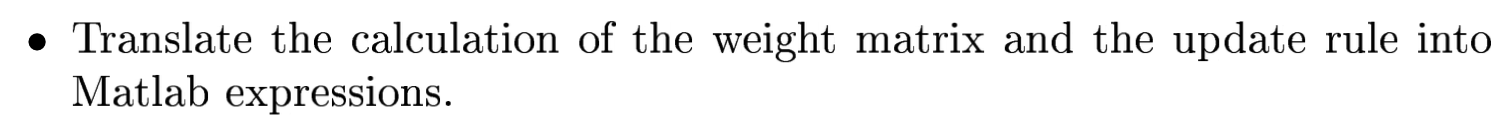
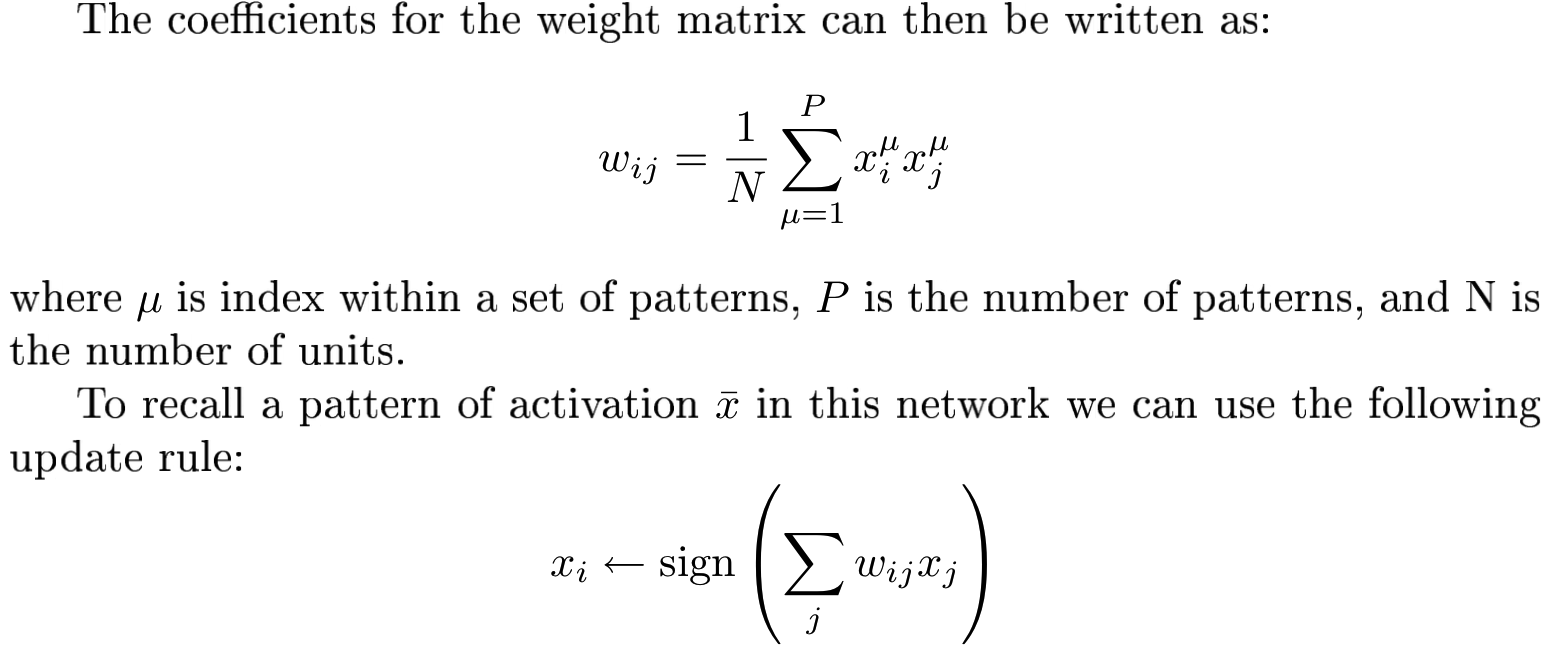
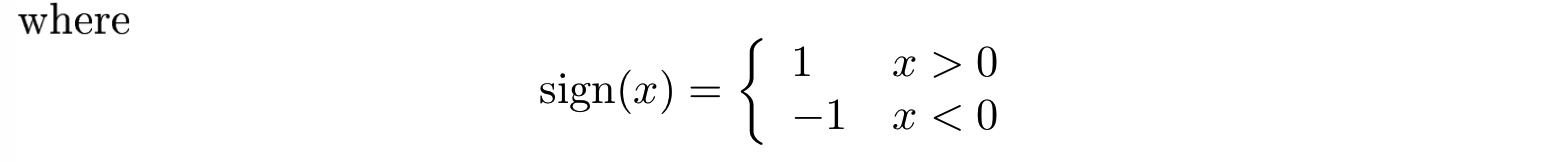
**Lab4 Hopfield Network**



**Question1**





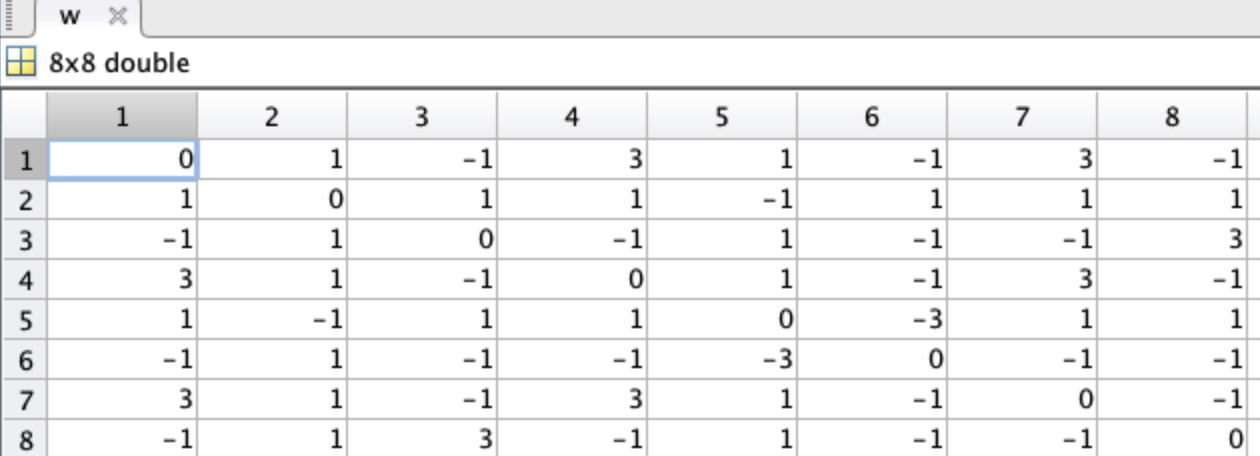


**Translate the calculation of the weight matrix into Matlab expressions: w = x’x;**

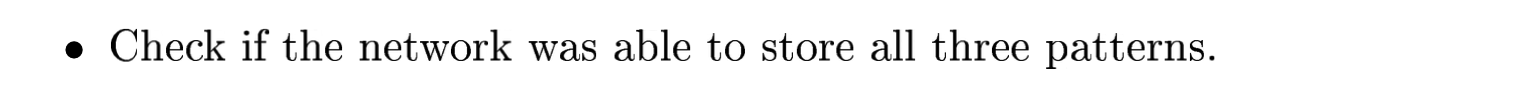
*Note: Weight will be positive if the values of i and j will tend to become equal, but weight will be negative if the bits corresponding to neurons i and j are different.*

**Translate the update rule into Matlab expressions: x\_update = sgn(x\_previous\*w);**

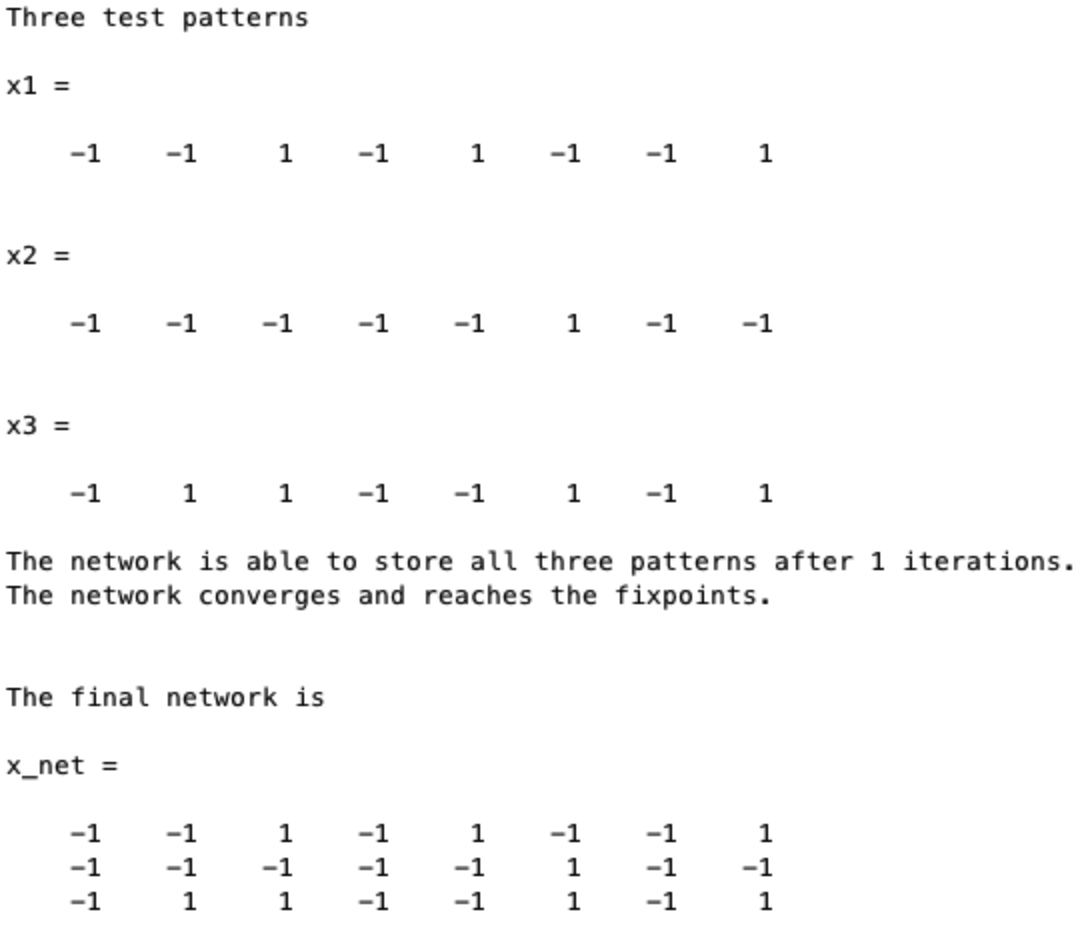
A final weight matrix is symmetric. The diagonal elements are zeros.

