

Artificial Neural Networks and Deep Architecture, DD2437

SHORT REPORT

Lab Assignment 3 : Hopfield Networks

Author :

Putra, Ramadhani Pamapta

Rahgozar, Parastu

Setiawan, Stanley

1. Aim and Objectives

After the lab completion, students are expected to be able to know :

- explain the principles underlying the operation and functionality of autoassociative networks.
- how the Hopfield network is trained.
- explain the attractor dynamics of Hopfield networks the concept of energy function
- how auto-associative networks can do pattern completion and noise reduction.
- investigate the question of storage capacity and explain features that help increase it in associative memories.

2. Scope

Most of the operations of the Hopfield-type of networks can be seen as vector-matrix operations. First, we will look at some simple networks using the Hebbian learning principle, which is often employed in recurrent networks. Students will construct an autoassociative memory of the Hopfield type, and explore its capabilities, capacity and limitations. In most of the tasks, you will be asked to study the dynamics and analyze its origins (potentially explain different behaviours you observe).

3. Tools Used

- MATLAB 2017a

4. Results/Findings

4.1 Convergence and Attractors

```

---- (1 iteration)
Pattern: -1 -1  1 -1  1 -1 -1  1
Output : -1 -1  1 -1  1 -1 -1  1
---- (1 iteration)
Pattern: -1 -1 -1 -1 -1  1 -1 -1
Output : -1 -1 -1 -1 -1  1 -1 -1
---- (1 iteration)
Pattern: -1  1  1 -1 -1  1 -1  1
Output : -1  1  1 -1 -1  1 -1  1

```

Figure 4.1.1 Output and pattern are the same, meaning that the algorithm works

```

---- (2 iterations)
Input  : 1  0  1  0  1  0  0  1
Output: 0  0  1  0  1  0  0  1 (pattern #1)
---- |(3 iterations)
Input  : 1  1  0  0  0  1  0  0
Output: 0  0  0  0  0  1  0  0 (pattern #2)

```

Figure Fig 4.1.2 – By changing more than 3 units of the pattern, the network is unable to recall the pattern

The first task of this lab was to use a *little model*. We have 3 input which are x_1 , x_2 and x_3 . What we have done for the first one is to see whether or not the network was able to store all the three patterns. We have learned that yes, it was able to store all three patterns when we recall the data by multiplying weight matrix (W) with the input. The W in this lab can be seen in Figure 4.1. The values, however are not put into the Sign function, hence the number is not -1 or 1. But, we can already see that the matrix have the same diagonal values, and the matrix is like a mirror matrix.

In addition, when we multiply this matrix with the input, it will result the correct result. This can be seen in Figure 4.2. Hence all patterns converge towards stored patterns.

Next, we try to find the attractors in this network. There are 14 attractors in this network.

We then tried to use x_{1d} , x_{2d} and x_{3d} as an input pattern to be tested with the W that we have trained. I all showed that when there are 1 – 3 error bits, the network is able to make a correct result. However, when we tried using 4 errors, the network failed. Thus, we conclude that this network is only able to withstand only a maximum of 3 error in the input patterns.

4.2 Sequential Update

In this part of the lab, we try to load a 1024 unit network instead of a simple 8-neuron network.

First, we start by running the first three patterns P1, P2 and P3, plus two the distorted patterns which are P11 (Half of data randomly distorted) and P22 (Mixture of two patterns P1 and P2).

Letting the network reconstruct the pictures, it can be seen that, since P11 was half from p1 and half distorted, network can recall the pattern and therefore in 2 iterations, it can be reconstructed.

On the other hand, for P22, a mixture of two patterns is not a train attractor for the network, so the output keeps distorting after a few iterations.

In the third task, we are asked to choose random units to destroy the first patterns and then dynamically watch as the pattern converges to the initial state, some of the iterations can be seen in table 1

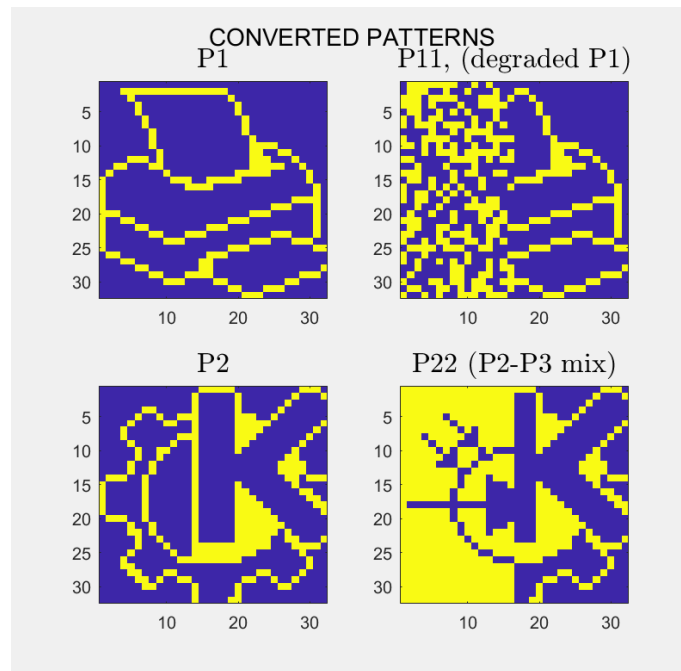


Fig 4.2.1 – Running the first two patterns, and the two last patterns.

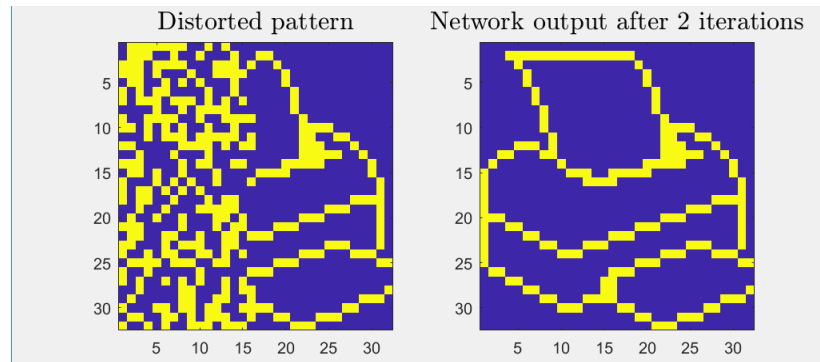


Fig 4.2.2 – After two iterations the distorted pattern could converge to the actual input

