Lab4 Hopfield Network

5 Hebbian learning and Hopfield memory

Question1

- Translate the calculation of the weight matrix and the update rule into Matlab expressions.

The coefficients for the weight matrix can then be written as:

$$w_{ij} = \frac{1}{N}\sum_{\mu=1}^{P} x_i^\mu x_j^\mu$$

where μ is index within a set of patterns, P is the number of patterns, and N is the number of units.

To recall a pattern of activation $\bar{x}$ in this network we can use the following update rule:

$$x_i \leftarrow \text{sign}\left(\sum_j w_{ij} x_j\right)$$

where

$$\text{sign}(x) = \begin{cases} 1 & x > 0 \\ -1 & x < 0 \end{cases}$$

Translate the calculation of the weight matrix into Matlab expressions: w = x'x;

Note: Weight will be positive if the values of i and j will tend to become equal, but weight will be negative if the bits corresponding to neurons i and j are different.

Translate the update rule into Matlab expressions: x_update = sgn(x_previous*w);

A final weight matrix is symmetric. The diagonal elements are zeros.

w

8x8 double

	1	2	3	4	5	6	7	8
1	0	1	-1	3	1	-1	3	-1
2	1	0	1	1	-1	1	1	1
3	-1	1	0	-1	1	-1	-1	3
4	3	1	-1	0	1	-1	3	-1
5	1	-1	1	1	0	-3	1	1
6	-1	1	-1	-1	-3	0	-1	-1
7	3	1	-1	3	1	-1	0	-1
8	-1	1	3	-1	1	-1	-1	0

Question2

- Check if the network was able to store all three patterns.

The final network is able to store all three patterns, i.e. the output of the network is the same as the input.

```
Three test patterns

x1 =

    -1    -1     1    -1     1    -1    -1     1


x2 =

    -1    -1    -1    -1    -1     1    -1    -1


x3 =

    -1     1     1    -1    -1     1    -1     1

The network is able to store all three patterns after 1 iterations.
The network converges and reaches the fixpoints.


The final network is

x_net =

    -1    -1     1    -1     1    -1    -1     1
    -1    -1    -1    -1    -1     1    -1    -1
    -1     1     1    -1    -1     1    -1     1
```

5.1 Convergence and attractors

Can the memory recall the stored patterns from distorted inputs patterns? Define a few new patterns which are distorted versions of the original ones.

original ones：

```
>> x1=vm([0 0 1 0 1 0 0 1])
>> x2=vm([0 0 0 0 0 1 0 0])
>> x3=vm([0 1 1 0 0 1 0 1])
```

distorted ones：

```
>> x1d=vm([1 0 1 0 1 0 0 1])
>> x2d=vm([1 1 0 0 0 1 0 0])
>> x3d=vm([1 1 1 0 1 1 0 1])
```

The first, second and fifth patterns are distorted [1,2,5].

Question1

- Apply the update rule repeatedly until you reach a stable fixpoint. Did all the patterns converge towards stored patterns?

```
x1 =

    -1    -1     1    -1     1    -1    -1     1


x2 =

    -1    -1    -1    -1    -1     1    -1    -1


x3 =

    -1     1     1    -1    -1     1    -1     1

The network is able to store all three patterns after 1 iterations.

x_net =

    -1    -1     1    -1     1    -1    -1     1
    -1    -1    -1    -1    -1     1    -1    -1
    -1     1     1    -1    -1     1    -1     1
```

The network stores the all the patterns. All the patterns converge towards stored pattern. We find that that x1, x2 and x3 are attractors in this network.

Question2

- How many attractors are there in this network? Hint: automate the searching.

Try to recall the stored version from slightly distorted version.
We find only one attractor.

```
x_net =

    -1    -1     1    -1     1    -1    -1     1
    -1    -1    -1    -1    -1     1    -1    -1
    -1     1     1    -1    -1     1    -1     1


x_recall =

    -1    -1     1    -1     1    -1    -1     1
    -1    -1    -1     1    -1     1     1    -1
    -1     1     1    -1    -1    -1    -1     1


Count_attractors_1 =

     1
```

Question3

- What happens when you make the starting pattern even more dissimilar to the stored ones (e.g. more than half is wrong)?

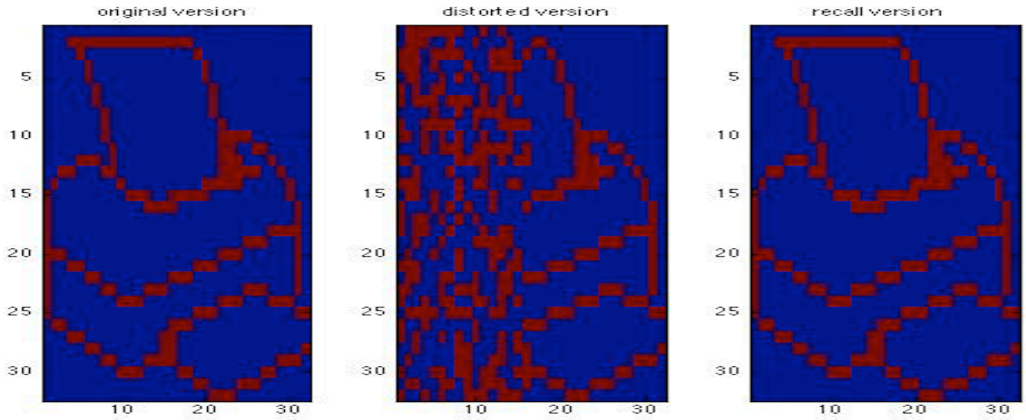
Try to recall the stored version from severely distorted version.
We find no one attractor.

```
x_recall =

     1    -1     1     1     1    -1     1    -1
     1     1     1     1     1     1     1    -1
    -1     1    -1     1    -1     1    -1     1


Count_attractors_2 =

     0
```

