COEN 6331 Problem Set 3

Yuelin Yao 40194926

March 10, 2022

1 Design of Hopfield Network

The hopfield network can reconstruct a model of the complete data to determine the type of data that is missing. Each node of this network is in a state of -1 or 1 at any point in time. Using -1 and 1 to denote the set of 8 patterns in the problem set, the results are plotted as shown in Figure 1.

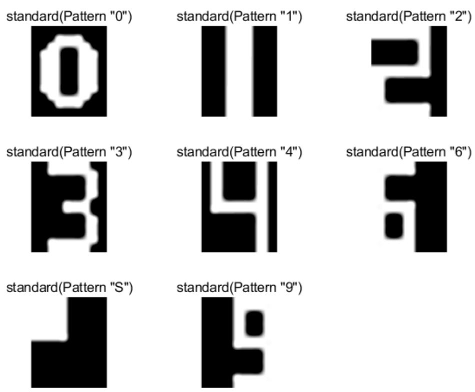

Figure 1: Standard Pattern

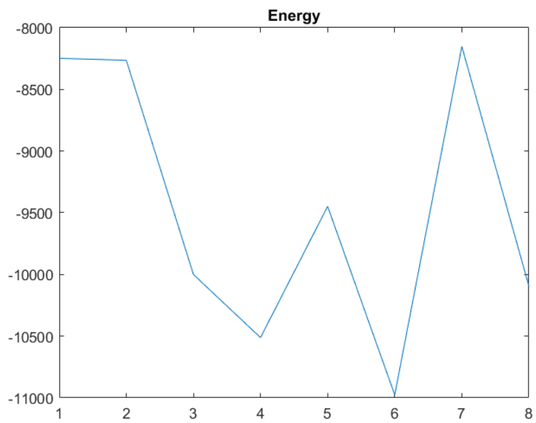

Figure 2: Energy Image

Updating the weights using hebbian learning rules allows each neuron to update its parameters according to its own input and output data. This learning rule usually uses a bipolar activation function that updates its own weights with the product of the current neuron's input and output, and then zeroes out the main diagonal before it can be used to train the network.

Introducing an energy function to the hopfield network, a complex image with many minimal points can be obtained, and the energy function image diagram is shown in Figure 2.

The hopfield network designed for this experiment implemented both synchronous and asynchronous learning procedures, and the performance of the recall mode of the network was compared and tested by using clean patterns and patterns containing 25% noise, respectively. Finally, pattern 7 and 8 were cited as confusion matrices to test the recall performance of the network.

2 Synchronous Learning Procedure

When the hopfield network performs a synchronous learning procedure, at any given moment, at least the state of two neurons will change. In special cases, all neurons may be updated at the same time, which can also be interpreted as working in full parallel.

2.1 Clean Pattern

Testing the performance of the network in recall mode with a noise-free pattern map, i.e. a clean pattern, yields the results shown in Figures 3, 4, and 5. As can be seen from the three results figures, the performance of the network in the recall mode with clean patterns can be achieved with 100% accuracy.