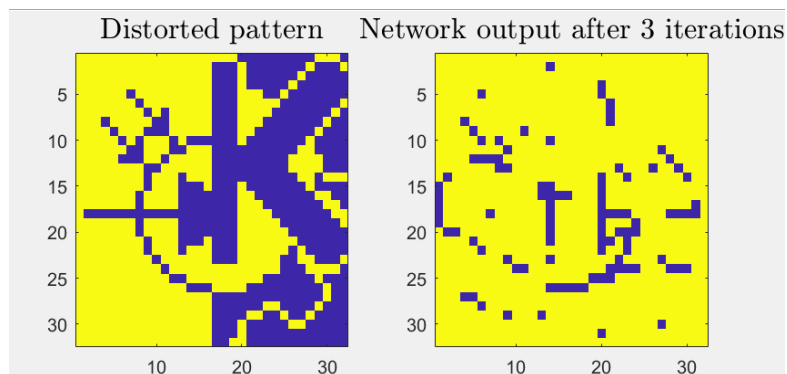
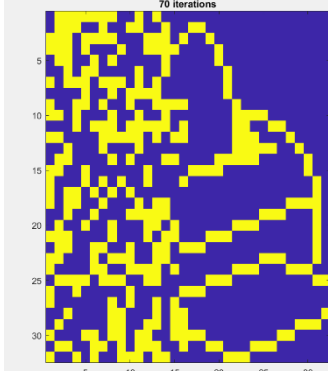
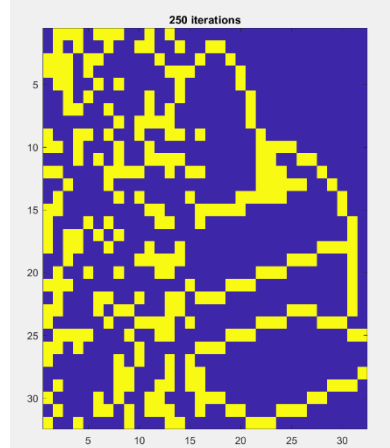


Fig 4.2.3 – Since P22 was a mixture of two patterns, it was not reconstructed by network

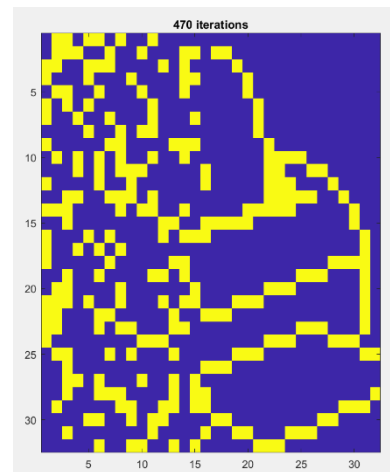
Table 1 – Dynamic convergence of P1 after random distortion

n-th iteration	Picture/image patterns
70th	

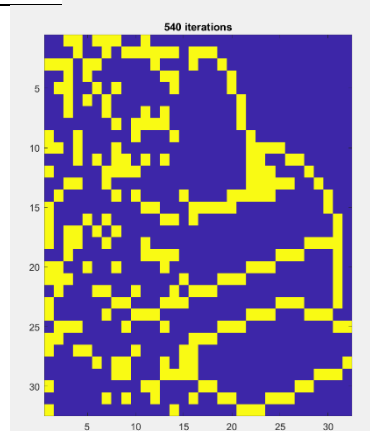
250th



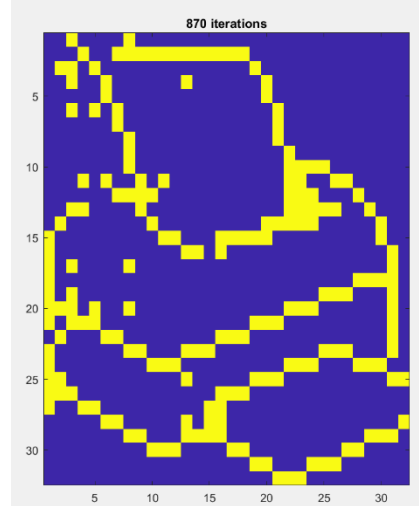
310th



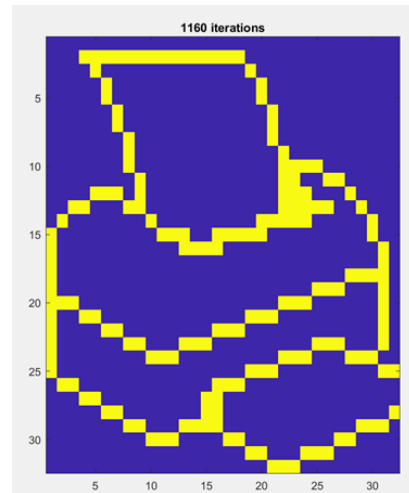
540th



870th



1160th



4.3 Energy

The energy from P1 and P2 as the attractors are:

$$P1 = -1439.390625$$

$$P2 = -1365.640625$$

Whereas the energy from P10 and P11 as the distorted patterns are smaller, the values are:

$$P10 = -415.980469$$

$$P11 = -173.5$$

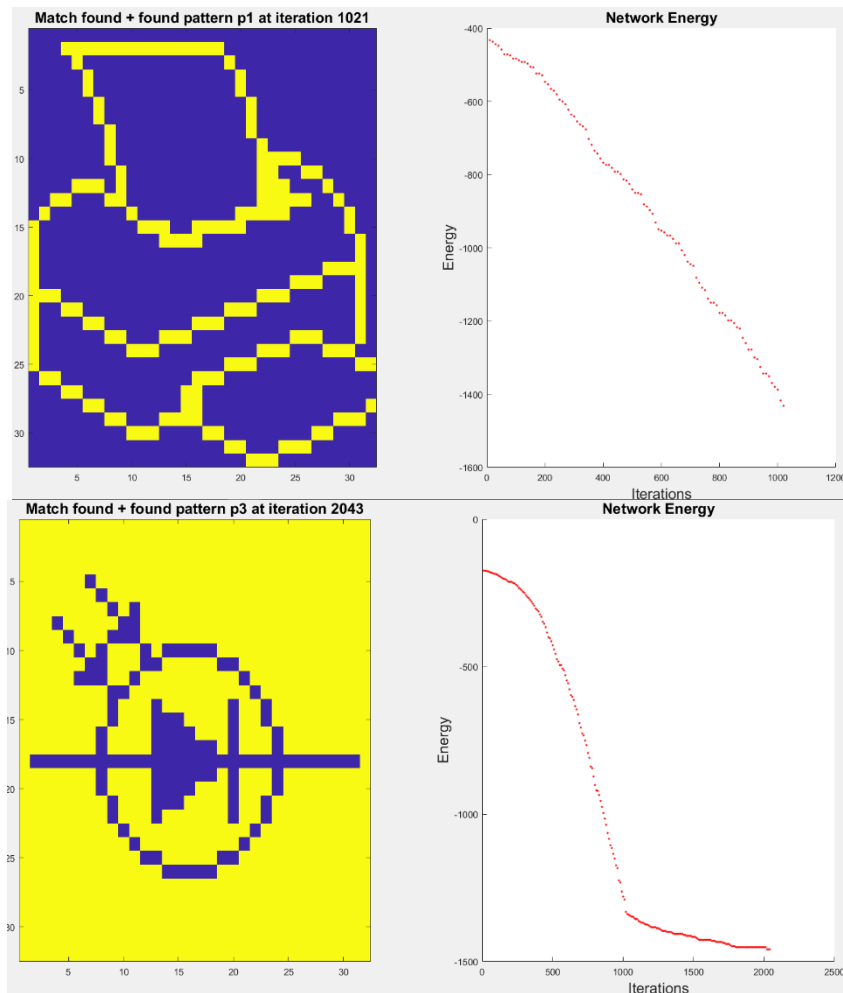


Figure 4.3.1 Energy Changes during the iterations. Top shows P10, bottom shows P11

