j-8)';
```

```
        end
    end

    check(1,:) = zeros(1,8);
    for j = 1:numSubWords
        if (sum(X_in{i}(j,:)) ~= sum(check(1,:)))
            X_norm{i}(j,:) = X_in{i}(j,:) ./
sqrt(sum(X_in{i}(j,:).^2));
        else
            X_norm{i}(j,:) = zeros(1,8);
        end
    end

    for k = 1:15,
        if isempty(W_norm{k}),
            Z_out(k,:) = [0 0];
        else
            Z = X_norm{i}(k,:)*W_norm{k};
            index(k) = find((Z == max(Z)),1);
            L(k,:) = L_w{k}(index(k),:);
            Z_out(k,:) = L(k,:)*Z(index(k));
        end
    end
    final_Z = sum(Z_out);
    disp(['Test Pattern: ' num2str(i) ' |output: '
num2str(sigmoid(final_Z)>0.5)]);
end
```

c. sigmoid.m

```
function val = sigmoid(data)
    val = 1 ./ (1 + exp(-data));
end
```

d. displayData.m

```
function [h, display_array] = displayData(X, example_width)
%DISPLAYDATA Display 2D data in a nice grid
%   [h, display_array] = DISPLAYDATA(X, example_width) displays 2D
data
%   stored in X in a nice grid. It returns the figure handle h and
the
%   displayed array if requested.

% Set example_width automatically if not passed in
if ~exist('example_width', 'var') || isempty(example_width)
   example_width = round(sqrt(size(X, 2)));
end

% Gray Image
colormap(gray);

% Compute rows, cols
[m, n] = size(X);
example_height = (n / example_width);
```

```
% Compute number of items to display
display_rows = floor(sqrt(m));
display_cols = ceil(m / display_rows);

% Between images padding
pad = 1;

% Setup blank display
display_array = - ones(pad + display_rows * (example_height + pad),
...
                       pad + display_cols * (example_width + pad));

% Copy each example into a patch on the display array
curr_ex = 1;
for j = 1:display_rows
   for i = 1:display_cols
         if curr_ex > m,
               break;
         end
         % Copy the patch

         % Get the max value of the patch
         max_val = max(abs(X(curr_ex, :)));
         display_array(pad + (j - 1) * (example_height + pad) +
(1:example_height), ...
                       pad + (i - 1) * (example_width + pad) +
(1:example_width)) = ...
                                  reshape(X(curr_ex, :),
example_height, example_width) / max_val;
         curr_ex = curr_ex + 1;
   end
   if curr_ex > m,
         break;
   end
end

% Display Image
h = imagesc(display_array, [-1 1]);

% Do not show axis
axis image off
drawnow;
end
```