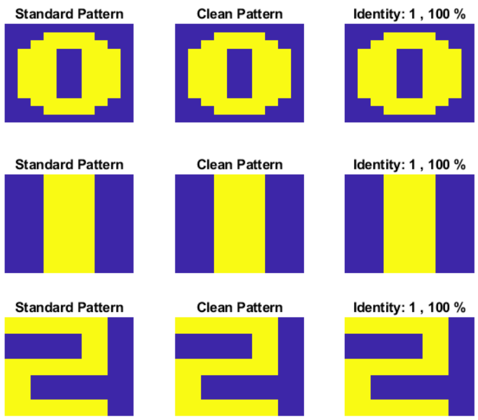

Figure 3: Synchronous with Clean Pattern

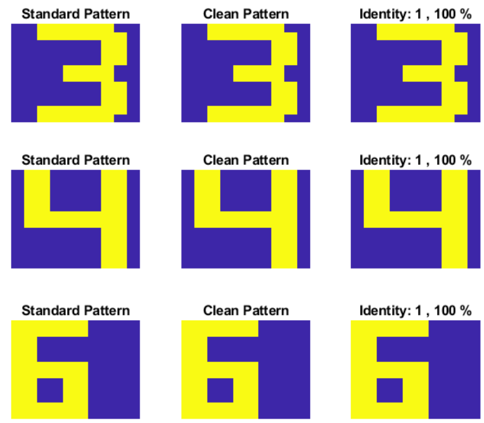

Figure 4: Synchronous with Clean Pattern

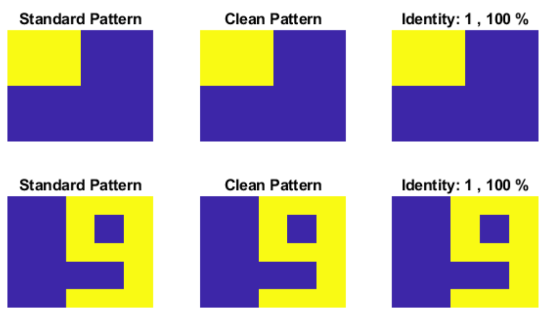

Figure 5: Synchronous with Clean Pattern

2.2 Noise Pattern

As the pattern matrix is 12 x 10, i.e. 120 points, a pattern with 25% noise only requires 30 points in the pattern matrix to be reversed at random.

As noise is generated randomly and different noise generation scenarios may affect the performance of the network, each pattern was tested at least five times. The time taken for the energy function to reach the equilibrium point for each pattern was recorded according to the number of iterations. Also, the accuracy of the network recognition performance can be obtained by calculating the results based on the final recognition with the original patterns. Each pattern was tested five times. The collated data are shown in Table 1.

	$Pattern_0$	$Pattern_1$	$Pattern_2$	$Pattern_3$	$Pattern_4$	$Pattern_6$	$Pattern_s$	$Pattern_9$
1	75%	100%	100%	63%	88%	75%	88%	75%
2	50%	100%	75%	100%	100%	75%	63%	88%
3	100%	25%	63%	88%	88%	100%	100%	25%
4	88%	75%	100%	63%	50%	63%	88%	75%
5	100%	100%	50%	100%	88%	100%	75%	63%
Total	82.6%	80%	77.6%	82.8%	82.8%	82.6%	82.8%	60.2%

Table 1: Accuracy of Noise Patterns with Synchronous Learning Procedures

The number of iterations of the synchronous learning procedure was all one. Although random noise affects the recognition performance of the network to some extent, it is clear from the data in Table 1 that the recognition performance of pattern 9 is poor. The recognition performance of all other patterns is more stable and the correct rate is high. The recognition effect of the poorly recognized pattern 9 is shown in Figure 6 and the recognition effect of the better recognized pattern 0 is shown in Figure 7.

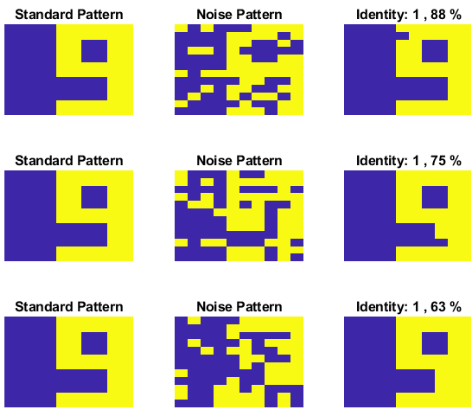

Figure 6: Poorly Recognized Pattern 9

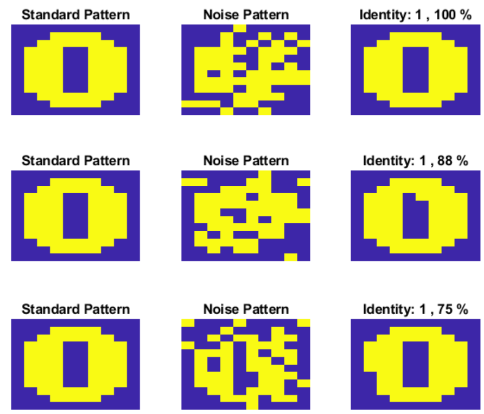

Figure 7: Better Recognized Pattern 0

3 Asynchronous Learning Procedure

When the hopfield network performs an asynchronous learning procedure, at any given moment, only one neuron's state changes and the rest of the neurons remain unchanged.