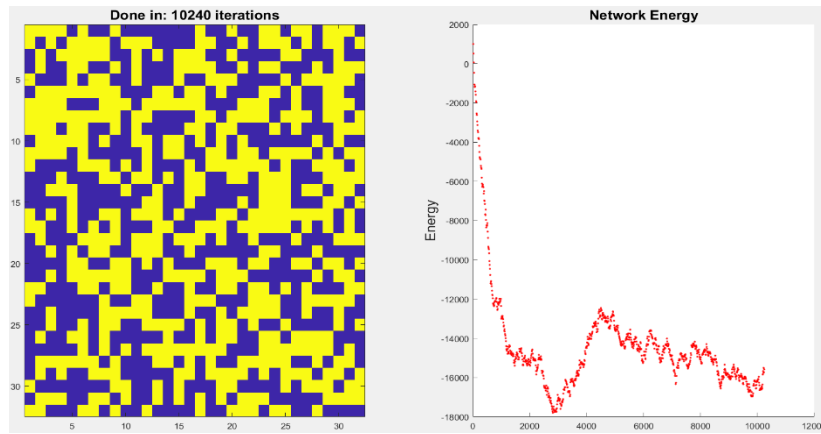


Figure 4.3. 3 Network Energy when we randomize the W

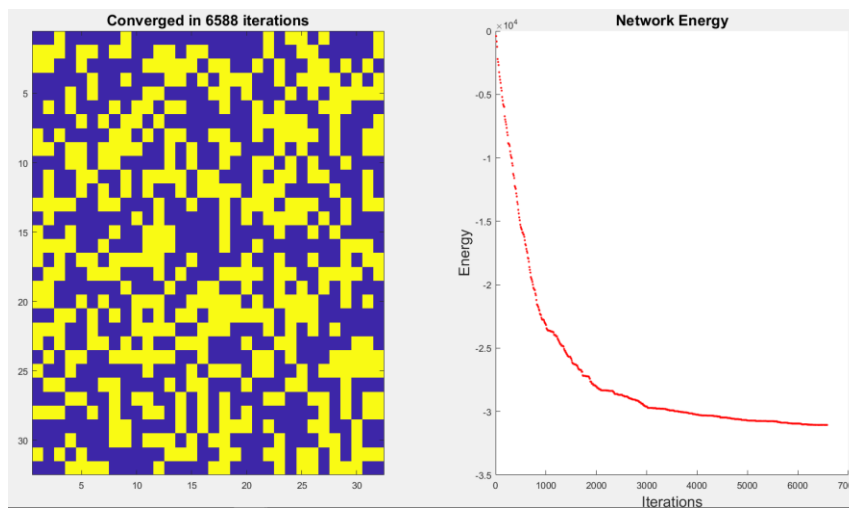


Figure 4.3.2 By using p_1 as the input and using a symmetric matrix, we can see that the energy goes down until the maximum number of iterations

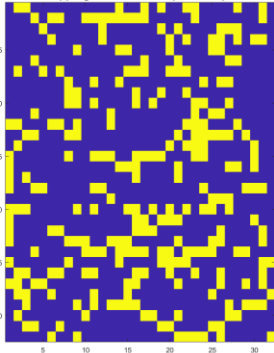
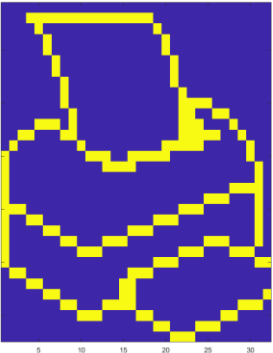
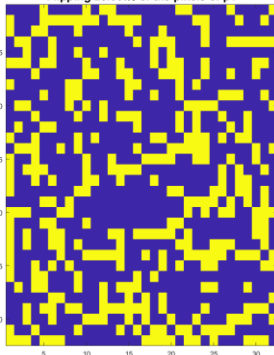
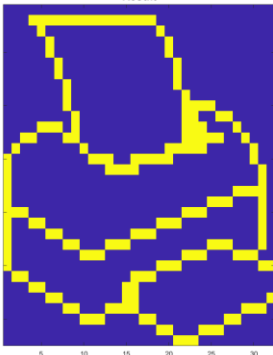
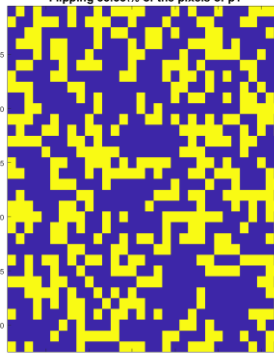
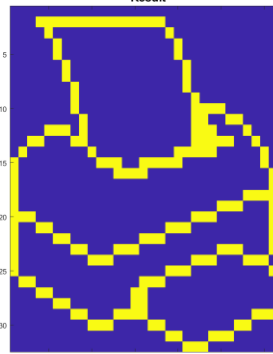
The energy decreases in each iteration when we try to approach the attractors. From p_{10} and p_{11} we can see that convergence are slower in p_2 , hence the energy changes are slower for this case. This is due to the fact that p_{11} is a combination of p_2 and p_3 . Although it mainly succeeded, sometimes p_{11} failed to show the matched picture. We could say that the energy are smaller used for a “simpler” data such as p_{10} .

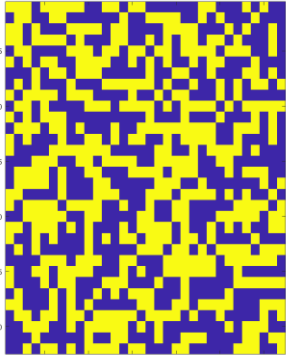
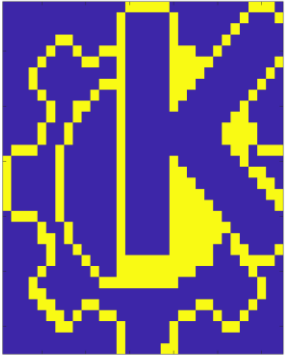
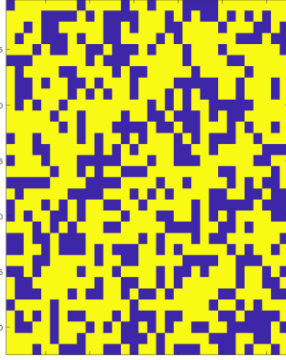
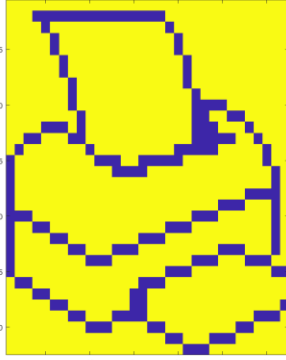
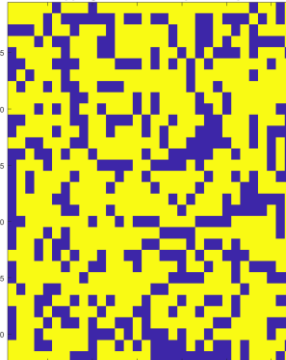
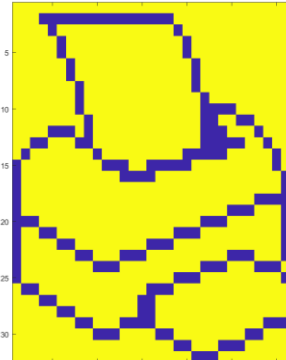
When we try to randomize the W values and used the first pattern, we can observe that the network energy have fluctuations and it tries to converge to the attractor’s energy value as well. This is due to the non-symmetric property of the matrix. This can be seen at Figure 4.3.2