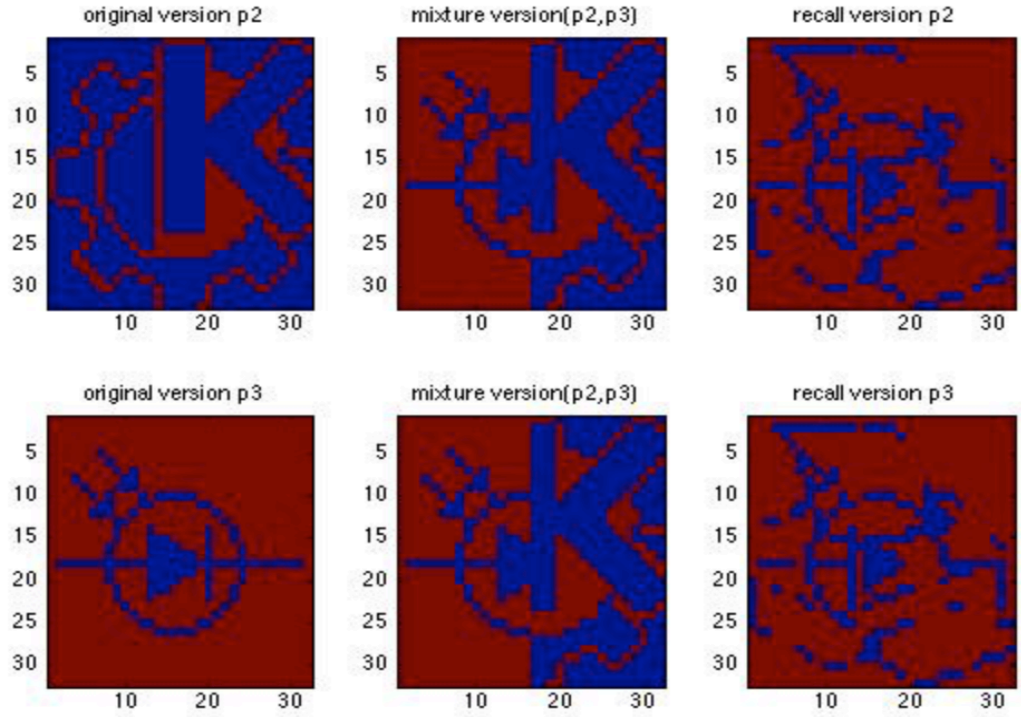
5.2 Sequential Update

Question1

- Can the network complete a degraded pattern? Try the pattern `p11`, which is a degraded version of `p1`, or `p22` which is a mixture of `p2` and `p3`.



The network can complete a degraded pattern (from left to right: origin-distorted-recall degraded pattern).



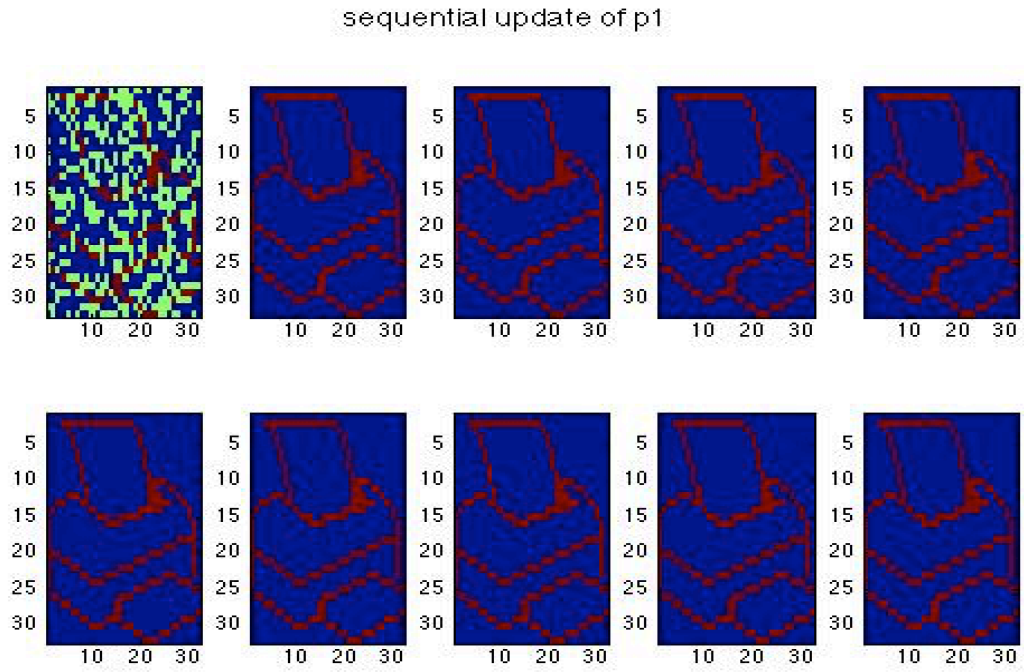
The network can complete a degraded mixture pattern.

Question2

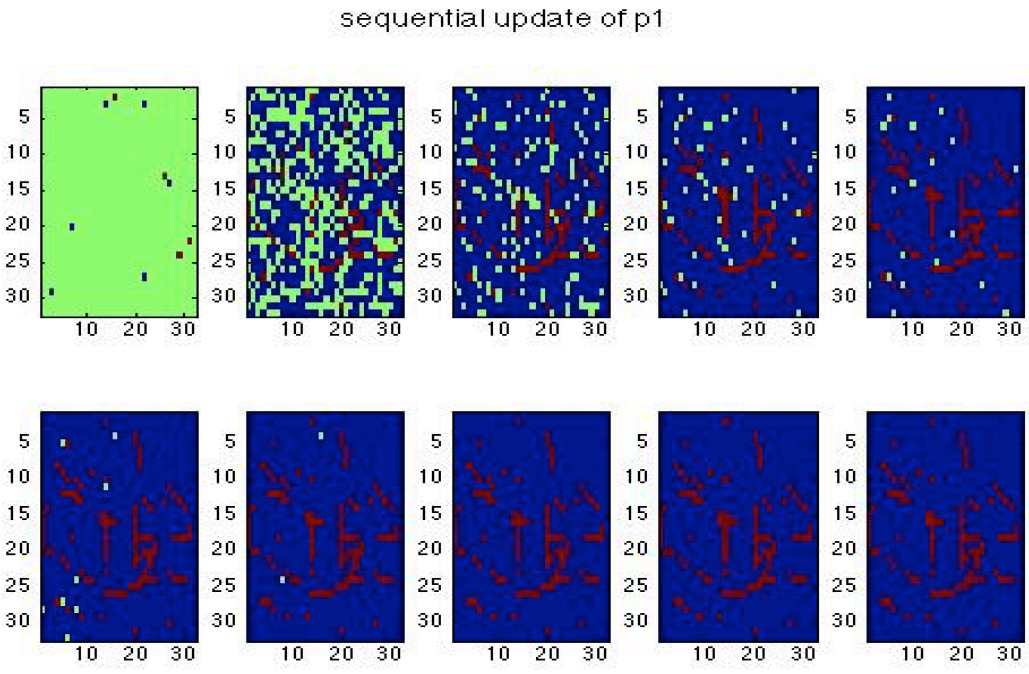
- Clearly convergence is practically instantaneous. What happens if we select units randomly, calculate their new state and then repeat the process (the original sequential Hopfield dynamics)? Write a matlab script that does this, showing the image every hundredth iteration or so.

