Artificial Neural Networks and Deep Architectures, DD2437

Report for lab assignment 3

Hopfield networks

Victor Castillo, Jun Zhang and Balint Kovacs

February 20, 2019

1 Main objectives and scope of the assignment

This exercise was predominantly concerned with Hopfield networks and associative memory. We had to

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

First, we looked at some simple networks using the Hebbian learning principle. We constructed an autoassociative memory of the Hopfield type, and explored its capabilities, capacity and limitations. We were asked to study the dynamics and analyze its origins (potentially explain different behaviours you observe).

2 Methods

For this exercise, we used Python, numpy and matplotlib.

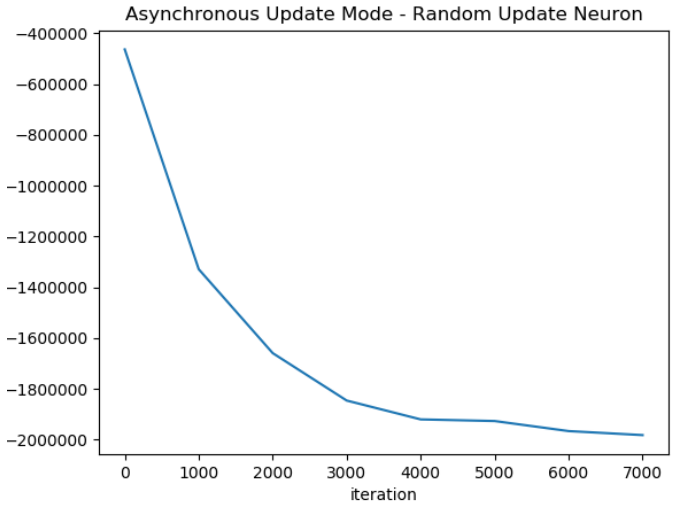
3 Results and discussion

3.1 Convergence and attractors

The network was able to store the tree vectors. In the case of the distorted patterns, the system was able to restore the original patterns for the first one. However, in the case of the second and the third pattern, the resulting association was oscillating between 2 states.

To find the number of attractors in the system, we generated all the possible patterns and checked which points they converge to. We have found 6 attractors: the original patterns, and their opposites. When we make the starting pattern more then half wrong compared to the original, it is closer to the opposite of the target, so recalling it is impossible.

For the asynchronous case with the image patterns in 3.2, we randomly update the neuron. And we can see that as the iteraion goes on, the energy decreases consistently and converge. At the same time the network reaches an attractor, as shown in **??**



Figur 1: Energy of image patterns recalled by p10.

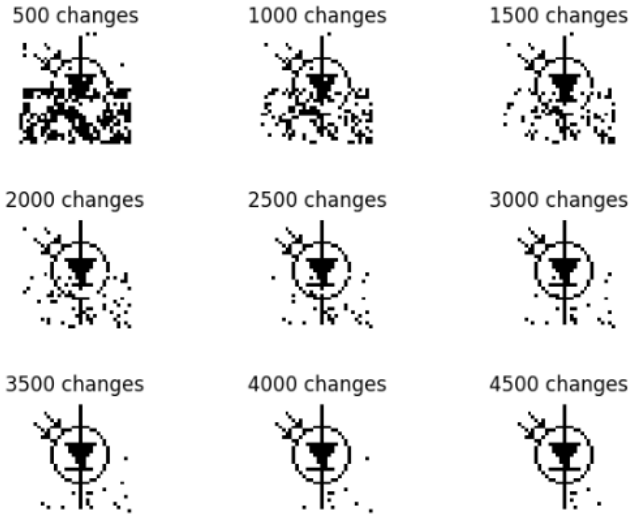
3.2 Sequential Update

The patterns can be stored and they are stable in this case as well. The network can restore `p1` from the distorted version, `p10`, in both the synchronous and asynchronous case. However, in the case of `p11`, it can only recall correctly with the asynchronous method. The restoration of `p10` is displayed in Figure 2, with sequential updates and selecting indices randomly.