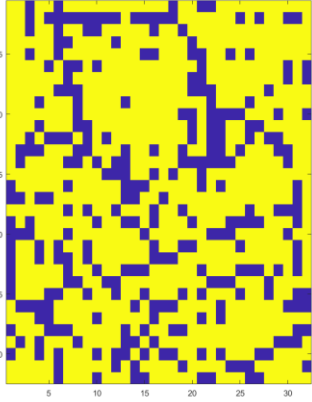
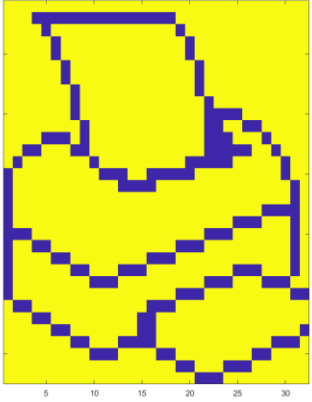
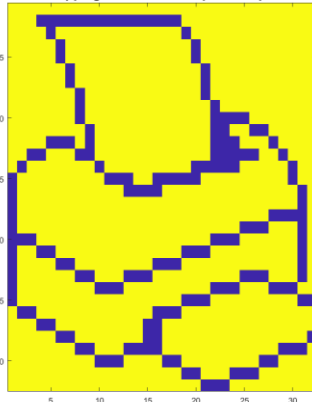
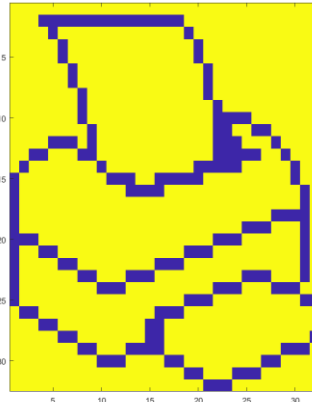
Now, when we use a symmetric matrix, the output goes down until it reaches the maximum number of iterations. When we tried using more number of iterations, it also goes all the way until the maximum value

4.4 Distortion Resistance

Table 2 Pattern recognition change by random flipping patterns

Flipping percentage	Image pattern comparison (left = image with flipped pattern, right = recognized pattern by network)
16.67%	<div> <div>Flipping 16.67% of the pixels of p1</div>  <div>Result</div>  </div>
25 %	<div> <div>Flipping 25.00% of the pixels of p1</div>  <div>Result</div>  </div>
33.33 %	<div> <div>Flipping 33.33% of the pixels of p1</div>  <div>Result</div>  </div>

<p>50 %</p>	<div> <div>Flipping 50.00% of the pixels of p1</div>  <div>Result</div>  </div>
<p>66.67 %</p>	<div> <div>Flipping 66.67% of the pixels of p1</div>  <div>Result</div>  </div>
<p>75 %</p>	<div> <div>Flipping 75.00% of the pixels of p1</div>  <div>Result</div>  </div>

83.33 %	<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;"> <p>Flipping 83.33% of the pixels of p1</p>  </div> <div style="text-align: center;"> <p>Result</p>  </div> </div>
100 %	<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;"> <p>Flipping 100.00% of the pixels of p1</p>  </div> <div style="text-align: center;"> <p>Result</p>  </div> </div>

The network tried to classify the flipped pattern into the memorized pattern. The flipped pattern is processed and readjusted (by removing the noise) until the processed pattern is becoming as similar as one of the stored/trained pattern. From Table 4, we can see that with lower percentages in flipped pattern, the network could classify the pattern correctly(p1). However, in case of 50 % flipped patterns, the resulting image was too difficult for the network to be recognized, and the flipped pattern ended up incorrectly associated (the pattern resembles p2 instead of the correct p1). When the flipped pattern reaches 66.67 % and higher, the network recall the flipped p1 pattern into the inverted/negative version of p1 pattern. This is due to when a dominant majority of a pattern is flipped, then it will resemble the pattern of its negative/inverted version. Since the attractors are placed at the same site, the network will correctly recall the flipped pattern.

4.5 Capacity

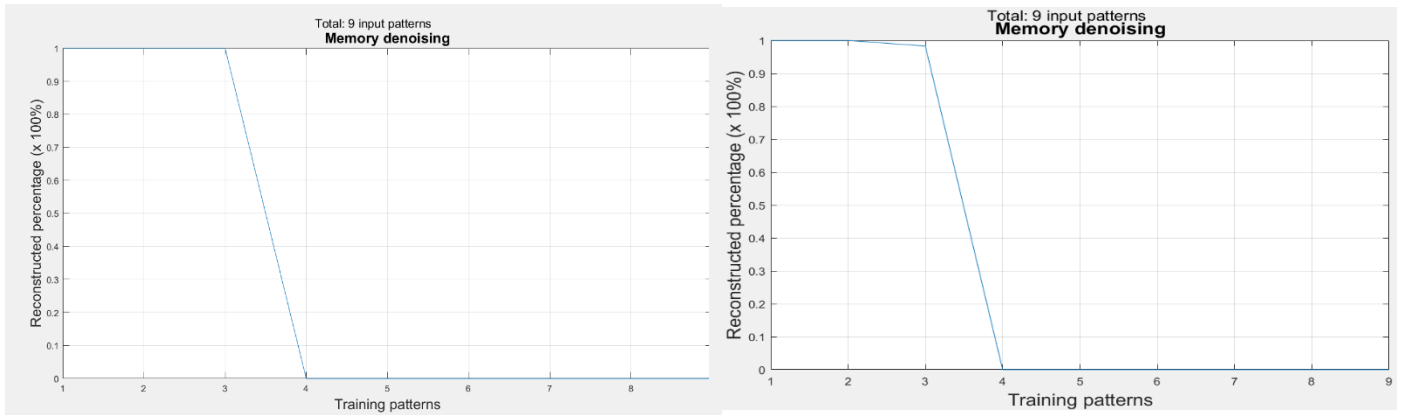


Figure 4.5.1 Denoising memory of all 9 patterns

We tried to put all the 9 patterns into the matrix and then make the algorithm learn with 9 different patterns to get the Weights. However, the performance was only good in 3 patterns. The 4th pattern already have 0% reconstruction percentage (Figure 4.5.1 Left). The degradation is almost abrupt. As there we can see that after the third pattern, there are 0% reconstruction for the 4th pattern. However, in Figure 4.5. Right (when we simulate the same script again) it is seen that the reconstruction for the 3rd pattern decreased from 100% to 98.33%; and the degradation is gradual (but it is still abrupt when it comes to the 4th pattern).