If we select units randomly to calculate their new state with ***patterns chosen to be learned.*** **Example: Try with different amount of random units.**

1000 units: With 1000 units, it takes less than 100 iterations to converge



10 units: With 10 units, it takes more than 600 iterations to converge.



Conclusion: With more amount of random units, the network takes less iteration to converge (converges faster) and have better performance.

5.3 Energy

Energy function:

$$E = -\sum_i \sum_j w_{ij} x_i x_j$$

Question1:

- How do you express this calculation in Matlab? (Note: you do not need to use any loops!)

x = [p1;p2;p3];
w = x'*x; % calculate weight matrix
w = w -diag(diag(w)); % diagonal elements are zeros
E = -sum(sum(w.*(x_net'*x_net),1),2);
%x_net means the current states of the network

Question2:

- What is the energy at the different attractors?

This means energy for different pattern.
E = -sum(sum(w.*(x_net(i,:)'* x_net(i,:)),1),2);
% x_net(i,:) means the current states of the network
Energy for three different patterns:

```
E =

    -1470864    -1395344    -1494272
```

Question3:

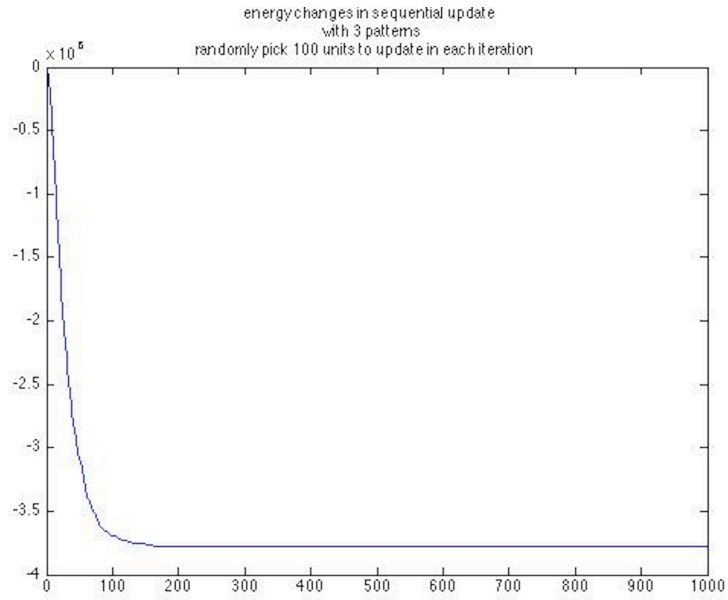
- What is the energy at the points of the distorted patterns?

E = -sum(sum(w.*(p11'* p11),1),2);
% p11 means the distorted pattern

Question4:

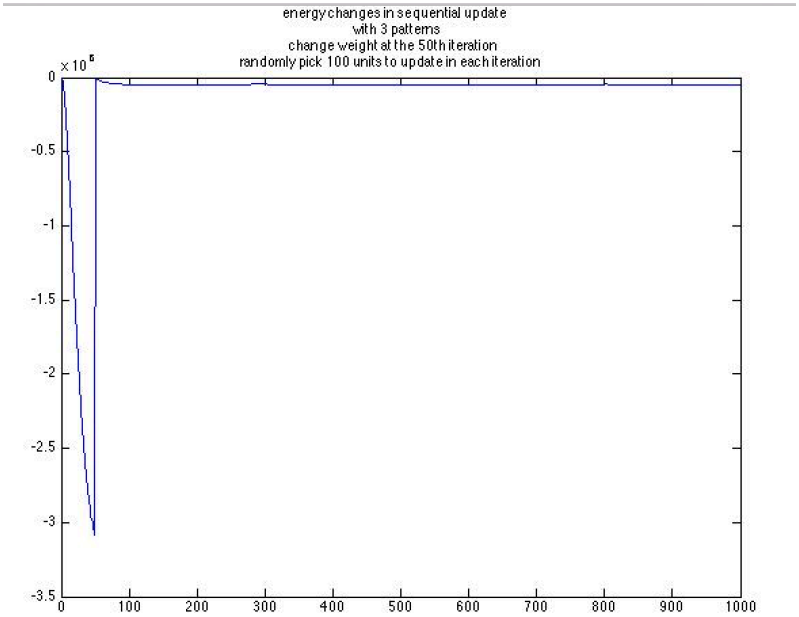
- Follow how the energy changes from iteration to iteration when you use the sequential update rule to approach an attractor.

The energy decreases from iteration to iteration. It finally converges to the local minima.



Question5:

- Generate a weight matrix by setting the weights to normally distributed random numbers, and try iterating an arbitrary starting state. What happens?