3.3 Energy

The energy function has a local minima at the attractors. The energy of the distorted patterns are decreasing over iterations. For the synchronous case with the basic patterns, a minima is achieved in the first 'epoch' for the original patterns, and on the first pattern, it stops at the energy state of the original pattern. However, in the case of the other two distorted patterns, it gets stuck between two incorrect patterns with same energy, resulting in a loop.

When we initialize the weights by random, it gets stuck in a loop for the synchronous case with both symmetric and asymmetric weight matrix. In the

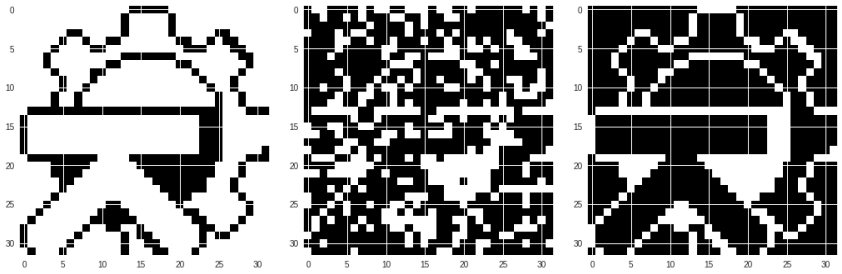


Figur 2: Restoration of image 9 over iterations.

asynchronous case, it converges to a local minima. The weight matrix being symmetric is a neccessaryly condition of convergence.

3.4 Distortion Resistance

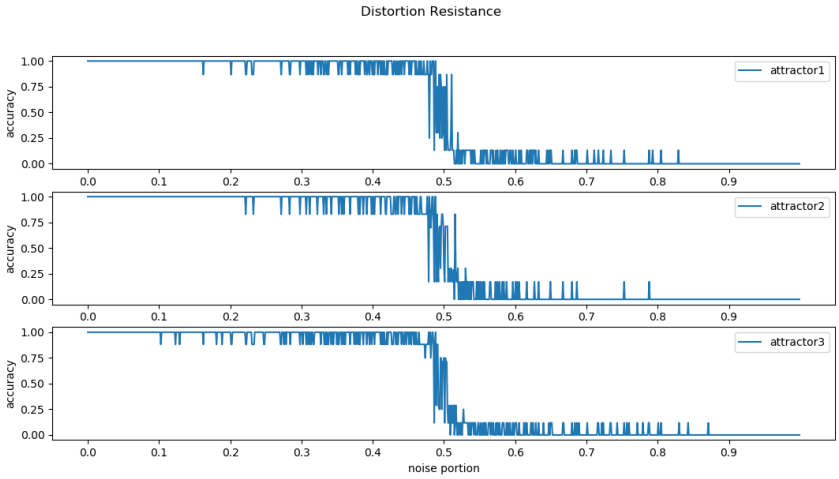
For pattern p1, by flipping random units from 0% noise to roughly 40%, it's possible to recall the original pattern. Between 40% and roughly 80%, the recalled patter is a spurious pattern and from 80% to 100%, the network is able to recall the opposite of the original pattern. The same pattern is seen with p2, as shown in Figure 3.



Figur 3: Original p2, 80% flipped units and recovered pattern.

Regarding p3, around 50% of flipped units result in recalling p1, from 60% to 70% recalls a spurious state and after 80% the opposite of the original pattern is recalled.

If we compare the distortion resistance ability of these three attractors, we can find that they have very similar performance regarding noise tolerance, as shown in Figure 4.



Figur 4: Comparsion between three attractors.

3.5 Capacity

The configuration is only able to remain stable with the three original patterns (p1, p2, p3). Adding additional patterns causes the network to become unstable and abruptly loses the ability to store the patterns.

By generating random patterns instead of the pictures, the network was able to store many patterns, but it starts to lose the ability to remain stable and store all the patterns at around 60 patterns. The difference is due to the high correlation of a point being positive or negative in the case of pictures. This effect does not appear using random patterns.