****

**Question2**



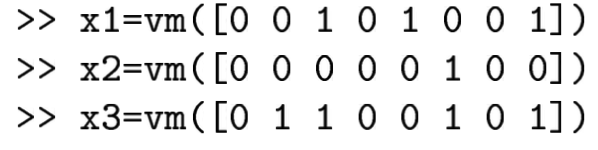
The final network is able to store all three patterns, i.e. the output of the network is the same as the input.



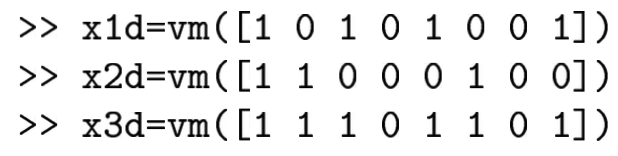


**Can the memory recall the stored patterns from distorted inputs patterns? Define a few new patterns which are distorted versions of the original ones.**

original ones：

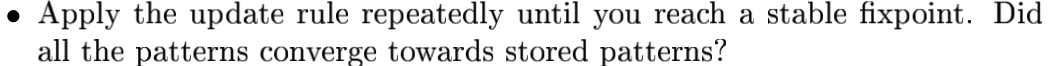


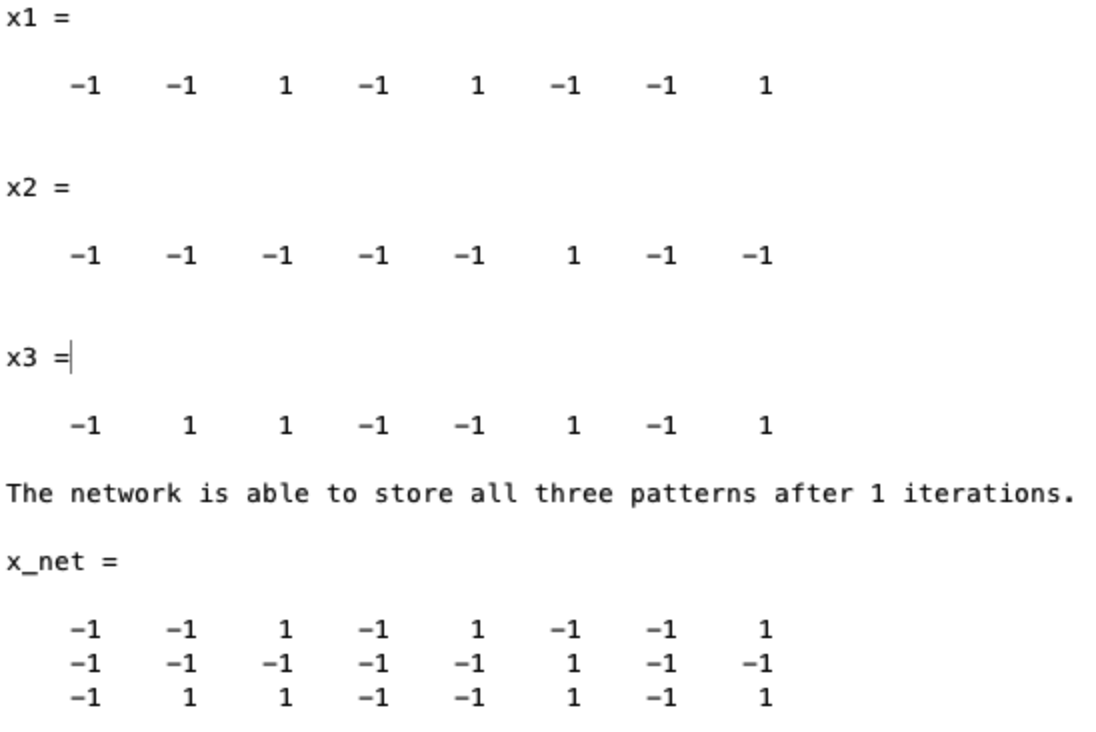
distorted ones：



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

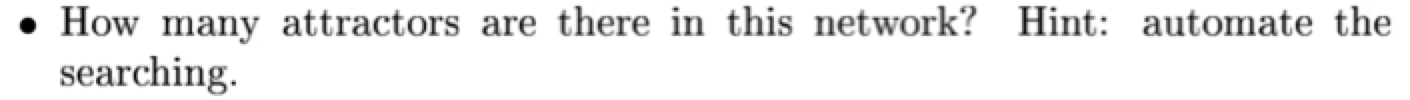
**Question1**





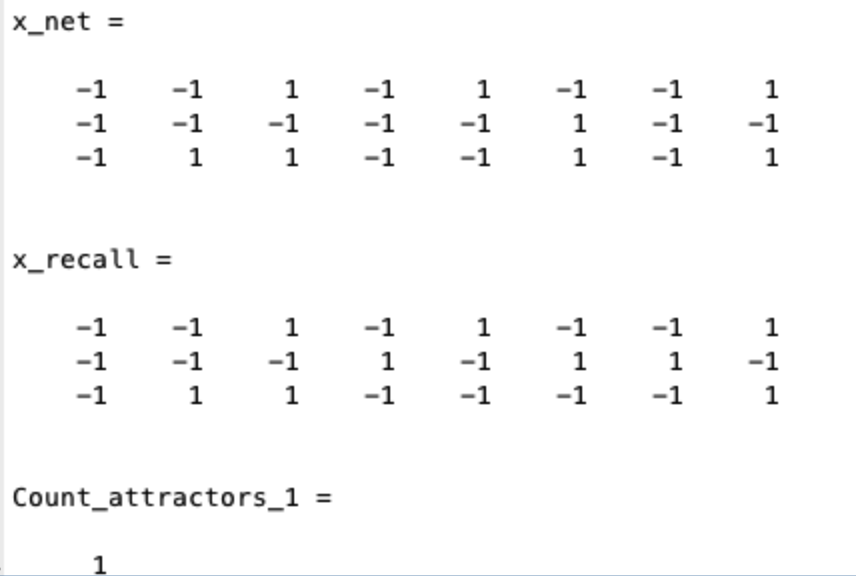
The network stores the all the patterns. All the patterns converge towards stored pattern. We find that that x1, x2 and x3 are attractors in this network.

**Question2**

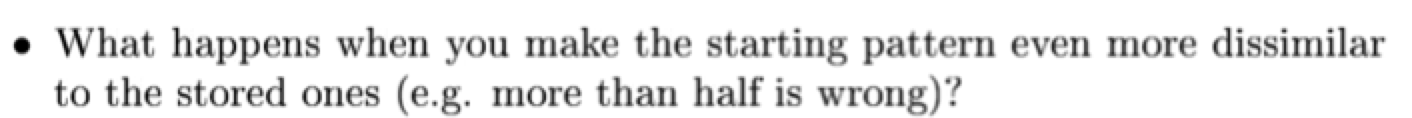


Try to recall the stored version from slightly distorted version.

We find only one attractor.

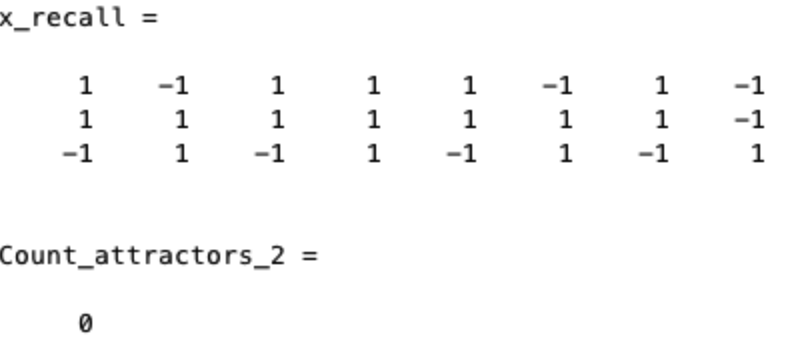


**Question3**



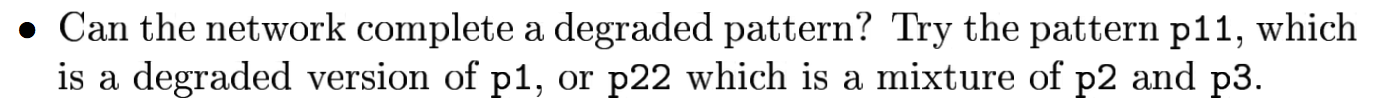
Try to recall the stored version from severely distorted version.

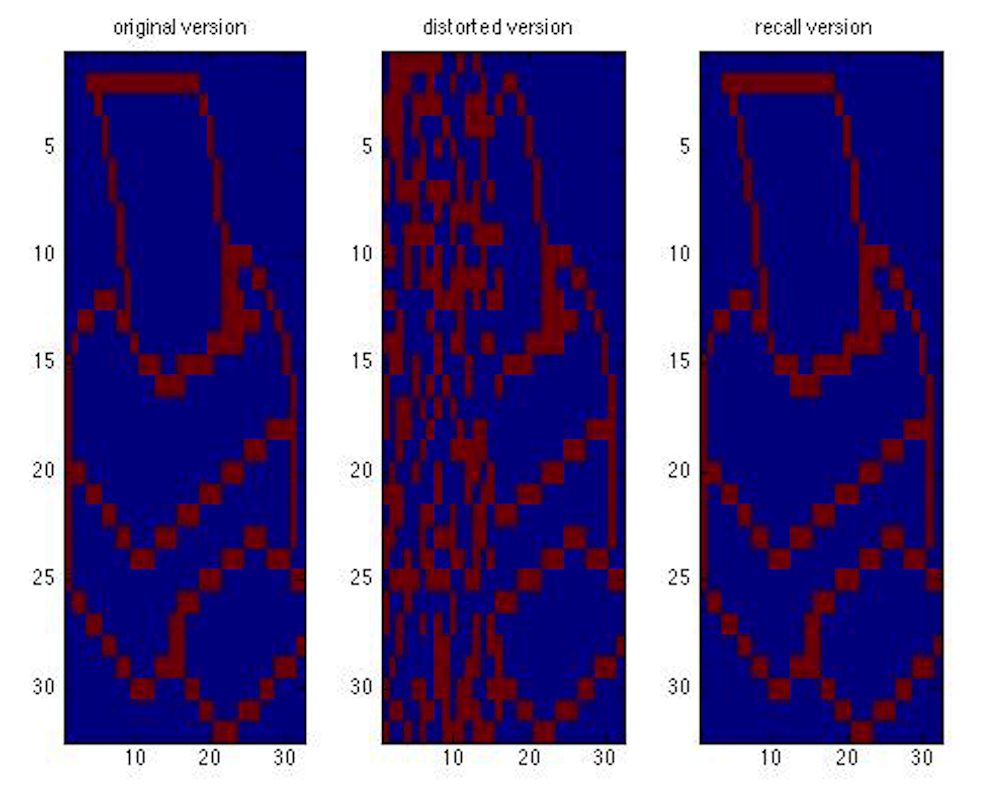
We find no one attractor.