When we tried using randomize patterns as the input (seen on Figure 4.5.2 Top), the network has no problem trying to denoise the input. Hence, we can see from the graph that for all 9 different patterns, our network successfully denotes and can detect all the patterns (seen on Figure 4.5.2 Bottom).

when asked about the relation about “the capacity of Hopfield network is around $0.138N$ ”, we can say that this result depends on the input. Because, we can see that when we randomize the input, it shows better result rather than the “picture” input. The relation we guess is on the amount

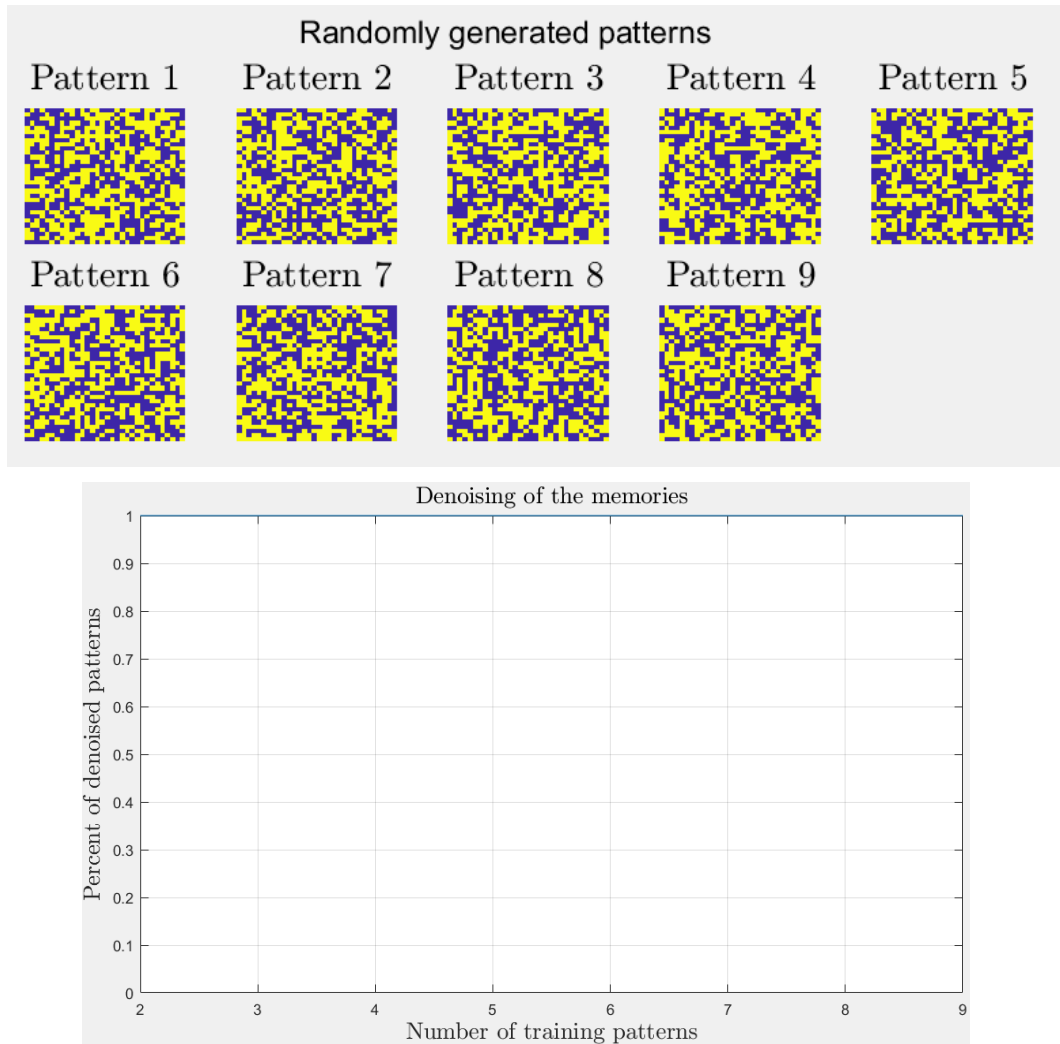


Figure 4.5.2 Top shows the randomly generated patterns. Bottom shows the quality of the denoised memories

of “-1” and “1” in the data. When it is equal, we can say the results are better than when one of the elements are greater than the other.

For the 300 random patterns, we tried using 100, 200 and 300 units as the training pattern shown in Figure 4.5.3. The figures on the left shows the stability of pattern memories and on the right shows denoising of the memories when we flipped the data. What we can say from these experiments is that the stability of the patterns is better on the first part and worsens in the middle but tried to improve when it is heading towards the end. Although, the most stable parts of the data are located in the first 20 data (we assume more than 50% is good). Towards the end, despite it shows “improvement of stability”, the percentage is not reaching 40%. But, this is not the case of improvement, but in fact this shows that the strong connections in the weights are showing that it is showing an illusion of good connection but it is only repeating the same bad data.