3.1 Clean Pattern

The performance of the hopfield network using the asynchronous learning procedure for clean patterns in recall mode was consistent with synchronous learning procedure, all the patterns are recalled correctly with 100% accuracy.

3.2 Noise Pattern

Following the same approach yields data for the asynchronous learning procedure, as shown in Table 2. If two or more data appear in a data cell, this indicates the recognition accuracy for a different number of iterations. The asynchronous learning procedure can reach a state of convergence by iteratively computing the noise pattern step by step closer to the original pattern.

	1	2	3	4	5
pattern_0	100%	88%-100%	50%-100%	100%	75%-100%
pattern_1	100%	100%	88%-100%	75%-100%	100%
pattern_2	100%	63%-63%	88%-100%	100%	100%
pattern_3	88%-100%	100%	88%-100%	100%	100%
pattern_4	100%	100%	100%	75%-100%	88%-100%
pattern_6	100%	100%	38%-88%-100%	100%	75%-100%
pattern_s	38%-38%	100%	63%-88%-100%	100%	100%
pattern_9	13%-25%	38%-25%	50%-25%	75%-25%	100%

Table 2: Accuracy of Noise Patterns with Asynchronous Learning Procedures

As can be seen from the data in Table 2, the hopfield network using the asynchronous learning procedure almost always achieve 100% recognition accuracy after going through the iterative process.

As with the synchronous learning procedure, pattern 9 is less well recognised. Pattern 2 and pattern s show isolated cases of incorrect recognition, but in most cases the recognition accuracy is 100%. As an example, pattern 9 with a high recall failure rate is shown in Figure 8, and pattern s with a high number of iterations is shown in Figure 9.

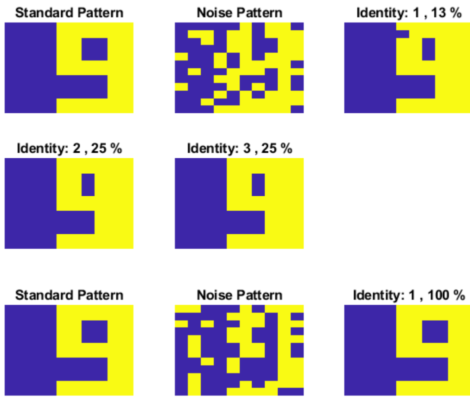

Figure 8: Poorly Recognized Pattern 9

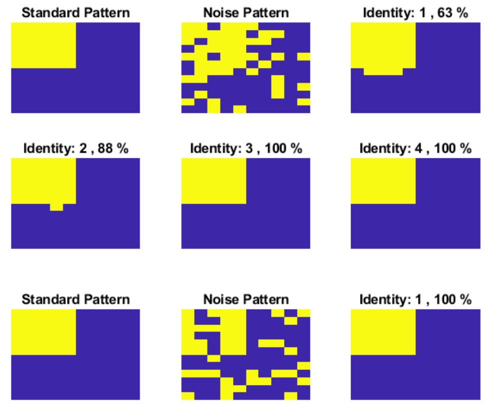

Figure 9: Better Recognized Pattern s

4 Performance

The performance analysis of the hopfield network consists of the following main parts. The first part is a comparison of the performance of synchronous and asynchronous learning. The two methods can be evaluated by comparing the accuracy of the recognition. The second part is the confusion matrix. The hopfield network is tested for recognition by introducing two untrained pattern 7 and pattern 8. The third section lists the cases of recall errors that occurred throughout the experiment.

4.1 Synchronous vs. Asynchronous

In the case of asynchronous operation, the network always converges to a balanced state. In the synchronous case, for the network to converge to an equilibrium state, the matrix needs to satisfy a non-negative definite matrix.

As can be seen from the data in Tables 1 and 2, the asynchronous learning procedure almost always achieves a recognition rate of 100% after iteration, but the recognition rate of the synchronous learning procedure depends heavily on the random noise pattern. The recognition performance of the synchronous learning procedure can reach 100% when the random noise is generated at locations close to the original pattern, but the accuracy of the recognition performance can be greatly affected when the random noise is easily confused with other patterns. Relatively speaking, the synchronous learning procedure is more prone to recognition errors than the asynchronous learning procedure.