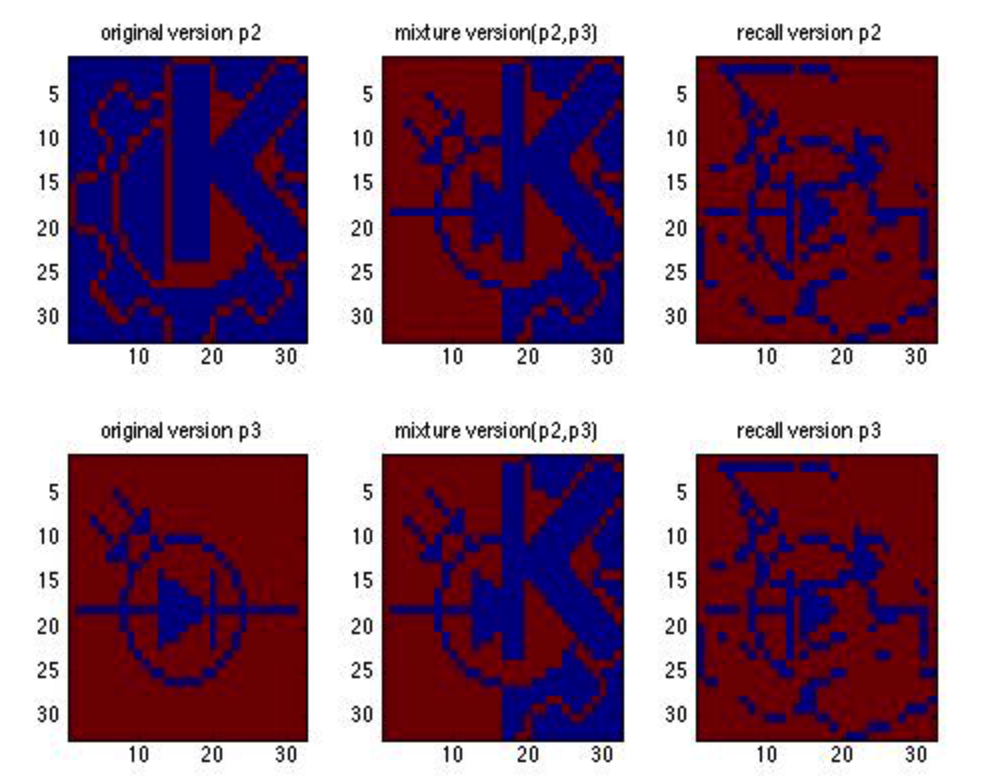


**Question1**



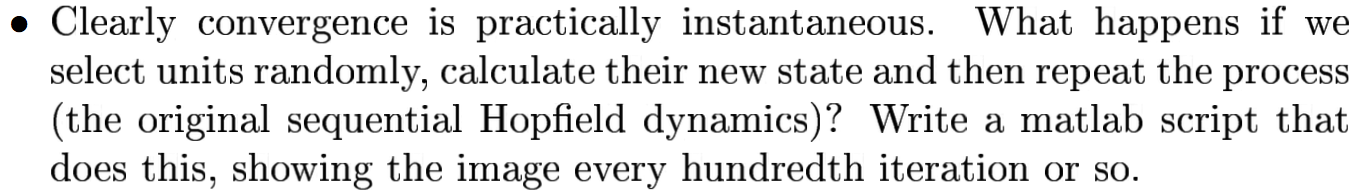


The network can complete a degraded pattern (from left to right: origin-distorted-recall degraded pattern).



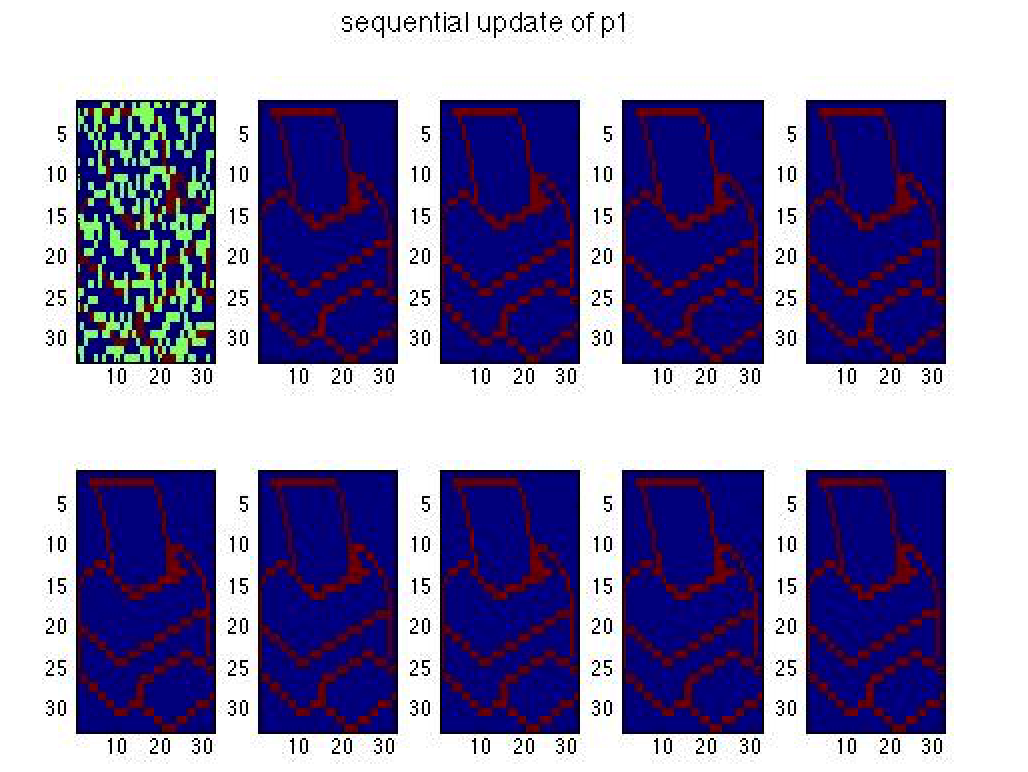
The network can complete a degraded mixture pattern.

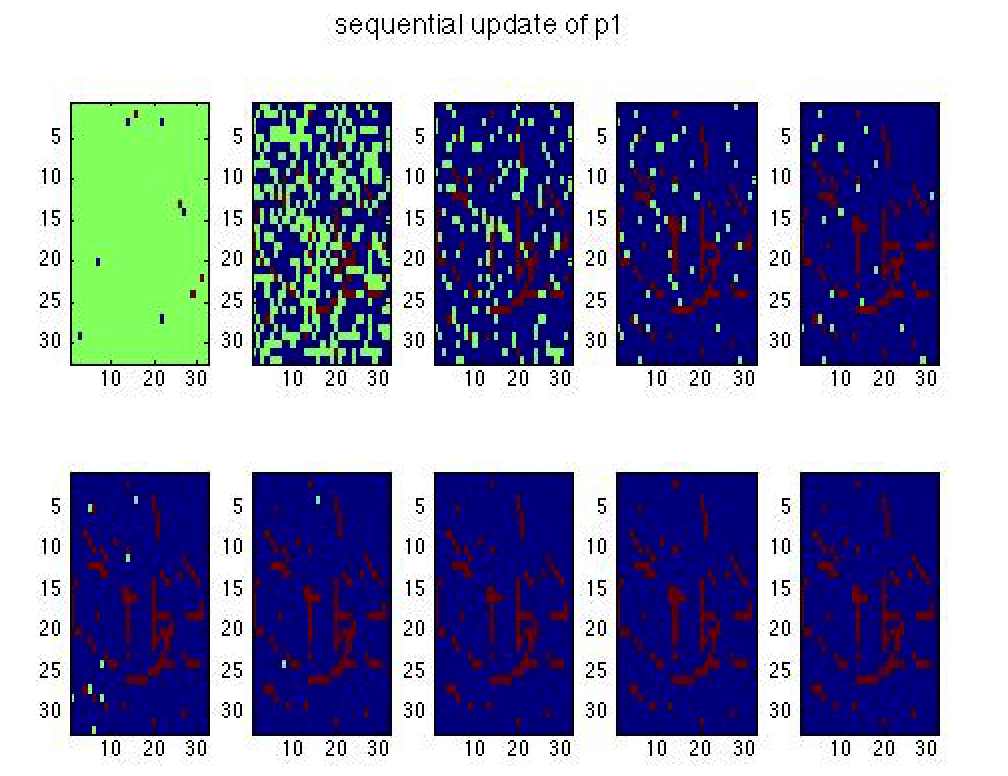
**Question2**



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

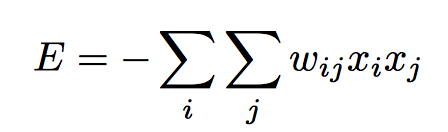
**10 units:** With 10 units, it takes more than 600 iterations to converge.

****

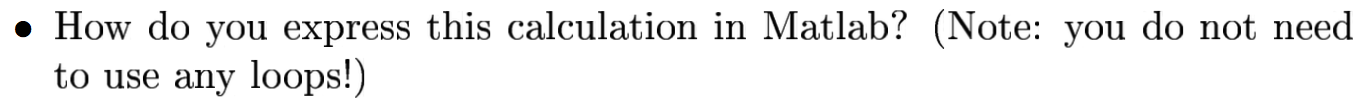
**Conclusion: With more amount of random units, the network takes less iteration to converge (converges faster) and have better performance.**



Energy function:



**Question1:**



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:**

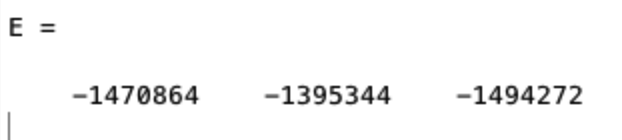


This means energy for different pattern.