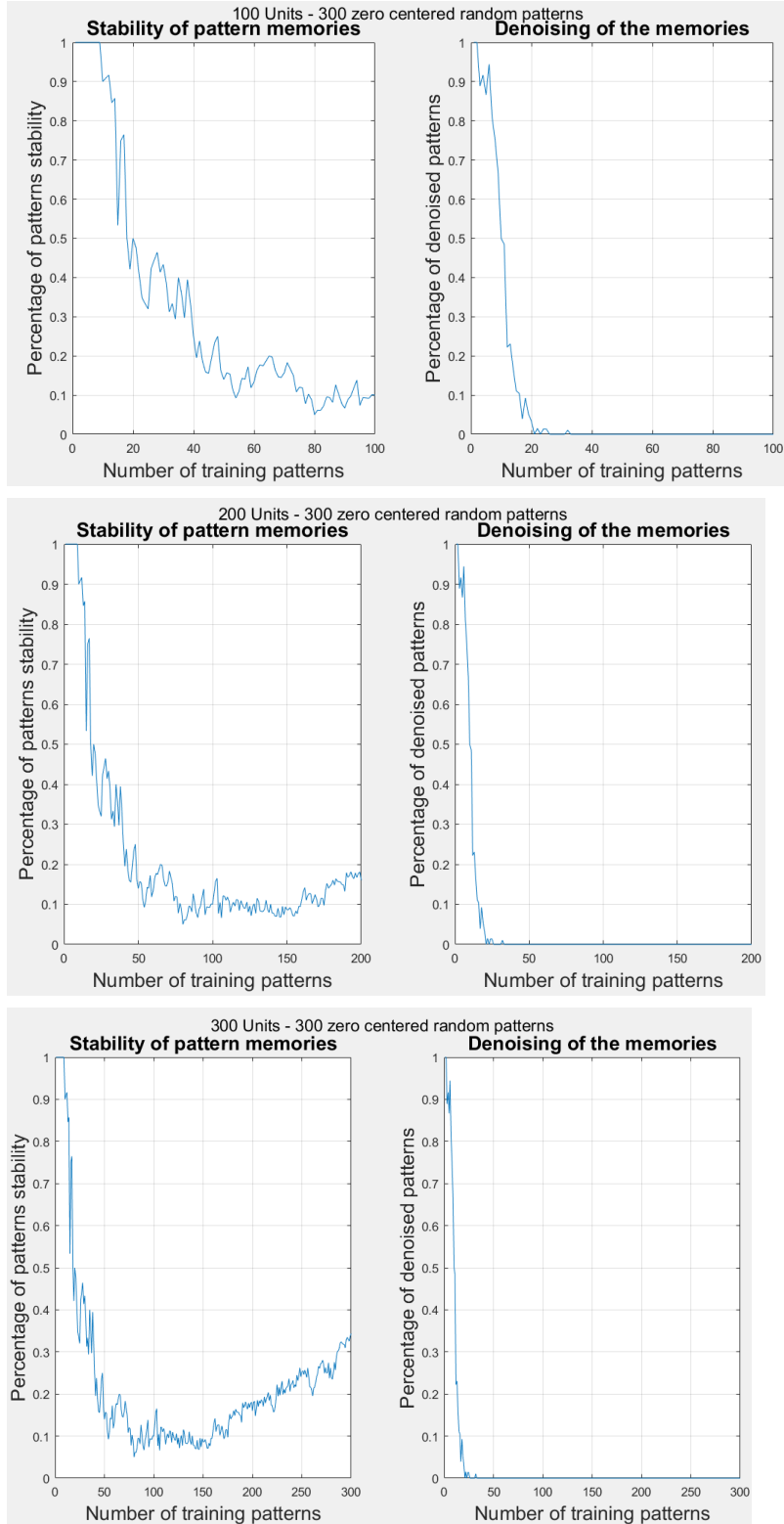


Figure 4.5.3 The stability and performance of 300 Zero centered random patterns. Top: 100 training units, Middle: 200 training units, Bottom: 300 training units

For recalling the data when we flipped the input, all three attempts have the same result. It

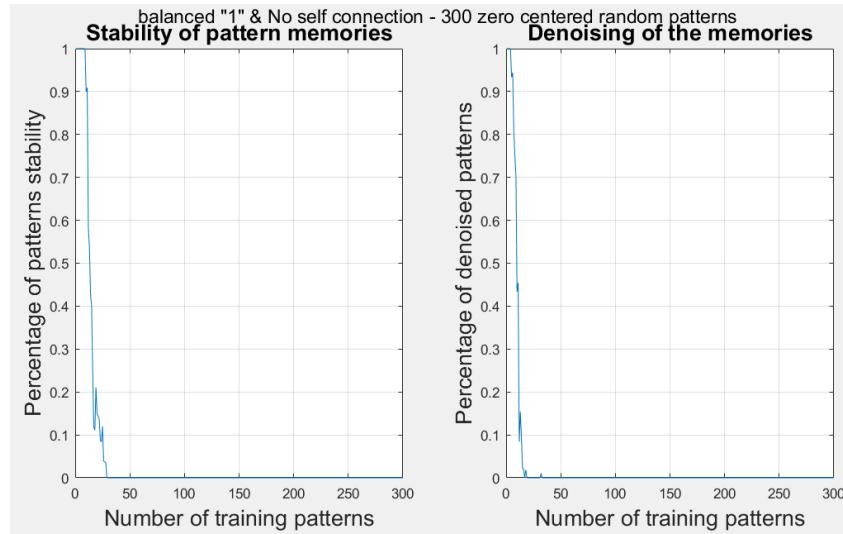


Figure 4.5.4 Balanced input for no self-connection

shows that the data have a good quality in 10 patterns (when we assume 50% is the minimum percentage as a “good quality”).

Larger number of patterns mean that the ability of Hopfield network to store data has surpass its capacity. Hence, we can see that the number of bad data represented when we have flipped input.

Last part of this exercise, we wanted to suppress the self-connections; which is the middle diagonal values of W . Results from this experiment can be seen from Figure 4.4.4. The amount of data which can be retrieved are 13 data, when we again assume that 50% is acceptable. In addition,

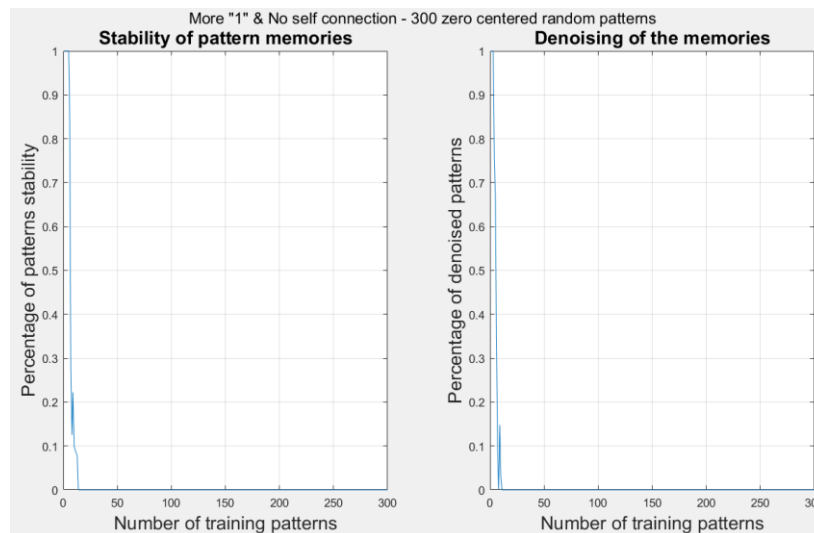


Figure 4.5.5 Unbalanced input for no self-connection

when we try to denoise the data (when we tried using the flipped data), the amount of patterns which acceptable are 9 data.

Now, we wanted to see when we have more 1 rather than -1 by using $\text{sign}(0.5 + \text{randn}(300, 100))$. The graph can be seen from Figure 4.5.5. It can be seen that it only have 6 data which is stable (with 50% error rate), and when we tried to flip the data and denoise it, we only have 5 data. Hence, less data can be used when we have unbalanced input in this type of weight. In conclusion, although the data is not good when recalled, we wanted to see the results from Figure 4.5.4 and 4.5.5. Why? Because it represents the “true” nature of this network and not showing an illusion of “learning” because of the strong network.