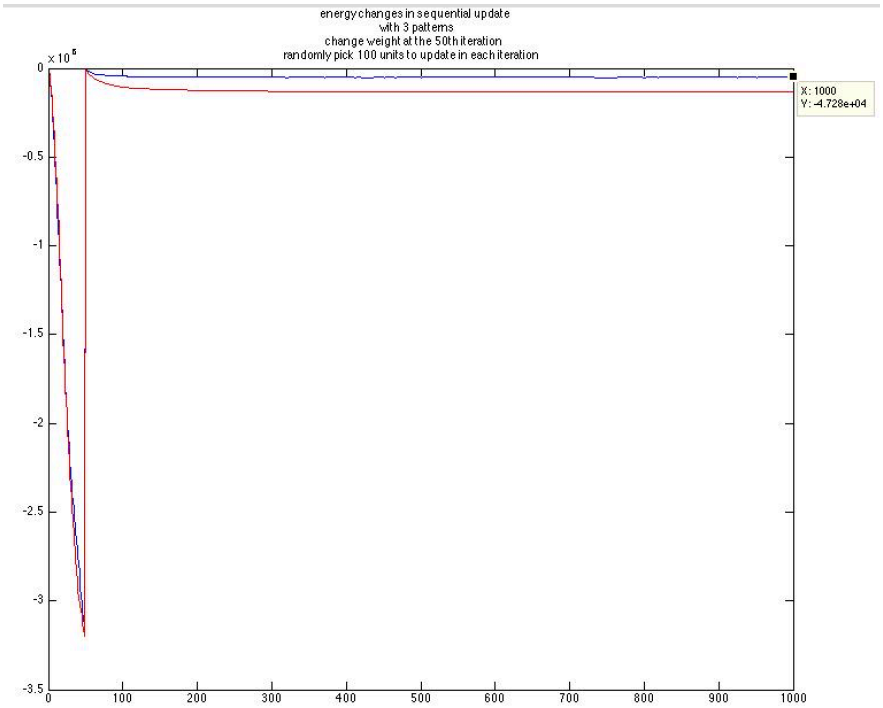


When try iterating an arbitrary starting state (at the 50th iteration), the energy starts to grow from iteration to iteration. It finally converges to the initial level.

The reason is because this weight matrix is totally different from the desired weight matrix. This wrong matrix destroys the desired update and therefore making the energy of the network high.

Question6:

- Make the weight matrix symmetric (e.g. by setting `w=0.5*(w+w')`). What happens now? Why?

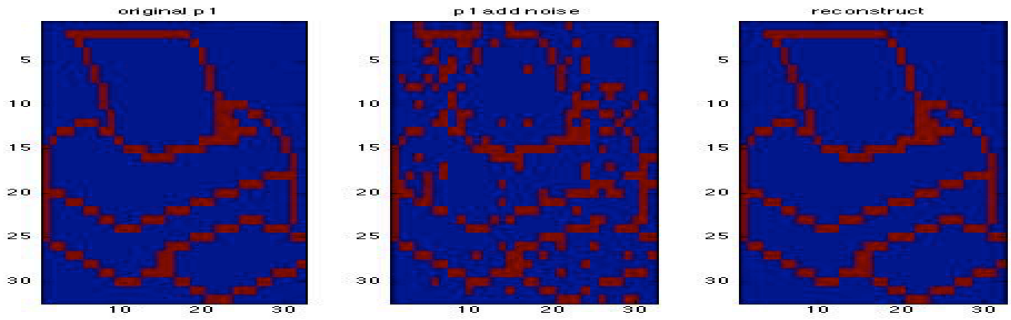


The red line is the one with symmetric weight (normally distributed random numbers). The result is similar to the one with non-symmetric weight. But it finally converges to a relative lower right level compared to the one with symmetric weight.
The reason is because the network is internally symmetric. A weight of a mutual connection between two nodes is same: wij=wji.

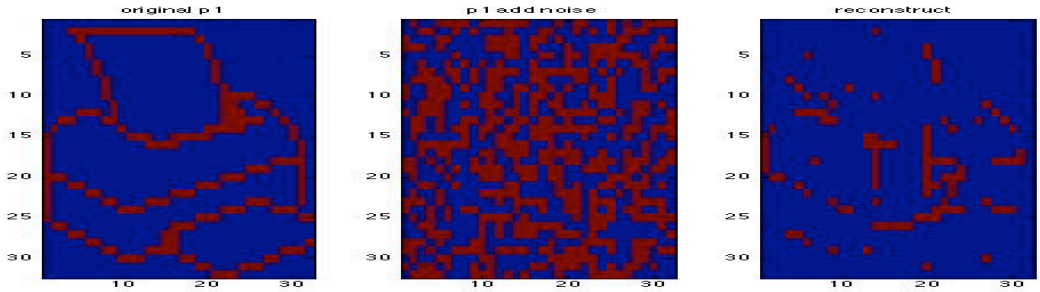
5.4 Distortion Resistance

- How much noise can be removed?

Step1: train a network with p1, p2, p3;
Step2: add noise to a pattern;
Step3: iterate it a number of times and check whether it has been successfully restored.
Result:
1) When adding 100 noise points, the network is able to restore the correct pattern.



2) When adding 500 noise points, the network is not able to restore the correct pattern.



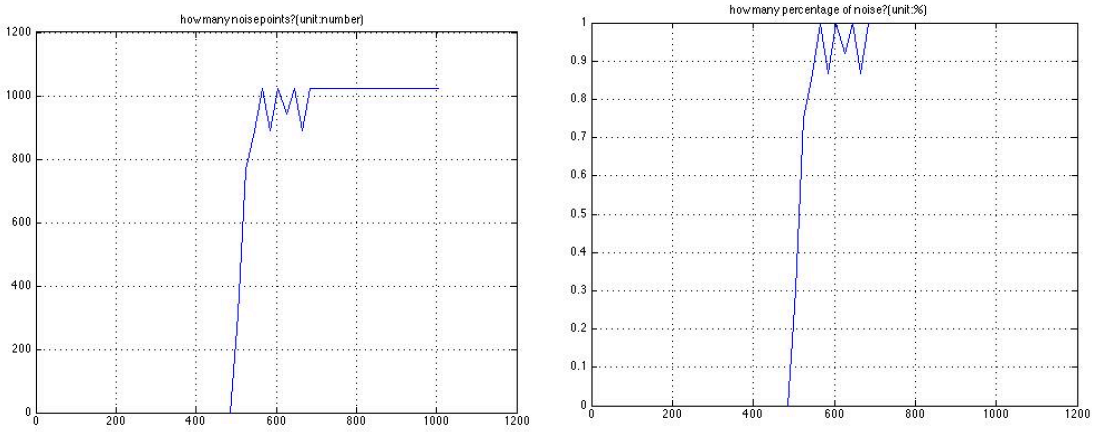
3) Let the script run across 0 to 100% noise and plot the result.