**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:



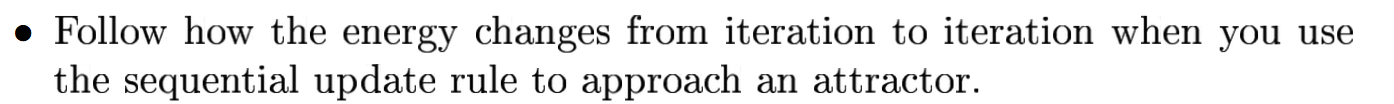
**Question3:**



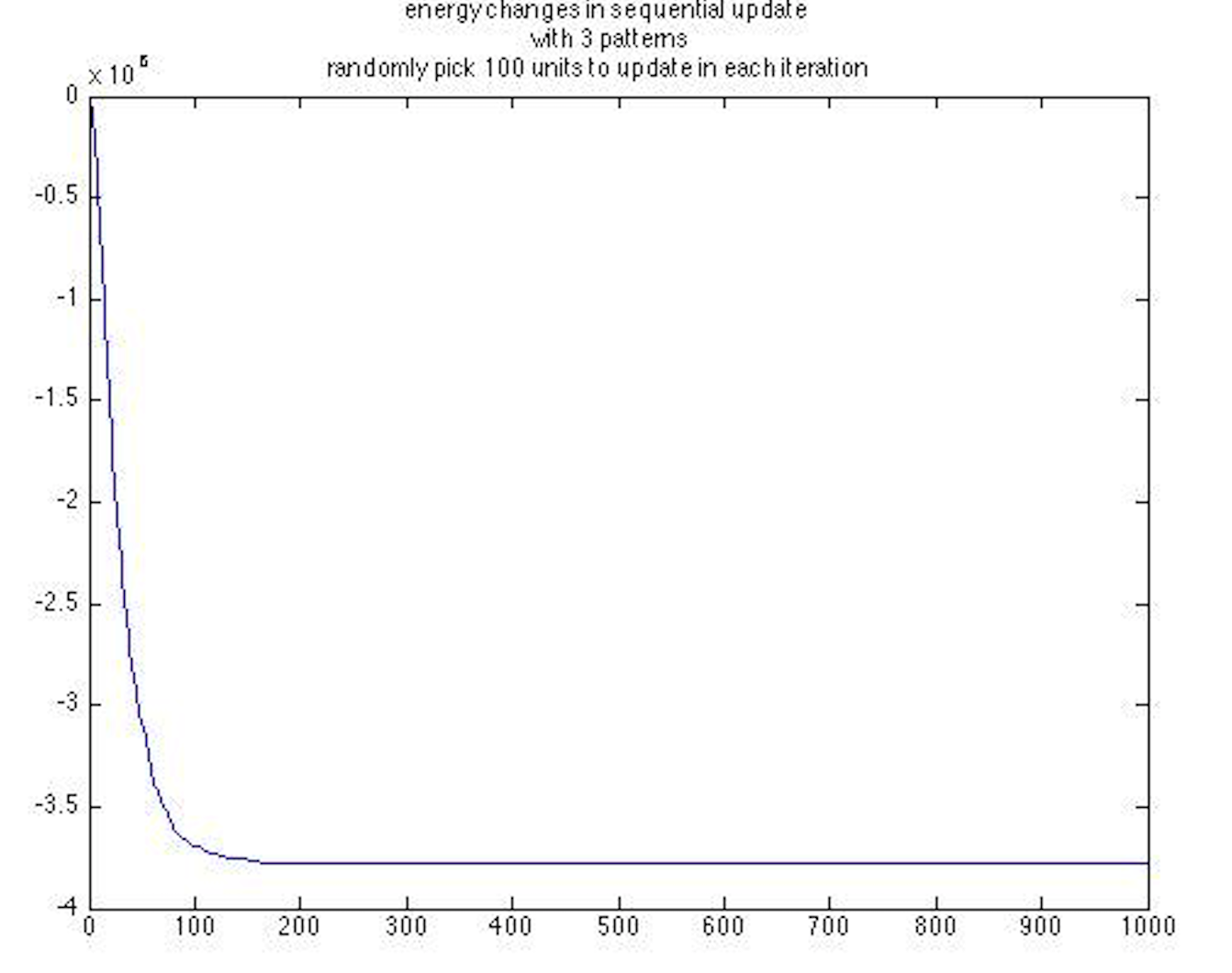
**E = -sum(sum(w.\*(p11’\* p11),1),2);**

**% p11 means the distorted pattern**

**Question4:**

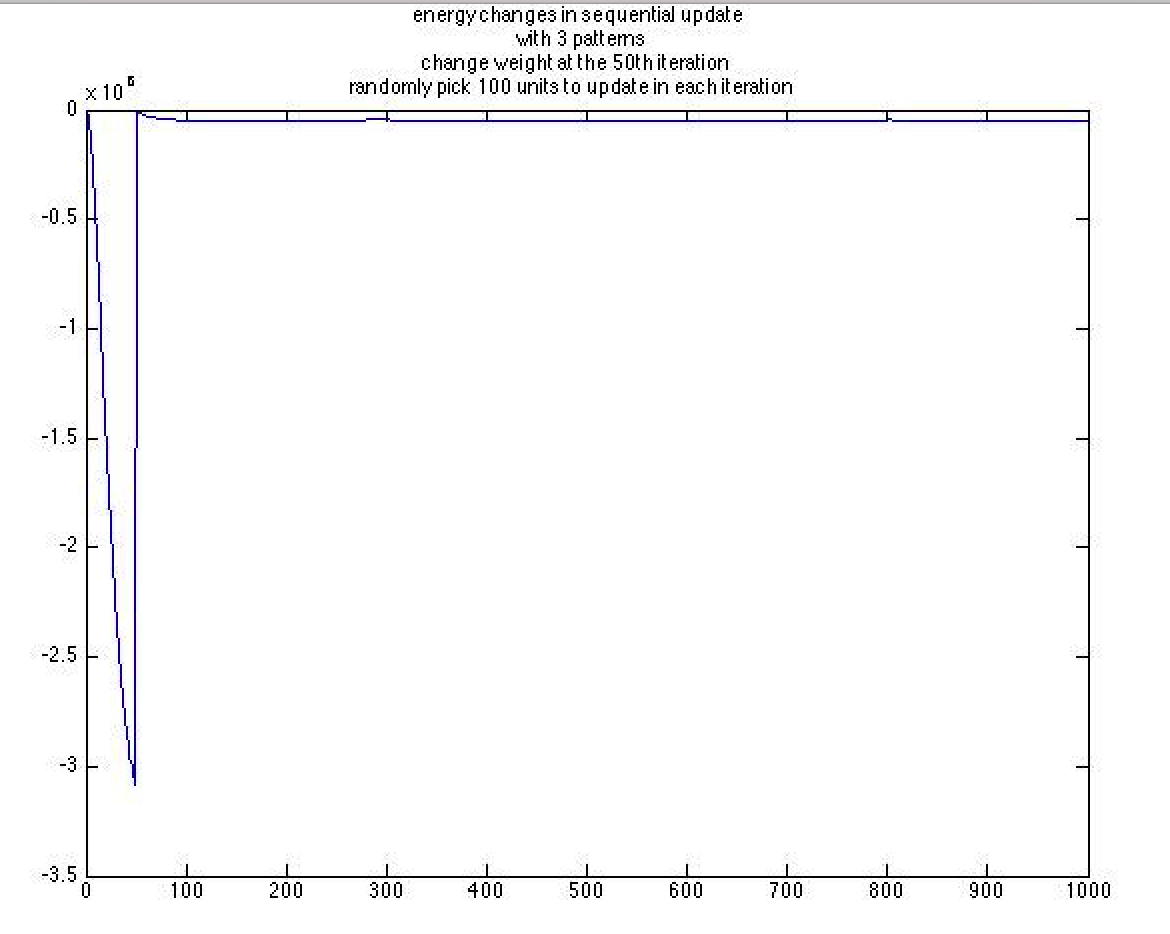


The energy decreases from iteration to iteration. It finally converges to the local minima.



**Question5:**

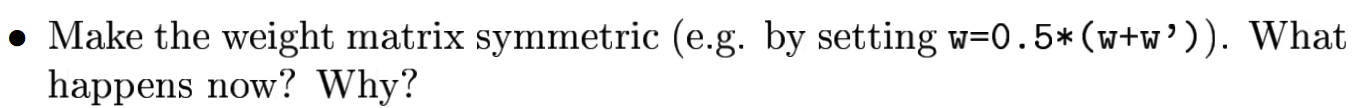


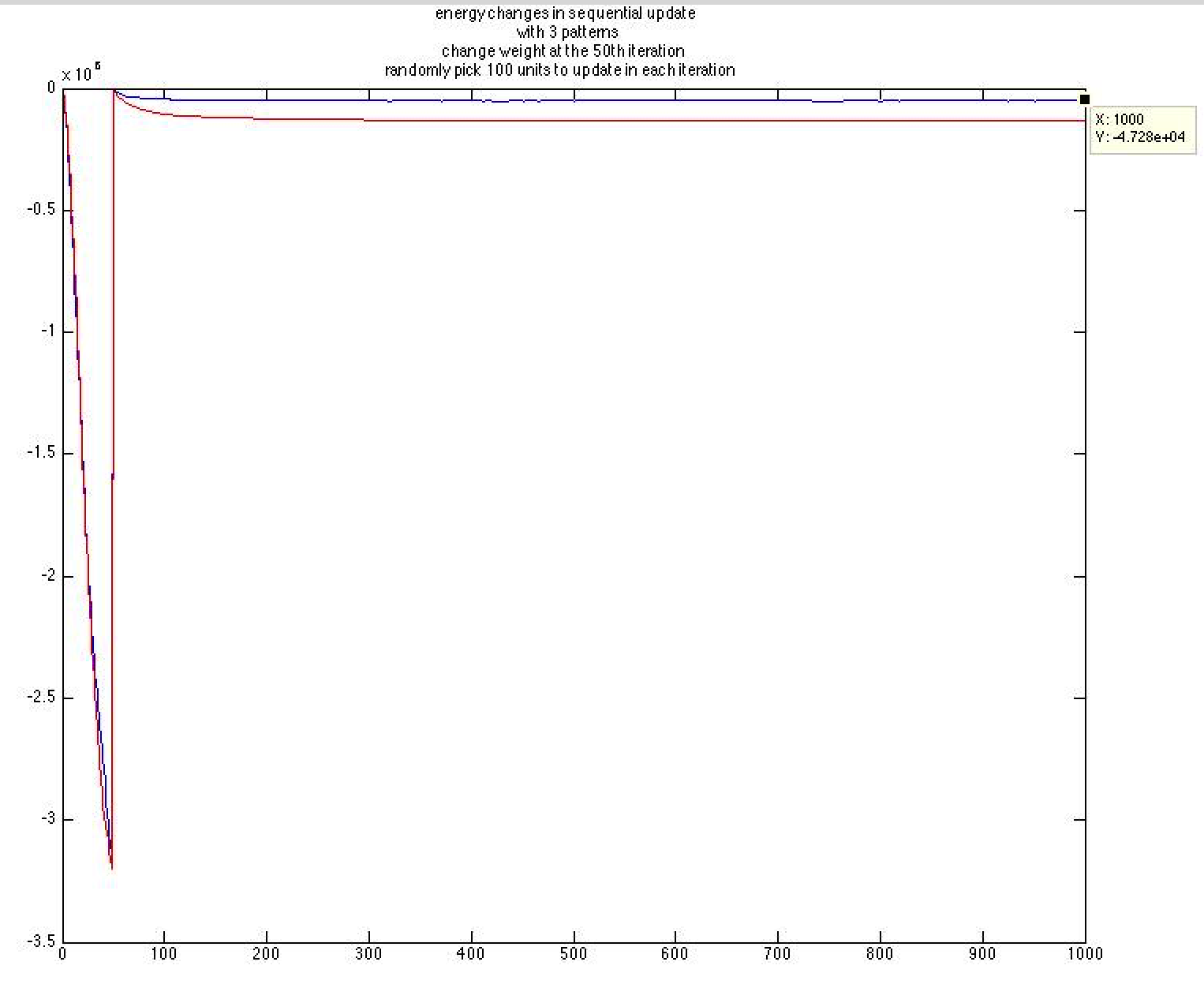


When try iterating an arbitrary starting state (at the 50th iteration), the energy starts to grow from iteration to iteration. It finally converges to the initial level.

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:**



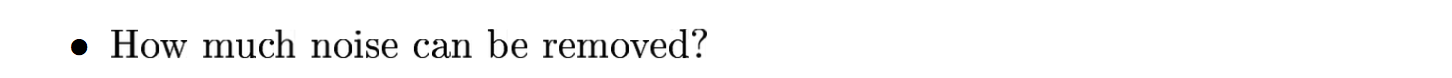


The red line is the one with symmetric weight (normally distributed random numbers).