4.2 Confuse Pattern

To verify the performance of the designed hopfield network, unlearned pattern 7 and pattern 8 were introduced as confusion matrices to test the recognition results under the synchronous and asynchronous learning procedures respectively.

From the results, although it could not achieve recognition of the two untrained patterns, the hopfield network also approximated the closest pattern to it, and with multiple iterations under the asynchronous learning procedure, the recognition of the confusion matrix could even achieve a 100% overlap with the known approximate pattern.

4.2.1 Synchronous Learning Procedure

The confusion matrix identification results under the synchronous learning procedure are shown in Figure 10 and Figure 11, where Figure 10 uses a clean confusion matrix and Figure 11 uses a confusion matrix with 25% noise.

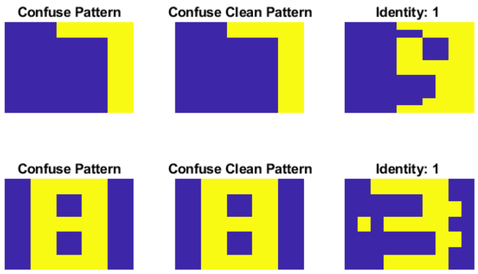

Figure 10: Clean Confuse Pattern

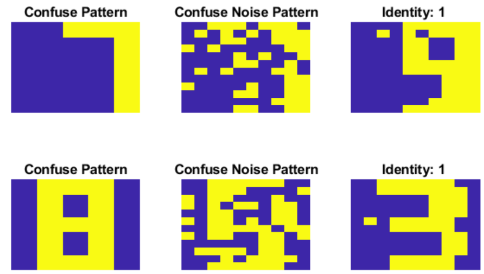

Figure 11: Noise Confuse Pattern

As can be seen in Figures 10 and 11, the hopfield network using the synchronous learning procedure approximately identifies pattern 7 as pattern 9 and pattern 8 as pattern 3, for both clean and noise patterns.

4.2.2 Asynchronous Learning Procedure

The confusion matrix identification results under the asynchronous learning procedure are shown in Figure 12 and Figure 13. As can be seen in Figures 12 and 13, with clean and noise pattern inputs, the hopfield network using the asynchronous learning procedure recognises pattern 7 as pattern 9 and pattern 8 as pattern 3, as the hopfield network using the synchronous learning procedure does, but the recognition results are much clearer and the recognition rate after iterative computation is much closer to the original pattern that has been trained.

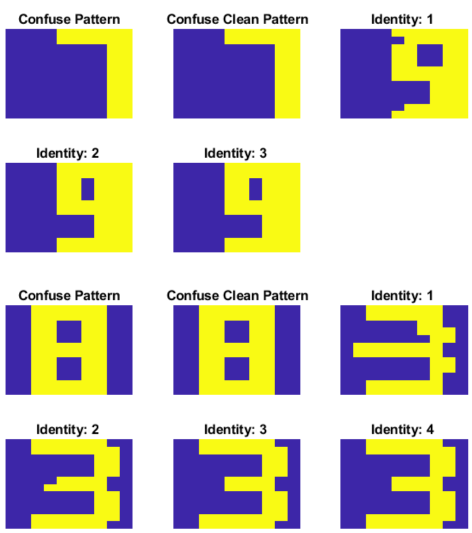

Figure 12: Clean Confuse Pattern

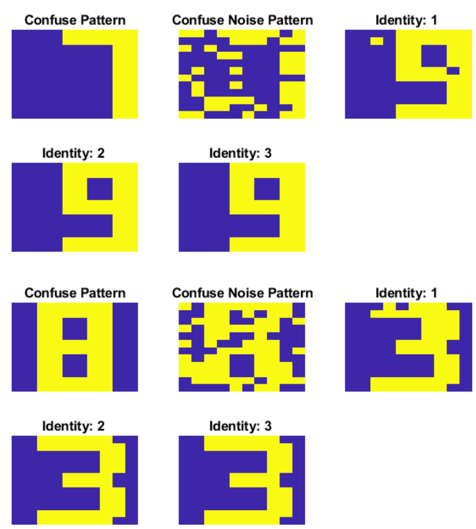

Figure 13: Noise Confuse Pattern

4.3 Mistakes

During testing of the asynchronous learning procedure, there have also been cases of misrecognition, as shown in Figures 14 and 15. As can be seen from the figures, the hopfield network incorrectly identified pattern 0 as the complement of pattern 6 and incorrectly identified pattern 4 as pattern 3. However, the probability of generating such recognition errors is small, mainly because 25% of the random noise generated happened to be at locations associated with other patterns.