When adding less than around 500 noise points, the network is able to restore the correct pattern. All noise points are compressed.

When adding more than around 500 noise points, the network is not able to restore the correct pattern. There are a lot of noise points.

Left: how many noise points in the reconstructed noisy pattern?

Right: how many percentage of noise in the reconstructed noisy pattern?



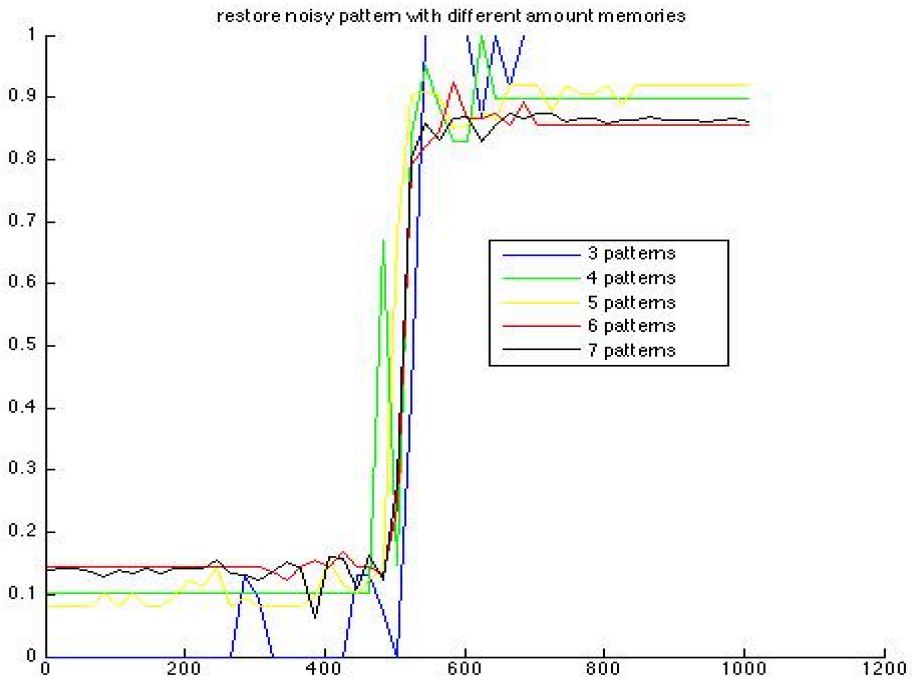
5.5 Capacity

Start by adding p4 into the weight matrix and check if moderately distorted patterns can still be recognized. Then continue by adding others such as p5, p6 and p7 in some order and checking the performance after each addition.

Add p4, p5, p6 and p7 to check the restoring performance:

Result:

It becomes harder for the network to restore a distorted pattern p1 and compress noise for it.



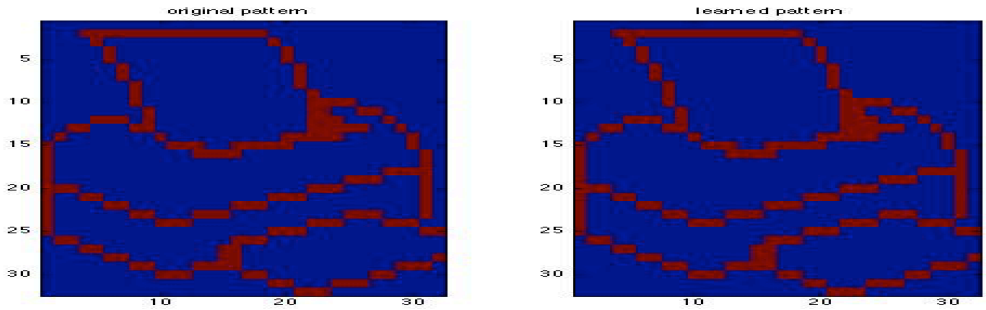
Question1

- How many patterns could safely be stored? Was the drop in performance gradual or abrupt?

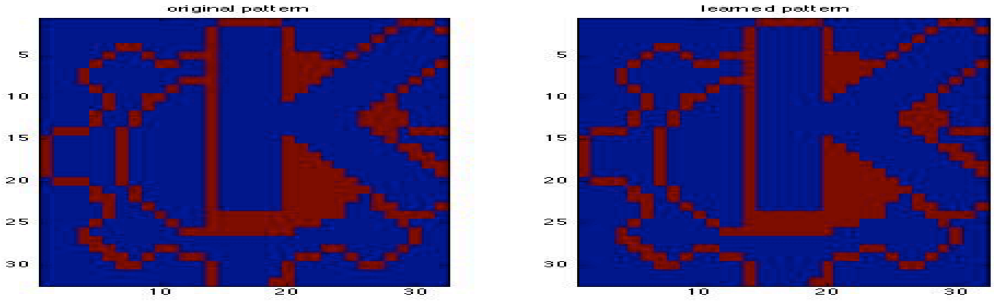
When learning 3 patterns, all 3 patterns can be safely stored. But when learning 4 patterns, none of the pattern can be safely stored. When learning 5 patterns, the performance is even worse.

In fact, when learning more than 4 patterns, none of the pattern can be safely stored. The drop in performance is abrupt.