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.





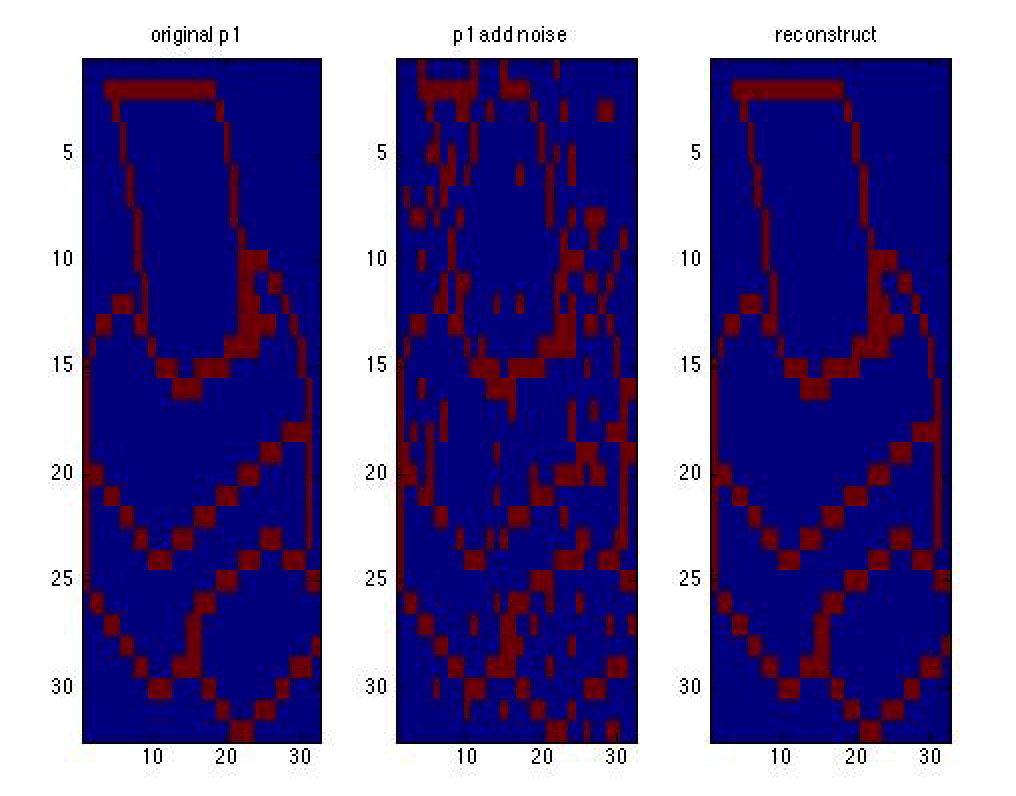
Step1: train a network with p1, p2, p3;

Step2: add noise to a pattern;

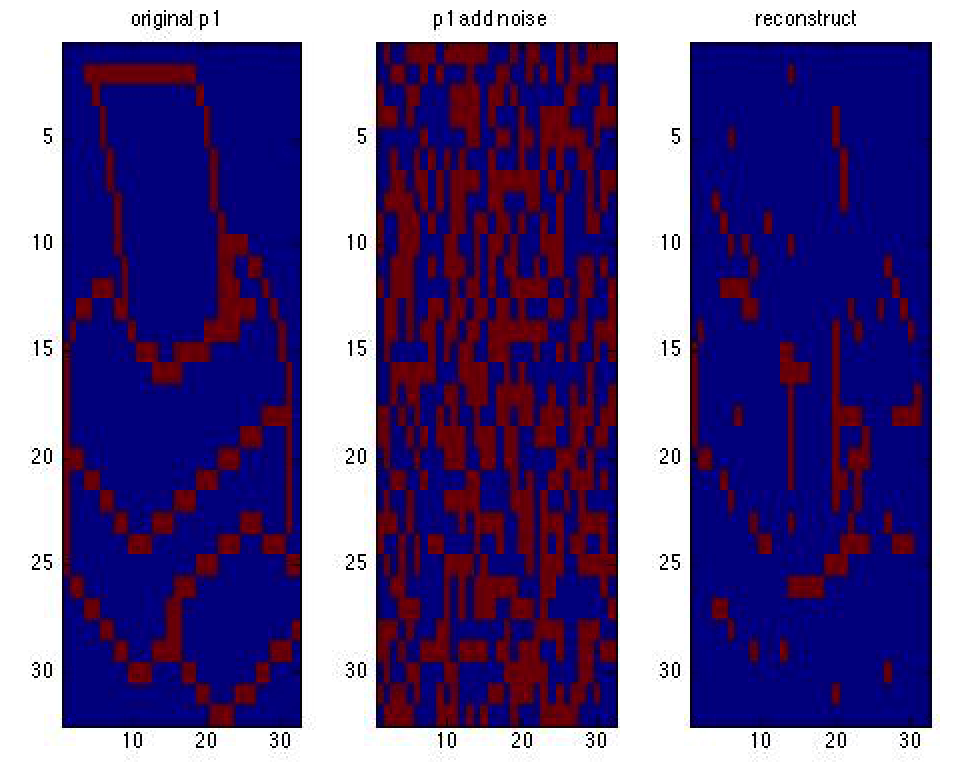
Step3: iterate it a number of times and check whether it has been successfully restored.

**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.



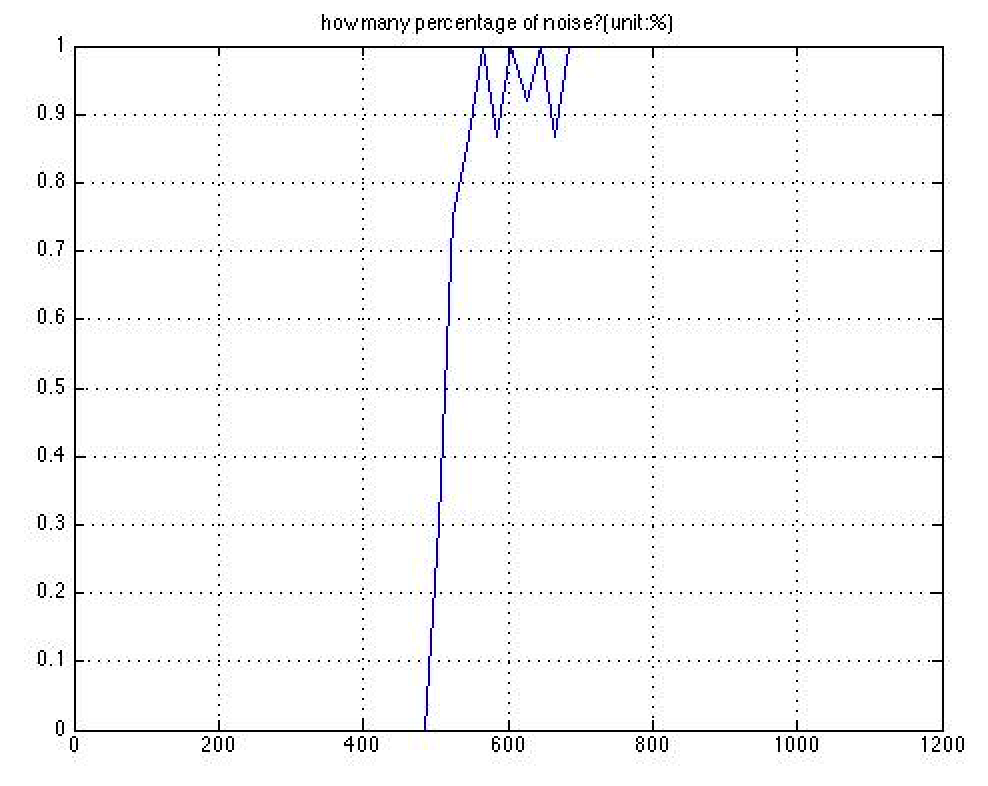
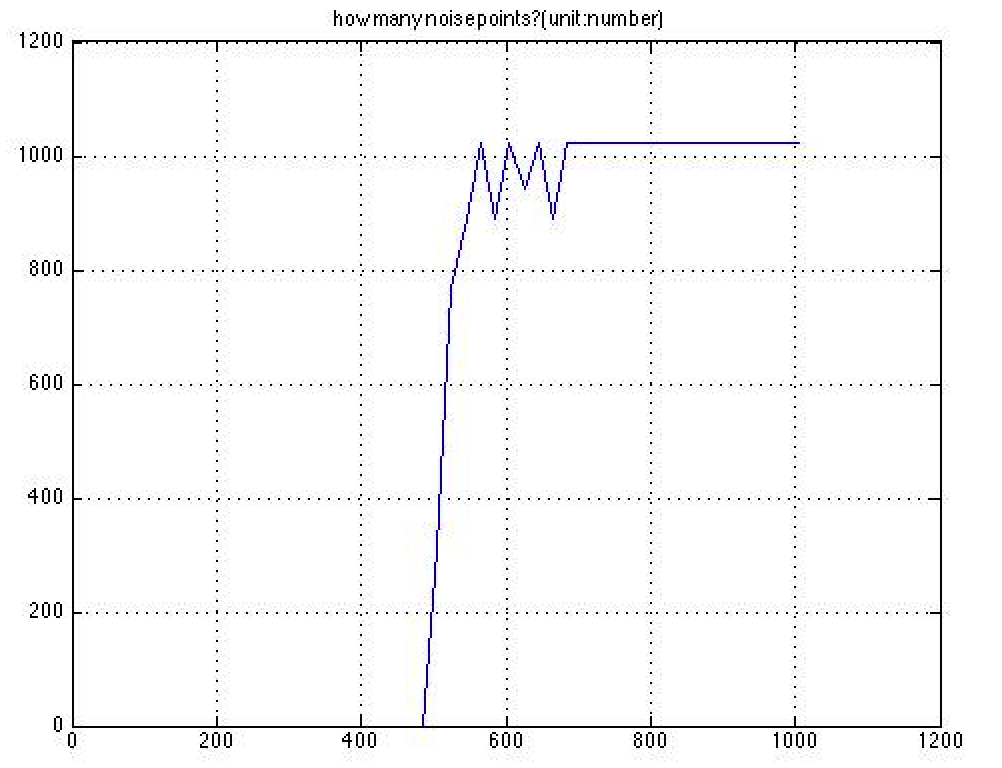
3) Let the script run across 0 to 100% noise and plot the result.