4.6 Sparse Patterns

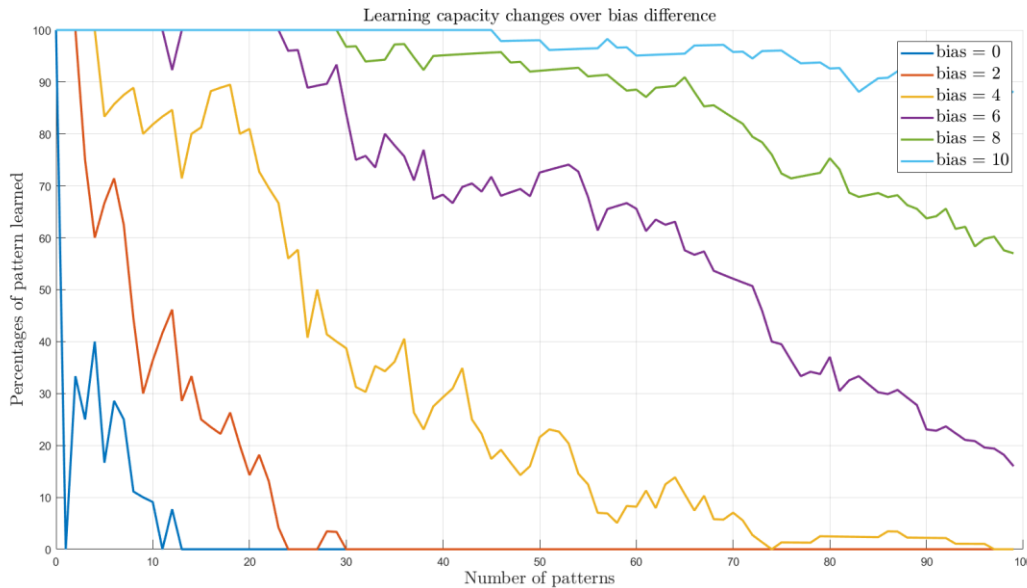


Figure 4.6.1 Graph showing different learning capacity over changing bias value (active patterns = 0.1 = 10%)

In this case we used 100 patterns and 10% of active patterns (binary pattern is used instead of bipolar pattern). From Figure 4.6.1 we observe that by increasing the value of bias (or θ) gradually, the maximum learning capacity also increasing gradually, from almost non-existent in 0 bias into 45 full patterns in bias value of 10.

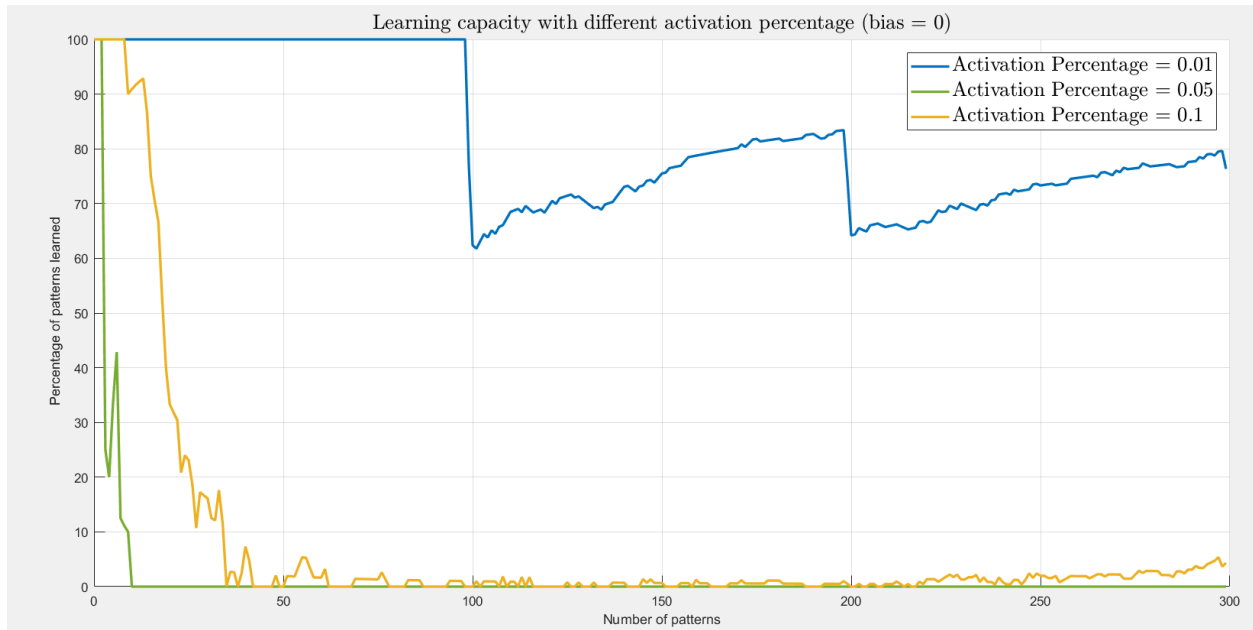


Figure 4.6.2.1 Graph showing progress of learning capacity by different activation percentages

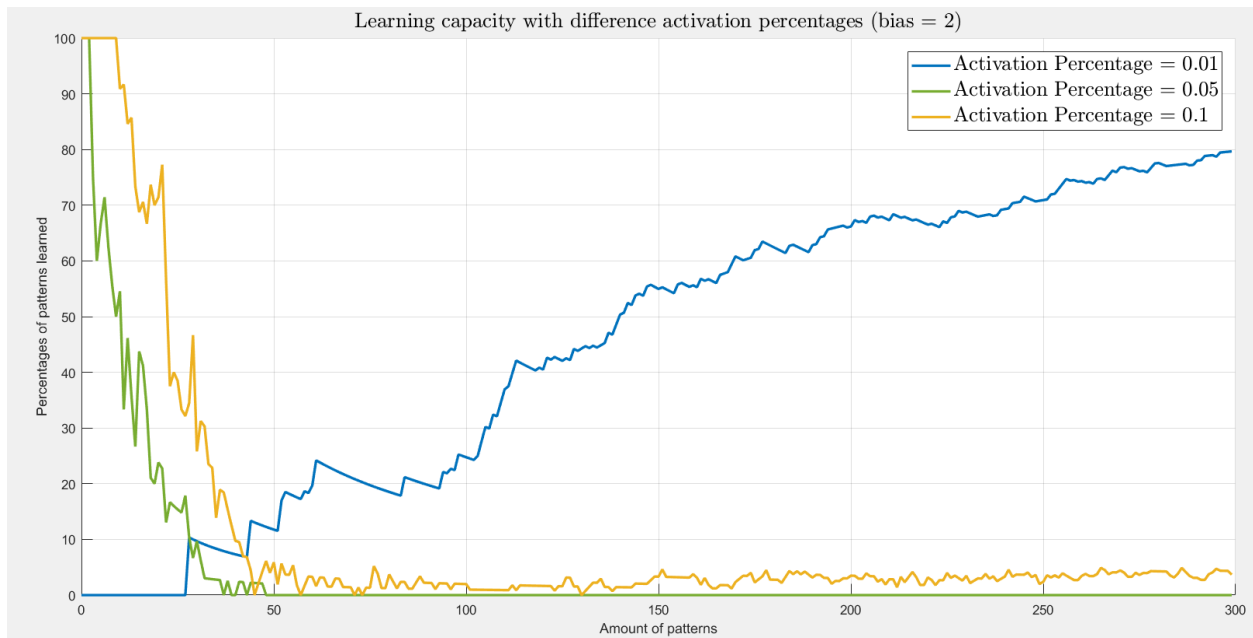


Figure 4.6.2.2 Graph showing progress of learning capacity by different activation percentages