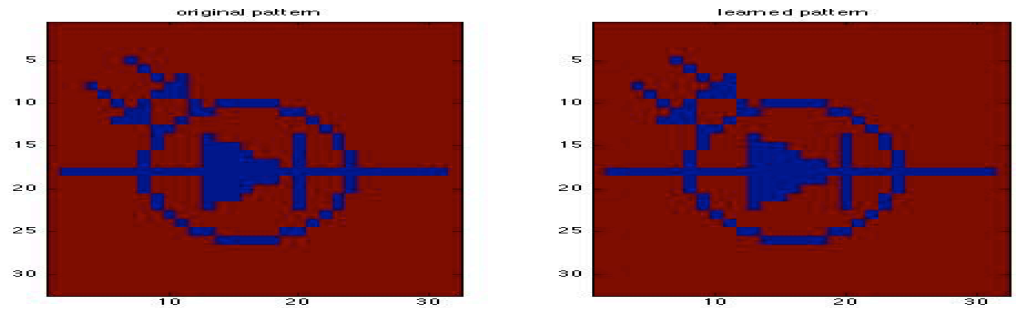
3 patterns - P1



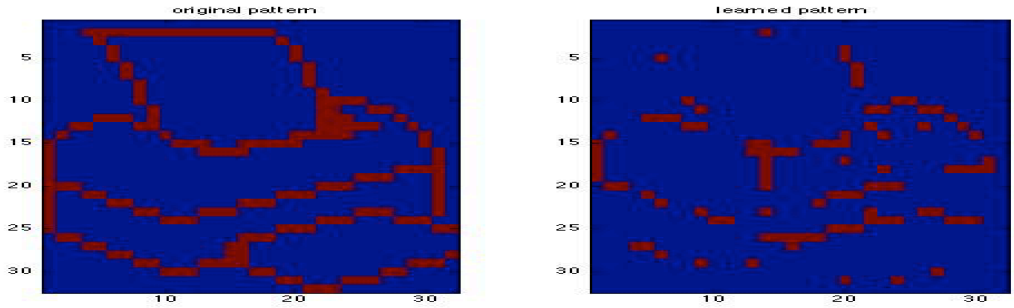
3 patterns – P2



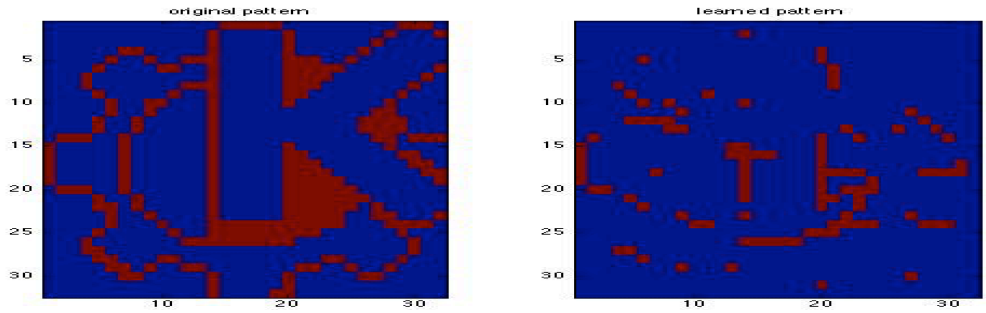
3 patterns – P3



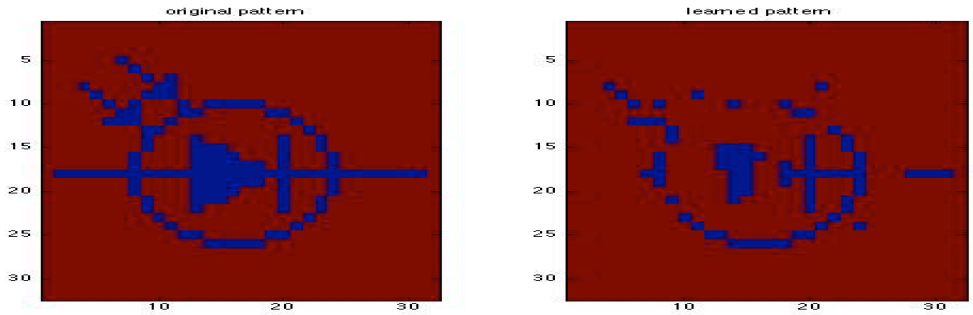
4 patterns - P1



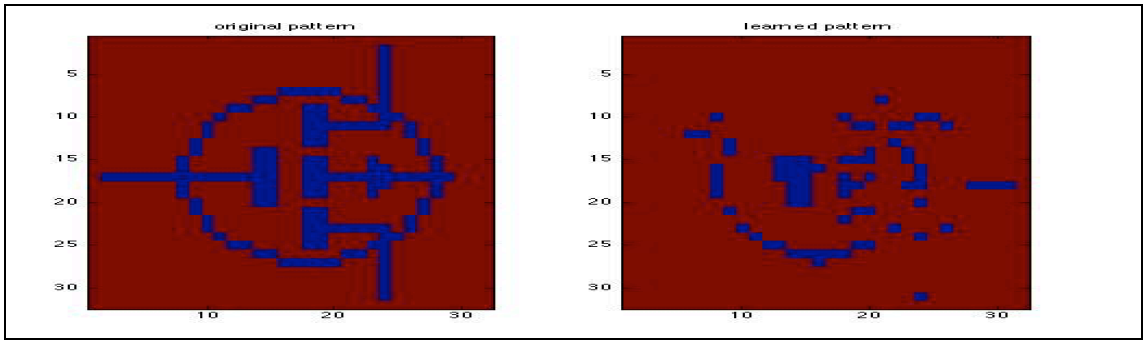
4 patterns – P2



4 patterns – P3



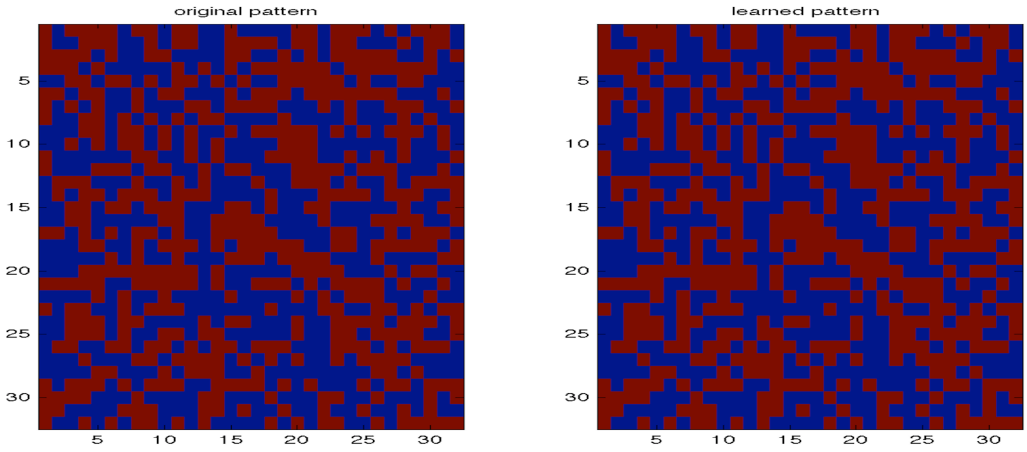
4 patterns – P4



Question2:

- Try to repeat this with learning a few random patterns instead of the pictures and see if you can store more. You can use `sgn(randn(1,1024))` to easily generate the patterns.

An origin pattern and its learned version: hard to compare with eyes!



Compare with command: isequal()

Try with 3 random patterns:

```
Pattern 1 is learned
Pattern 2 is learned
Pattern 3 is learned
```

Try with 4 random patterns:

```
Pattern 1 is learned
Pattern 2 is learned
Pattern 3 is learned
Pattern 4 is learned
```

Try with 5 random patterns:

```
Pattern 1 is learned
Pattern 2 is learned
Pattern 3 is learned
Pattern 4 is learned
Pattern 5 is learned
```

…

To try with more random patterns, we try from 3 patterns to 100 patterns. **When more than 50 random patterns, the network will not be able to store the patterns.**

Question3:

- It has been shown that the capacity of a Hopfield network is around $0.138N$. How do you explain the difference between random patterns and the pictures?

The capacity decreases when learning strong patterns, the pictures.
Strong patterns are harder to learn comparing with random patterns.
http://www.doc.ic.ac.uk/~ae/papers/Hopfield-networks-15.pdf
Strong patterns are strongly stable and have large basins of attraction compared to simple patterns. The training process is to minimize the energy function and reach to the basin of attraction. Since strong patterns have large basins of attraction, it is harder to reach these basins, and thus it is harder to reach the local minima.

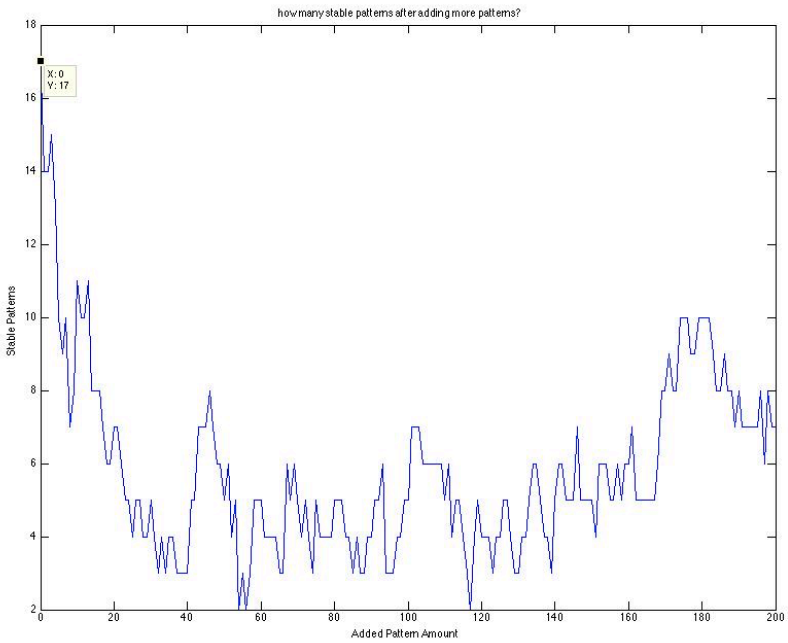
Question4

Create 300 random patterns (`sign(randn(300,100))` is a quick way) and train a 100 unit (or larger) network with them. After each new pattern has been added to the weight matrix, calculate how many of the earlier patterns remain stable (a single iteration does not cause them to change) and plot it.

- What happens with the number of stable patterns as more are learned?

STEP 1:Create 300 random patterns
STEP 2:train a 100 unit network with different amount of random patterns
STEP3:
1) add new patterns to the weight matrix
2) run the network with little model for one iteration
3) calculate how many earlier patterns remain stable

STEP 4:Plot it



With good luck, we can get 17 stable patterns when training a 100 unit network. When using more amounts of random patterns to train the network, **the network has**

less stable patterns.

Question5

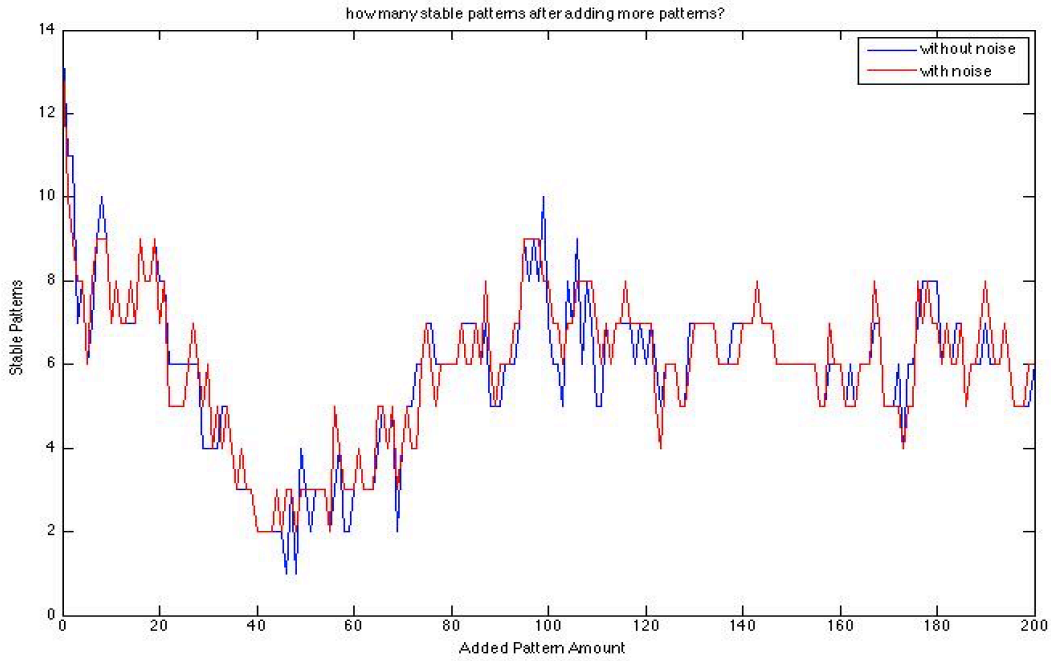
- What happens if convergence to the pattern from a noisy version (a few flipped units) is used? What does the different behavior for large number of patterns mean?