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?**



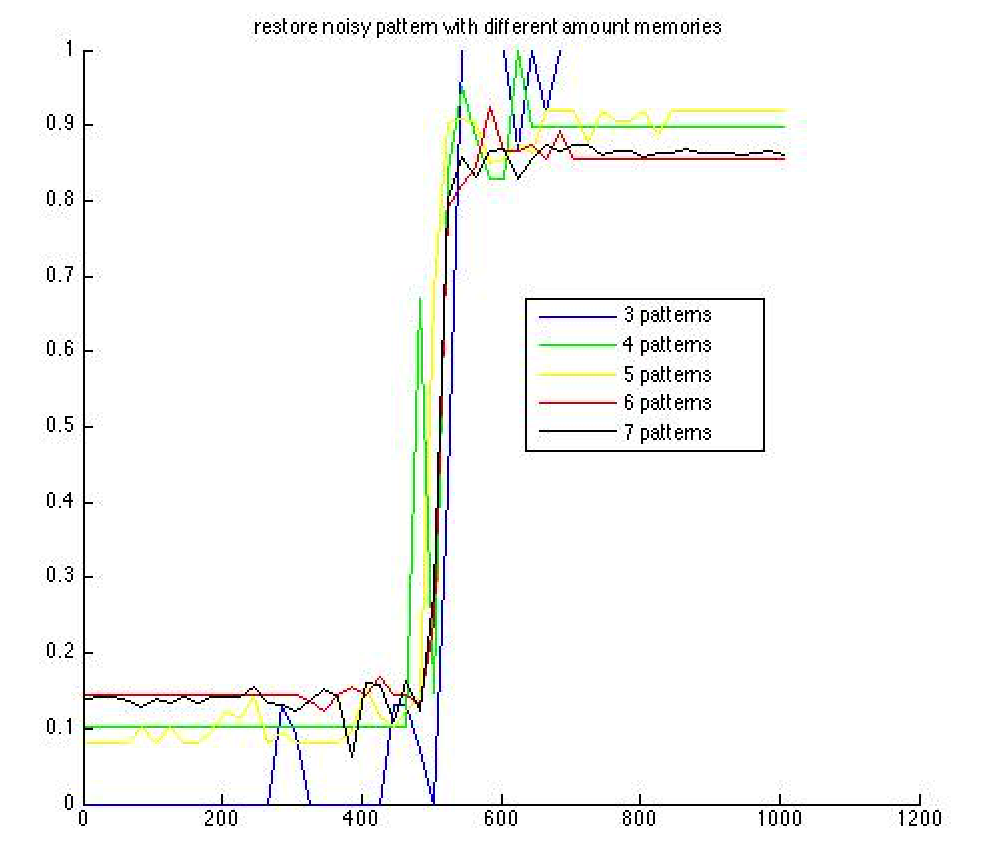


Start by adding p4 into the weight matrix and check if moderately distorted patterns can still be recognized. Then continue by adding others such as p5, p6 and p7 in some order and checking the performance after each addition.

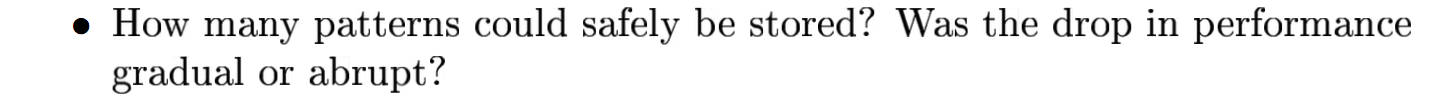
Add p4, p5, p6 and p7 to check the restoring performance:

**Result:**

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



**Question1**



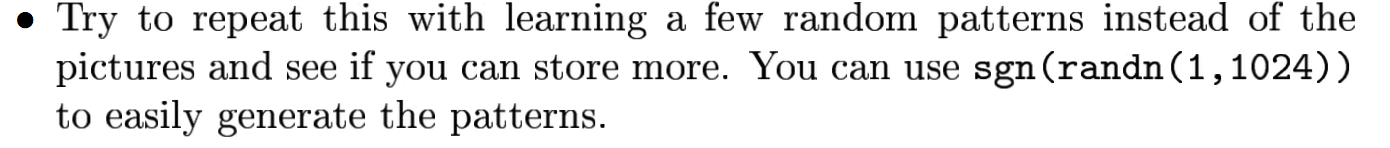
When learning 3 patterns, all 3 patterns can be safely stored. But when learning 4 patterns, none of the pattern can be safely stored. When learning 5 patterns, the performance is even worse.

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

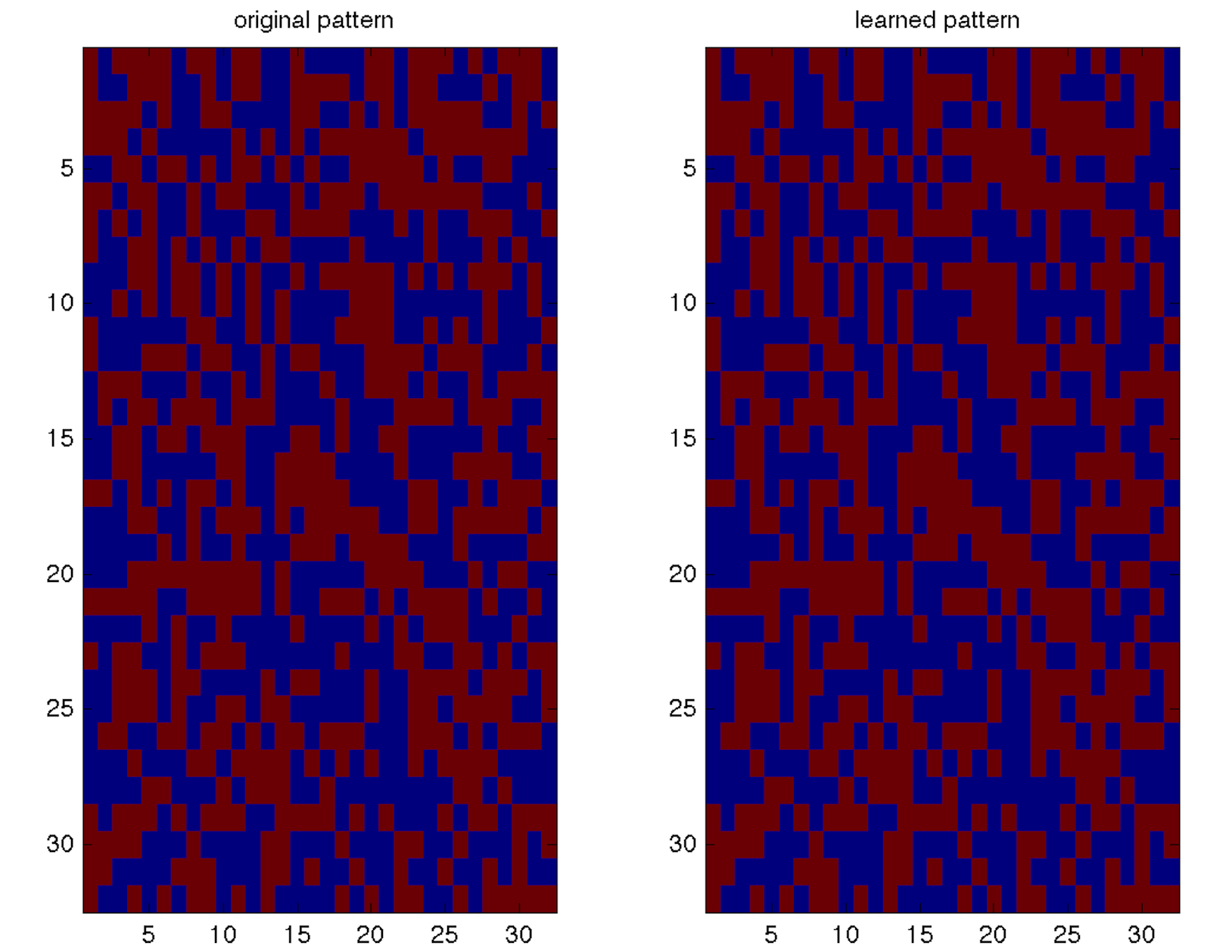
|  |
| --- |
| 3 patterns - P1 |
| 3 patterns – P2 |
| 3 patterns – P3 |

|  |
| --- |
| 4 patterns - P1 |
| 4 patterns – P2 |
| 4 patterns – P3 |
| 4 patterns – P4 |

**Question2:**

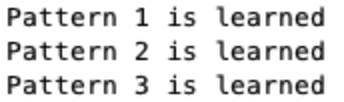


An origin pattern and its learned version: hard to compare with eyes!

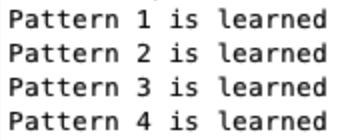


**Compare with command: isequal()**

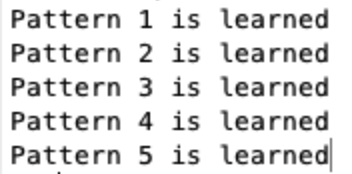
Try with 3 random patterns:



Try with 4 random patterns:



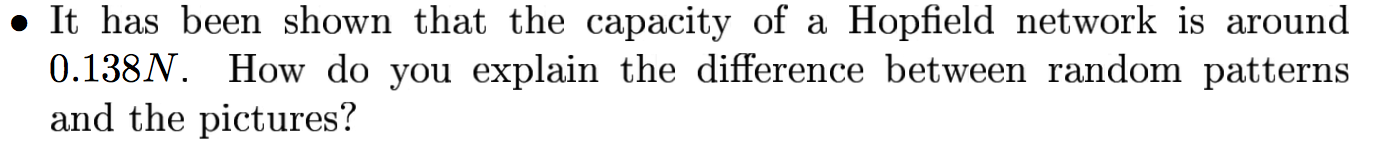
Try with 5 random patterns:



…

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:**



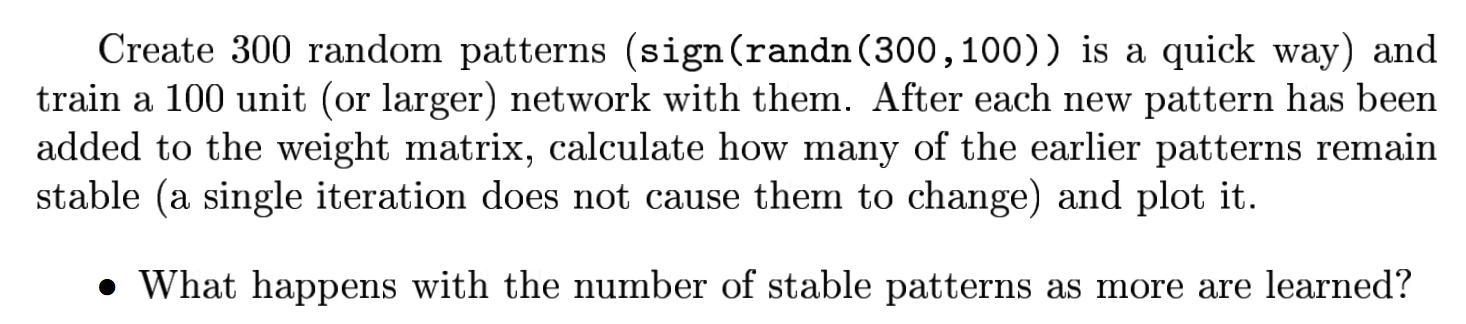
The capacity decreases when learning strong patterns, the pictures.