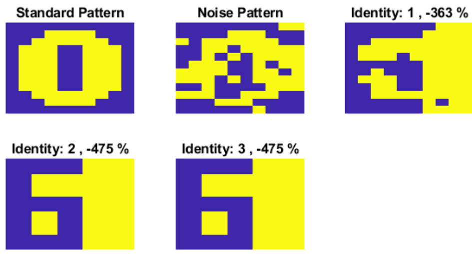

Figure 14: Mistake Pattern 0

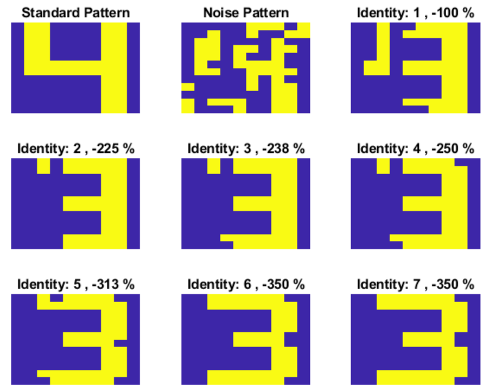

Figure 15: Mistake Pattern 4

A Appendix

A.1 Code for Hopfield Neural Network

```
clc;
close all;
clear;

%% Loading Clean Data
pattern_0=[-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  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 -1 -1 -1 -1 -1 -1 -1 -1];

pattern_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  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 -1 -1  1  1  1  1 -1 -1 -1];

pattern_2=[ 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  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  1  1  1  1  1  1 -1 -1];

pattern_3=[-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  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  1  1  1  1  1  1 −1 −1];

pattern_4=[−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  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 −1 −1 −1 −1 −1  1  1 −1];

pattern_6=[ 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 −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  1  1  1  1 −1 −1 −1 −1];

pattern_s=[ 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 −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 −1 −1 −1 −1 −1 −1 −1 −1];

pattern_9=[−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  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 −1 −1  1  1  1  1  1  1];
```

```
%% Load Confuse Matrix
pattern_7=[-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 -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 -1 -1 -1 -1 -1 -1  1  1];

pattern_8=[-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  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  1  1  1  1  1  1 -1 -1];

%% Build Clean Pattern Array
d(:,:,1)=pattern_0;
d(:,:,2)=pattern_1;
d(:,:,3)=pattern_2;
d(:,:,4)=pattern_3;
d(:,:,5)=pattern_4;
d(:,:,6)=pattern_6;
d(:,:,7)=pattern_s;
d(:,:,8)=pattern_9;
d(:,:,9)=pattern_7;      %confuse matrix
d(:,:,10)=pattern_8;     %confuse matrix

for i=1:8
   B(i,:)=reshape(d(:,:,i),1,120);
end

%% Weight Calculate: Hebbian Rule
T=zeros(120);
for i=1:8   %confuse pattern
%for i=1:size(d,3)
    T=T+B(i,:)'*B(i,:);
end
w=T;
for i=1:size(w,1)    %set diagonal elements to 0
    w(i,i)=0;
end
xlswrite('weight.xlsx',w)

%% Energy Function
```

```
Et=zeros(1,8);
for p=1:8
Et(p)=-0.5*B(p,:)*w*B(p,:).'-B(p,:)*B(p,:).';
end
xlswrite('energy.xlsx',Et)

figure(1)
plot(Et)
title('Energy')

%% Pattern Select
p=10;

figure(2)
subplot(3,3,1)
%imagesc(reshape(d(:,:,p),12,10))
%title('Standard Pattern')
imagesc(reshape(d(:,:,p),12,10))
title('Confuse Pattern')
axis off

%% Load Noise Data
noise_pattern_0=pattern_0;
noise_pattern_1=pattern_1;
noise_pattern_2=pattern_2;
noise_pattern_3=pattern_3;
noise_pattern_4=pattern_4;
noise_pattern_6=pattern_6;