STEP 1:Create 300 random patterns

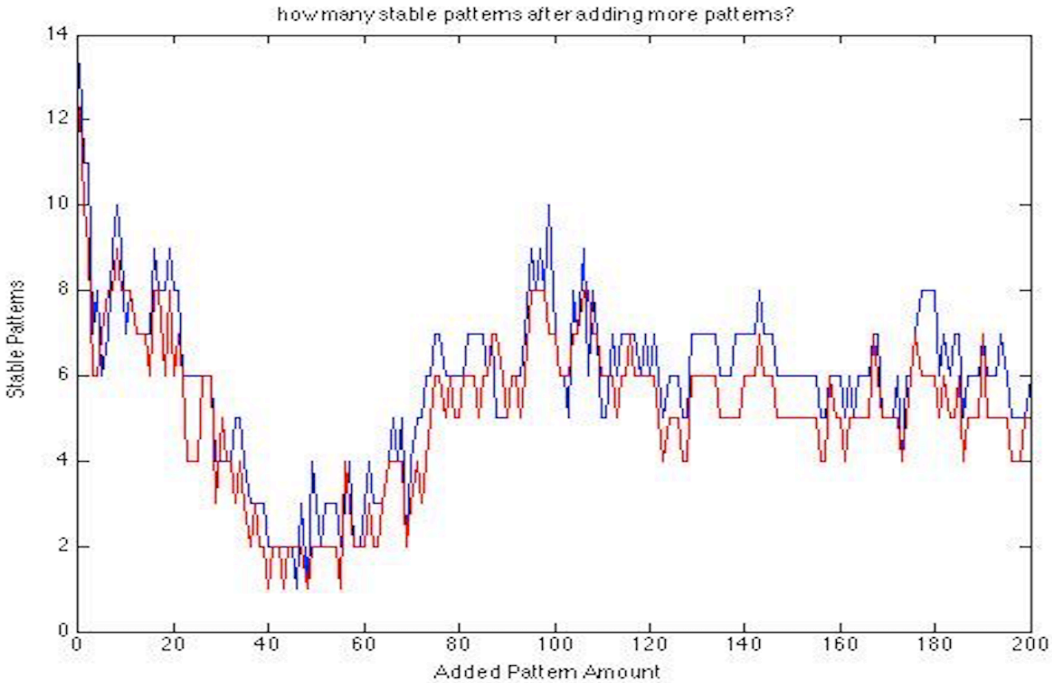
STEP 2:train a 100 unit network with different amount of random noisy patterns

STEP 3:Plot it

Adding 3 noise points:



Adding 5 noise points:



With smaller amount of noise (3), the network has similar performance compared to network trained with no noise.

With larger amount of noise (5), the network has slightly worse performance when the number of learned pattern is large compared to network trained with no noise.

Question6

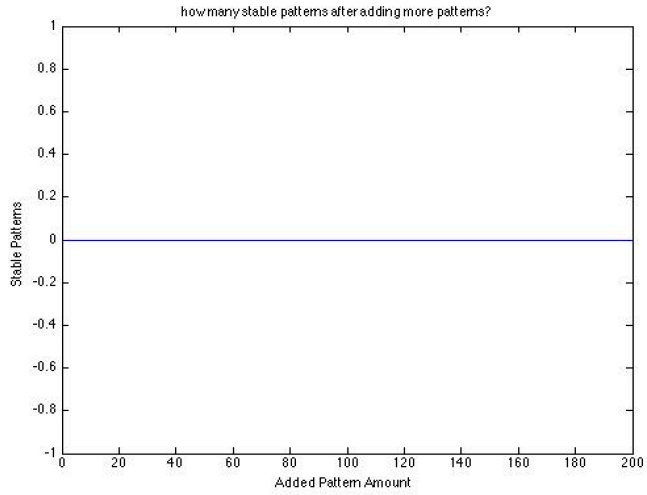
- What is the maximum number of retrievable patterns for this network?

The maximum amount is 14 (Store noisy patterns).

Question7

- What happens if you bias the patterns, e.g. use `sign(0.5+randn(300,100))` or something similar to make them contain more +1? How does this relate to the capacity results of the picture patterns?

When using bias pattern, we do not get any stable patterns. the capacity of the network decreases drastically.



5.6 Sparse Patterns

Here we will use binary (0,1) patterns, since they are easier to use than bipolar (± 1) patterns in this case and it makes sense to view the "ground state" as zero and differing neurons as "active". If the average activity $\rho = (1/NP)\sum_\mu \sum_i x_i^\mu$ is known, the learning rule can be adjusted to deal with this imbalance:

$$w_{ij} = \sum_{\mu=1}^{P}(x_i^\mu - \rho)(x_j^\mu - \rho)$$

This produces weights that are still on average zero. When updating, we use the slightly updated rule

$$x_i \leftarrow 0.5 + 0.5 * \text{sign}(\sum_j w_{ij}x_j - \theta)$$

where θ is a bias term.

- Try generating sparse patterns with just 10% activity and see how many can be stored for different values of θ (use a script to check different values of the bias).
- What about even sparser patterns ($\rho = 0.05$ or 0.01)?

Very hard to learn spare patterns