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**



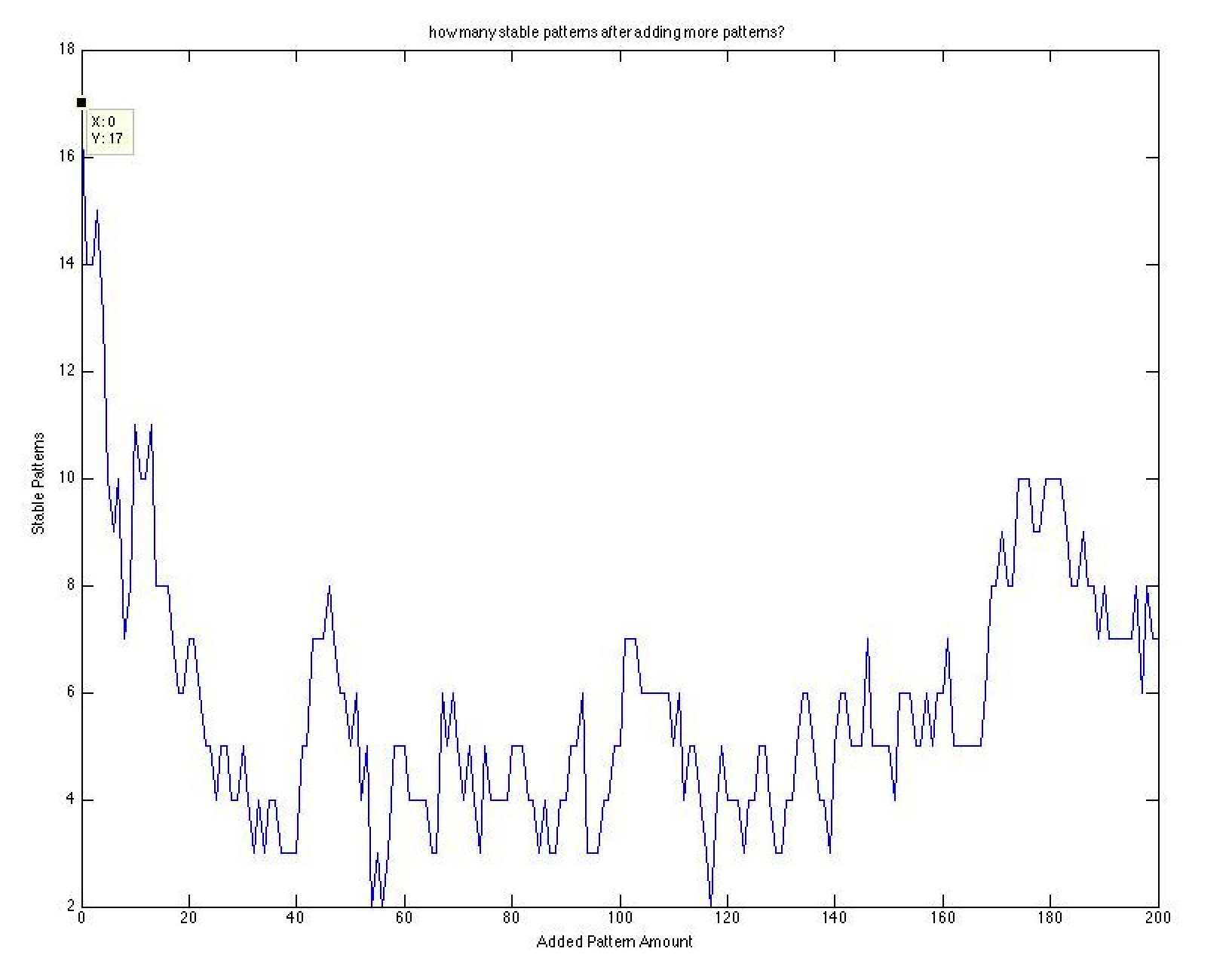
STEP 1:Create 300 random patterns

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

STEP3:

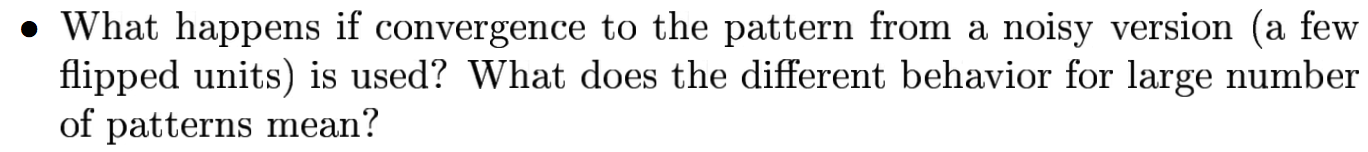
1. add new patterns to the weight matrix
2. run the network with little model for one iteration
3. calculate how many earlier patterns remain stable

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**

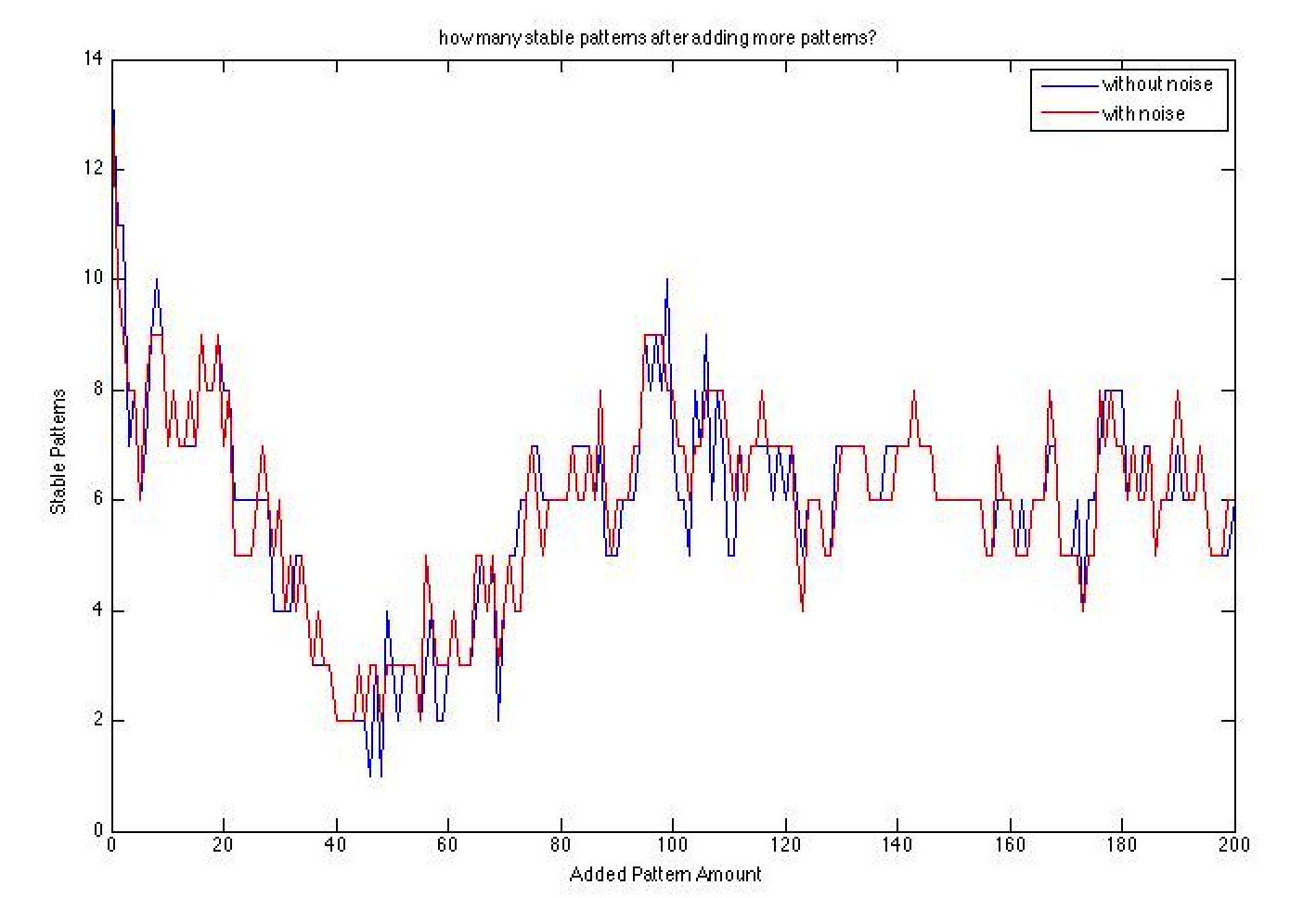


STEP 1:Create 300 random patterns

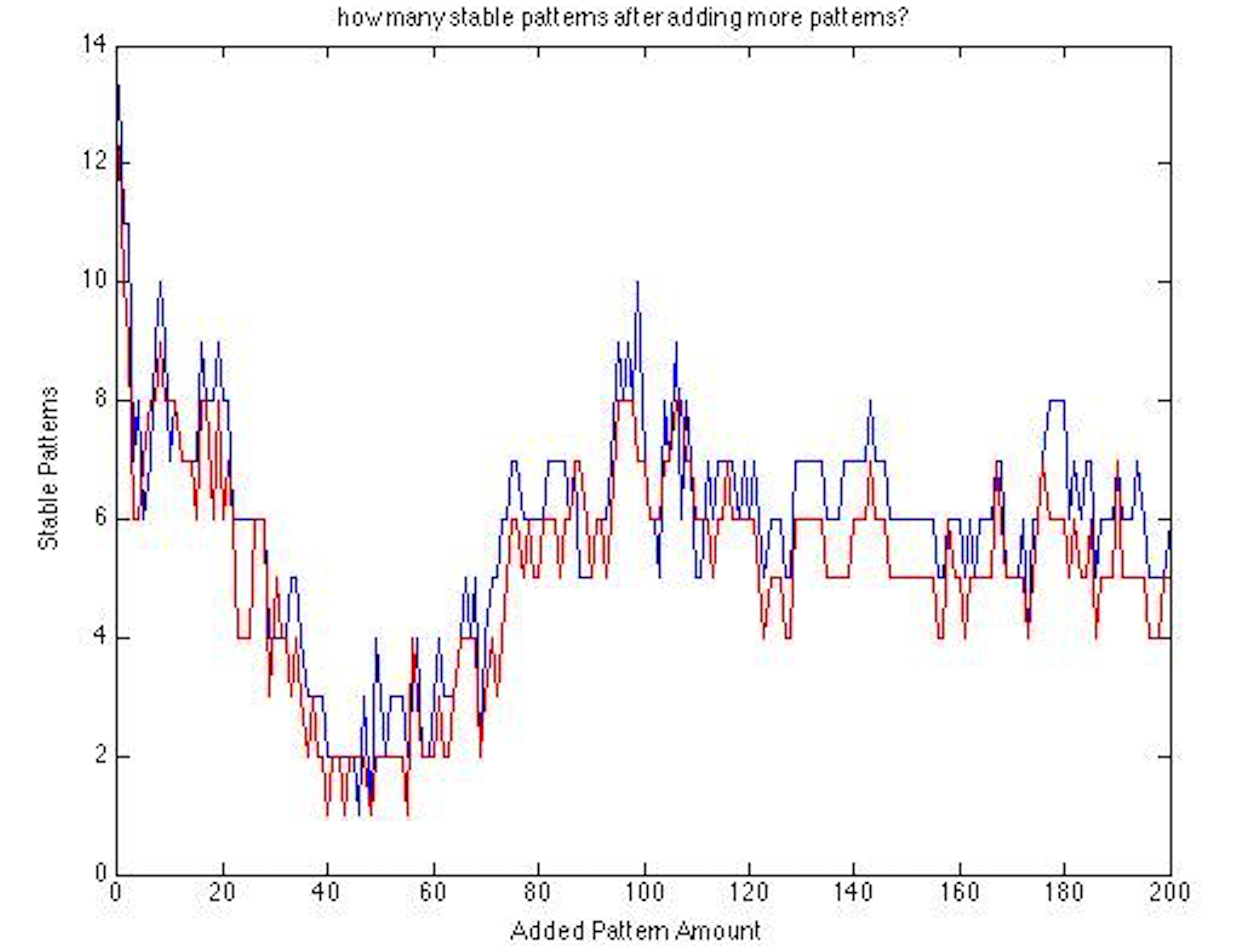
STEP 2:train a 100 unit network with different amount of random noisy patterns

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.