noise_pattern_s=pattern_s;
noise_pattern_9=pattern_9;

rate_noise=30;    %rate of noise=rate*row*column

i_0 = randperm(numel(noise_pattern_0),rate_noise);
for index = i_0
    noise_pattern_0(index)= -noise_pattern_0(index);
end

i_1 = randperm(numel(noise_pattern_1),rate_noise);
for index = i_1
    noise_pattern_1(index)= -noise_pattern_1(index);
end

i_2 = randperm(numel(noise_pattern_2),rate_noise);
for index = i_2
    noise_pattern_2(index)= -noise_pattern_2(index);
end

i_3 = randperm(numel(noise_pattern_3),rate_noise);
for index = i_3
    noise_pattern_3(index)= -noise_pattern_3(index);
end

i_4 = randperm(numel(noise_pattern_4),rate_noise);
for index = i_4
    noise_pattern_4(index)= -noise_pattern_4(index);
```

```
end

i_6 = randperm(numel(noise_pattern_6),rate_noise);
for index = i_6
    noise_pattern_6(index)= -noise_pattern_6(index);
end

i_s = randperm(numel(noise_pattern_s),rate_noise);
for index = i_s
    noise_pattern_s(index)= -noise_pattern_s(index);
end

i_9 = randperm(numel(noise_pattern_9),rate_noise);
for index = i_9
    noise_pattern_9(index)= -noise_pattern_9(index);
end

%% Load Confuse Noise Data
noise_pattern_7=pattern_7;
noise_pattern_8=pattern_8;

rate_noise=30;    %rate of noise=rate*row*column

i_7 = randperm(numel(noise_pattern_7),rate_noise);
for index = i_7
    noise_pattern_7(index)= -noise_pattern_7(index);
end

i_8 = randperm(numel(noise_pattern_8),rate_noise);
for index = i_8
    noise_pattern_8(index)= -noise_pattern_8(index);
end

%% Build Noise Pattern Array
d(:,:,1)=noise_pattern_0;
d(:,:,2)=noise_pattern_1;
d(:,:,3)=noise_pattern_2;
d(:,:,4)=noise_pattern_3;
d(:,:,5)=noise_pattern_4;
d(:,:,6)=noise_pattern_6;
d(:,:,7)=noise_pattern_s;
d(:,:,8)=noise_pattern_9;
d(:,:,9)=noise_pattern_7;    %confuse pattern
d(:,:,10)=noise_pattern_8;    %confuse pattern

J=d(:,:,p);
B_noise=reshape(J,1,120);
B(p,:)=B_noise;

subplot(3,3,2)
imagesc(reshape(B(p,:),12,10))
title('Confuse Noise Pattern')
axis off

%% Energy Function
E0=-0.5*B(p,:)*w*B(p,:).'-B(p,:)*B(p,:).';
```

```
E1=0;
y0=B(p,:);
y1=y0;

%% Learning Parameters
iterations=1;
converge=1;     %Convergence flag
while converge
%% Asynchronous Learning Procedures
%       index=randperm(120);
%       for i=1:120
%           y1(index(i))=sign(B(p,index(i))+y1*w(index(i),:).');
%       end

%% Synchronous Learning Procedures
    y1=sign(B(p,:)+y0*w.');

%% Recall Pattern
    E1=-0.5*y1*w*y1.'-B(p,:)*y1.';
    if (E1-E0)==0  %reach equilibrium
        converge=0;
    end
    E0=E1;

    %accuracy=(1-abs(1/8*sum(y1~=pattern_9(:)')))*100;

    subplot(3,3,iterations+2)
    imagesc(reshape(y1,12,10))
    %title(sprintf('%s %s %s %.f %s','Identity:',num2str(iterations),',',accuracy,'%'))
    title(sprintf('%s %s %s %.f','Identity:',num2str(iterations)))  %confuse pattern
    axis off

    iterations=iterations+1;
end
```