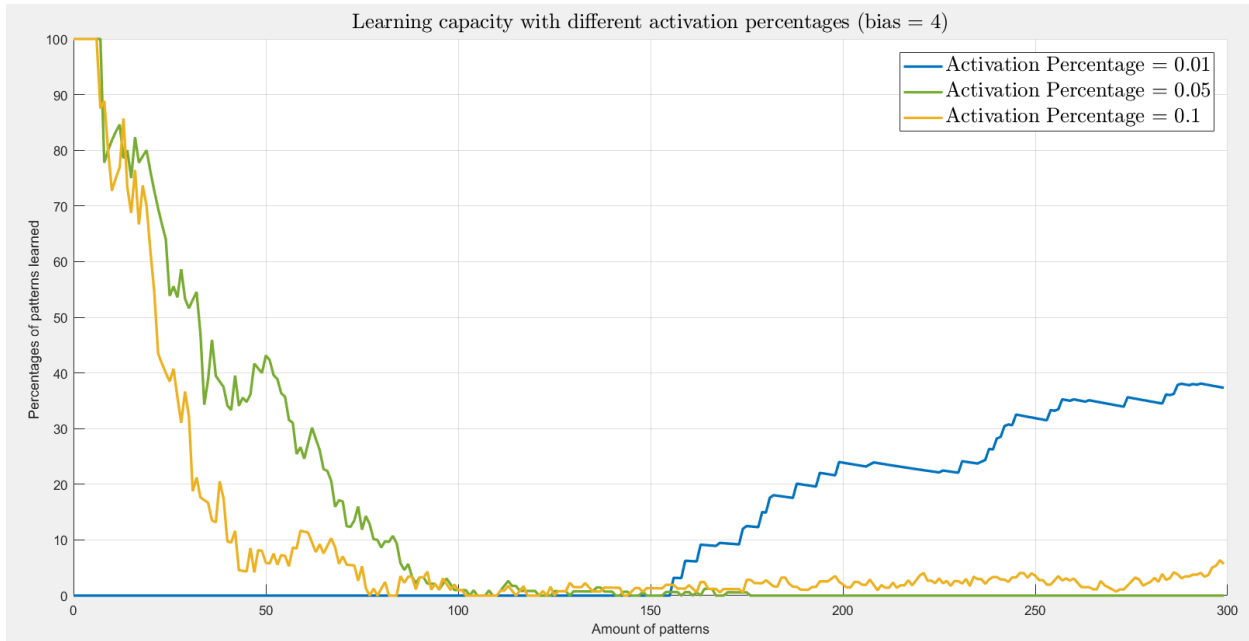


Figure 4.6.2.3 Graph showing progress of learning capacity by different activation percentages

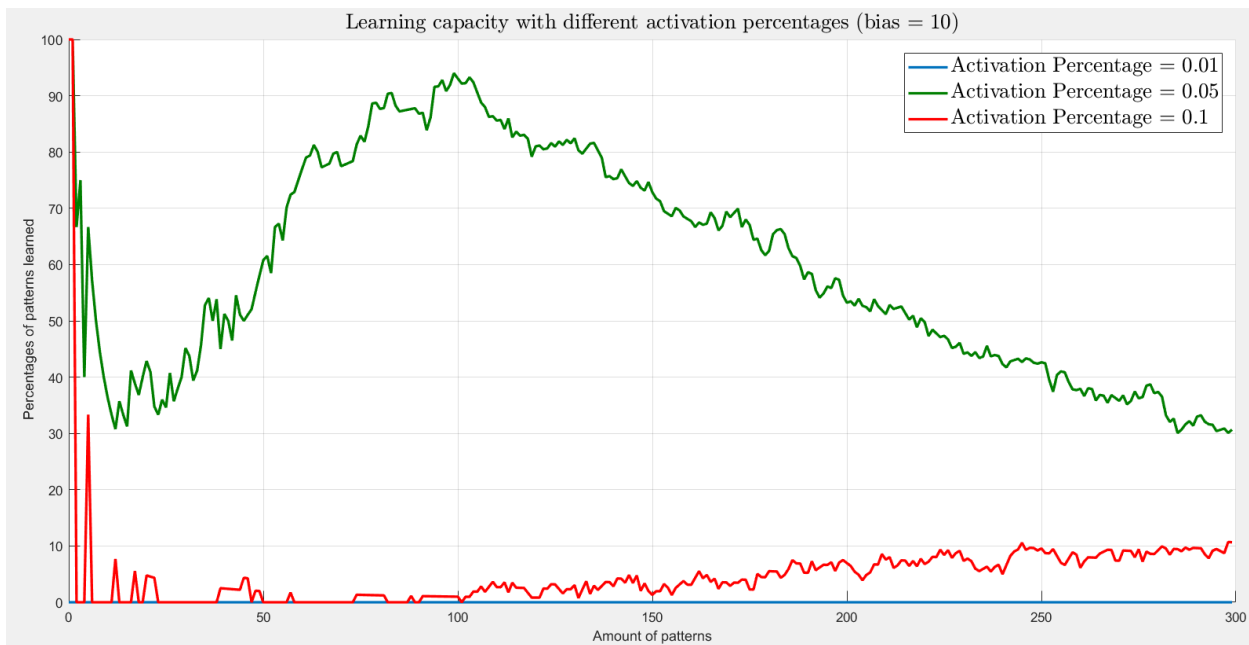


Figure 4.6.2.4 Graph showing progress of learning capacity by different activation percentages

From Figure 4.6.2.1 we see that with bias of zero, the performance of learning capacity increased as we use lower activation percentage in the randomly generated binary pattern. However, in case of using bias value of 2 and 4 (in Figure 4.6.2.2 and Figure 4.6.2.3, respectively)

the trend changes where learning capacity with lower activation percentage (1 %) performs less stable than learning capacity of the higher activation percentage (5% and 10%). In Figure 4.6.2.4, the learning capacity of 1% active pattern performs horribly if compared with the more active patterns. This shows that lower activation percentages in patterns should be used in low or zero bias, while in higher value of bias we can use moderate value of relative active patterns (from the experiment in this case, 5% of active pattern is ideal).

5. Reflection

- Energy is more likely to be same to the reference pattern when we are using less-flipped patterns.
- When we randomize the W matrix, the energy is fluctuating after it goes down in some extent and will be fluctuating until the maximum number of iterations.
- Using a symmetrical W matrix, the energy will be converging farther than the reference energy and will be like that until the end of iterations.
- In case of distorted image pattern, as the percentage of flipping pattern closer to 50%, the network will prone to misclassify the flipped/noised pattern.
- Having a strong connection in the weight matrix doesn't always act as a good thing, as in a large input, they will make an illusion that the network are learning but in fact it acts otherwise
- In case of binary sparse patterns, using bias is recommended especially in sizable number of active patterns.

6. Conclusion

- Energy is larger when we are recalling the memory and will be lower when we have noise. Largest noise will be when we combine 2 different images, rather than flipping the patterns.
- Capacity of Hebbian network is not much for recalling images because of the fact that these patterns look alike the others. Capacity will improve when we sparse the patterns and we used random patterns instead of images
- Activation percentage and the bias value should be chosen and used carefully as they seems to influence each other.
- Noise in pattern can be removed/readjusted as long as the most of the pattern location is similar to available attractors.