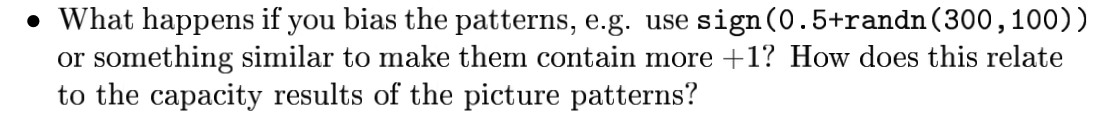
With larger amount of noise (5), the network has slightly worse performance when the number of learned pattern is large compared to network trained with no noise.

**Question6**

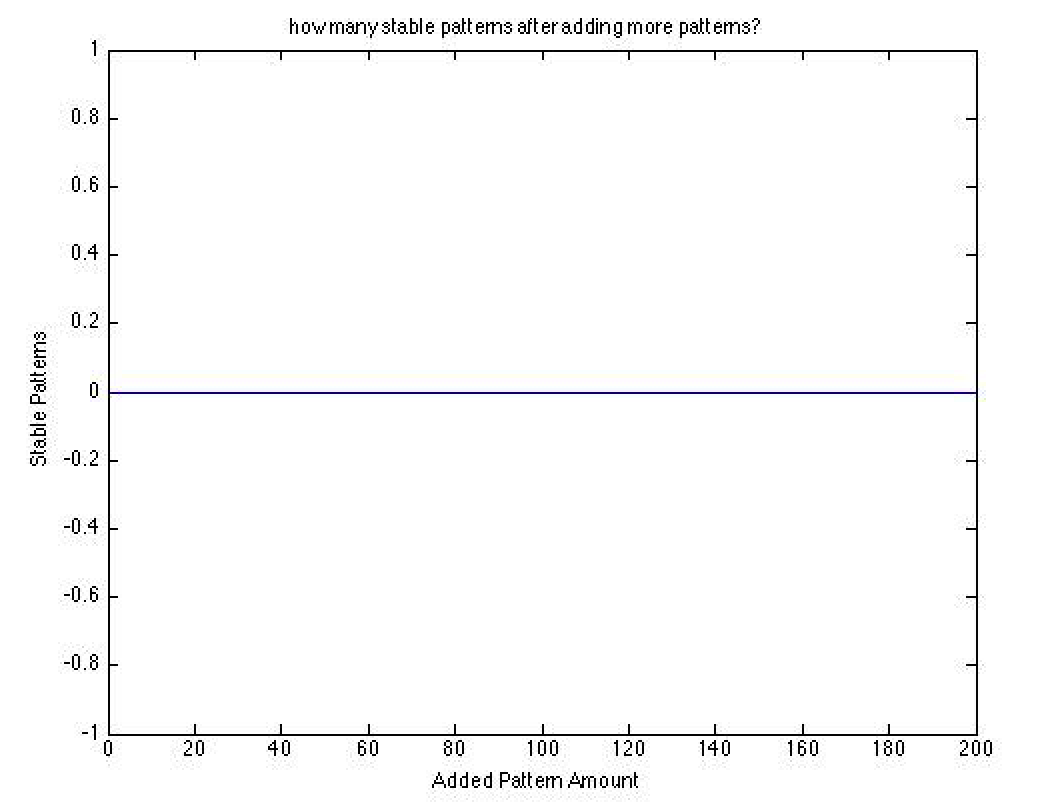


The maximum amount is 14 ( Store noisy patterns).

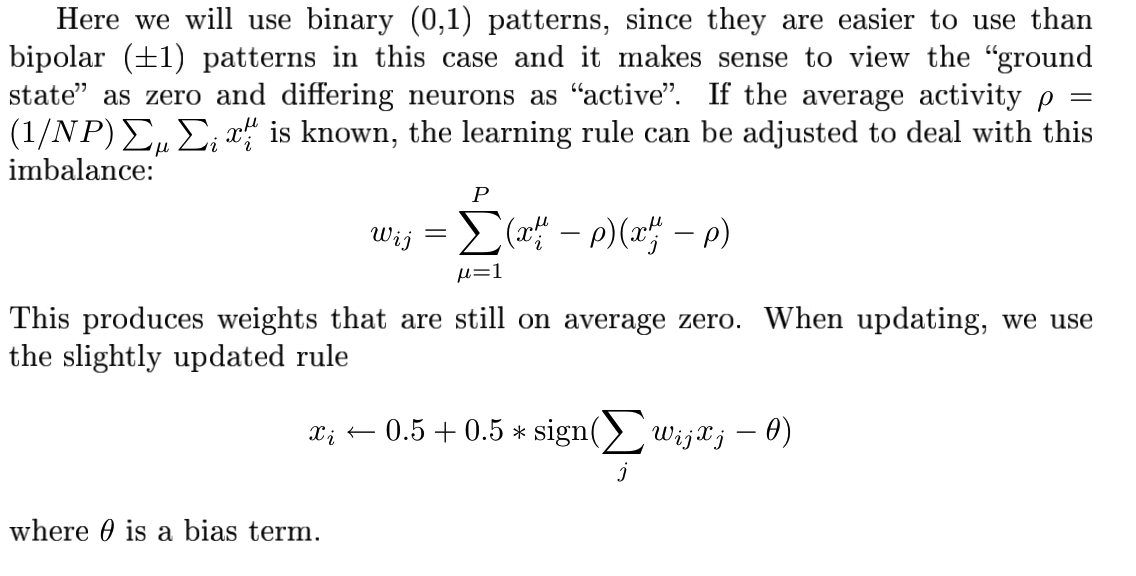
**Question7**

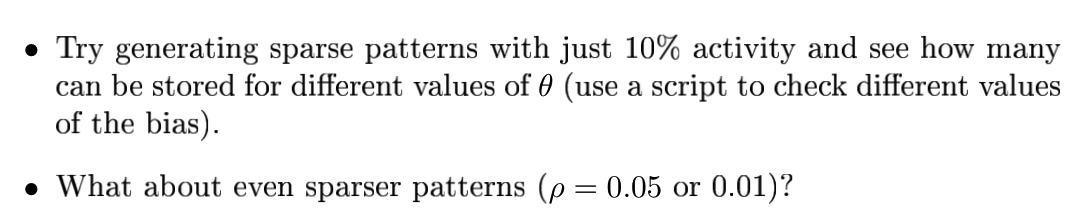


When using bias pattern, we do not get any stable patterns. the capacity of the network decreases drastically.

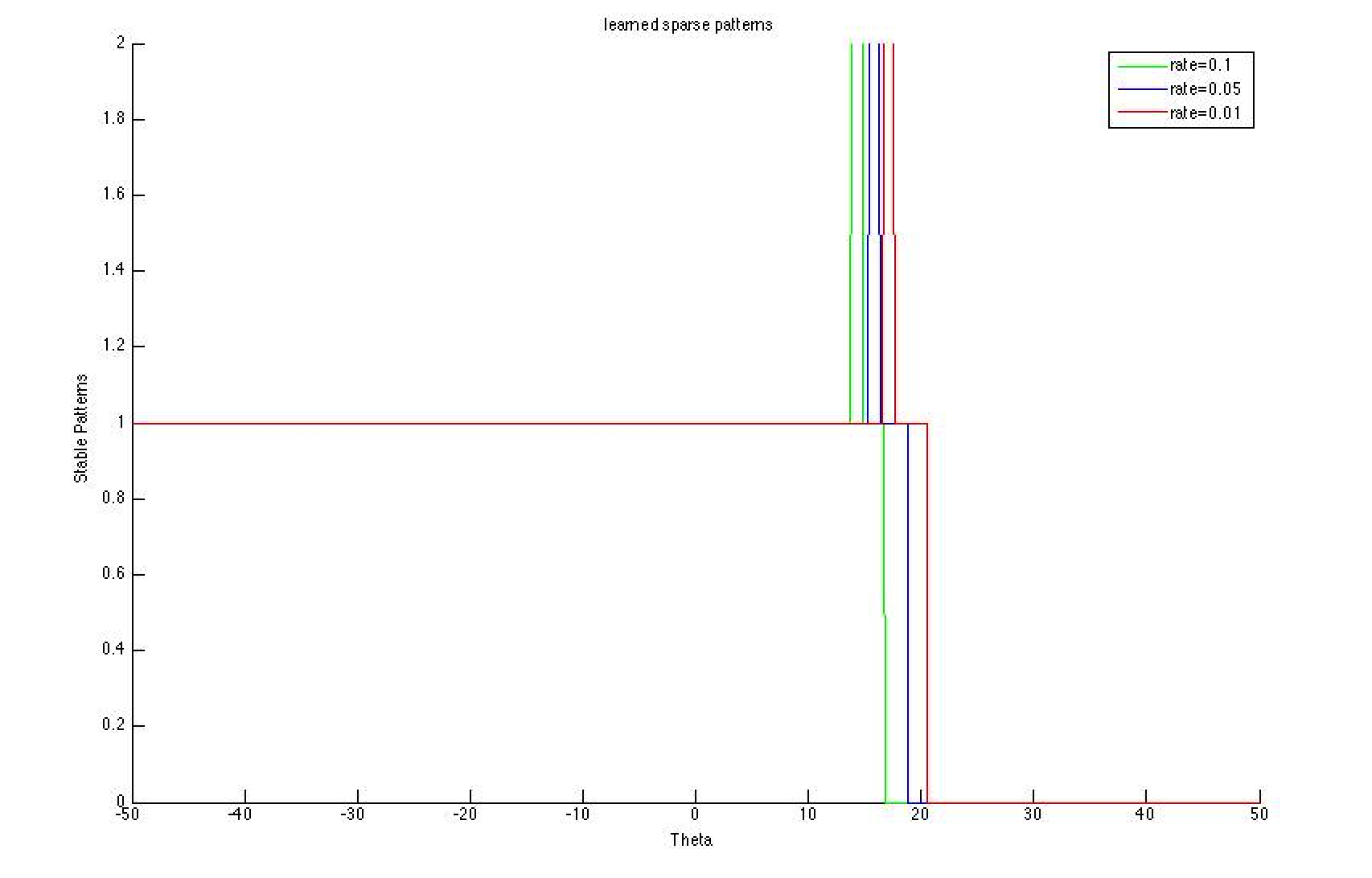






****

**Very hard to learn spare patterns**

****