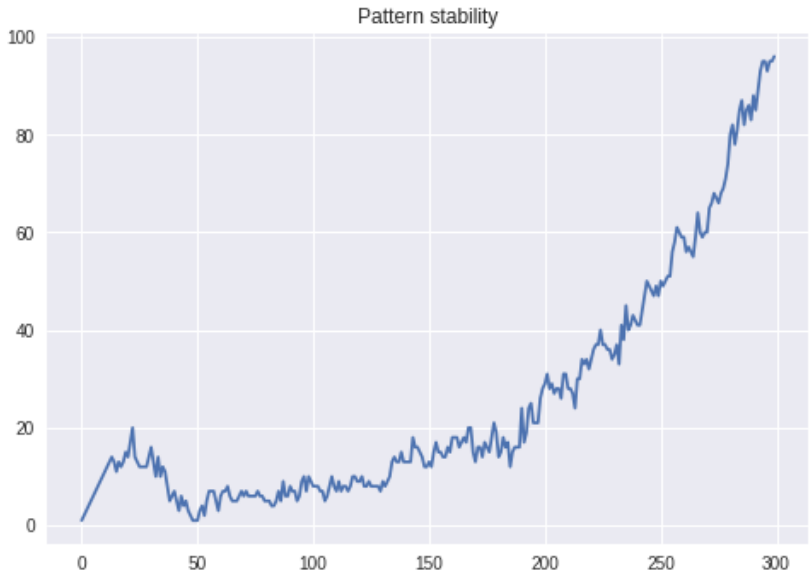
Creating 300 random patterns in a 100 units network we get the behaviour seen in Figure 5.

When adding bias to the patterns, the drop in performance occurs sooner, as can be seen in Figure 6. This is similar to the image data, because there is a skew in how many 'positives' are there, so the data is very imbalanced and that explains the difference when comparing against random generated data, where correlation is very low.

3.6 Sparse Patterns

In this section we use binary (0,1) patterns. We generated 30 patterns for all the experiments for this section. We find that if we set the bias to certain value, the network achieve best performance that it can memorize most figures. And if we decrease the activity, the bias should be decrease accordingly.

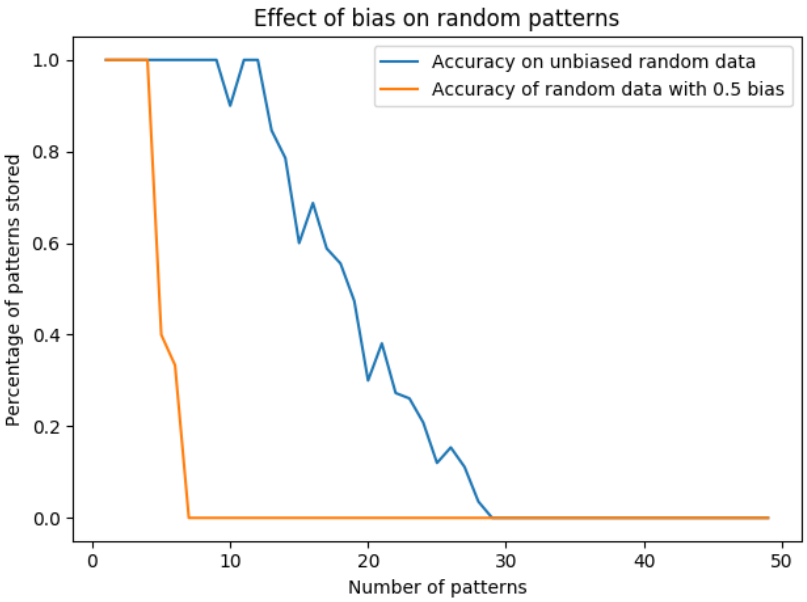
Another interesting finding is that with smaller activity, the network can remember more patterns. Our understanding is if the input patterns are sparse, they are more likely to be orthogonal. The results are shown as 7.



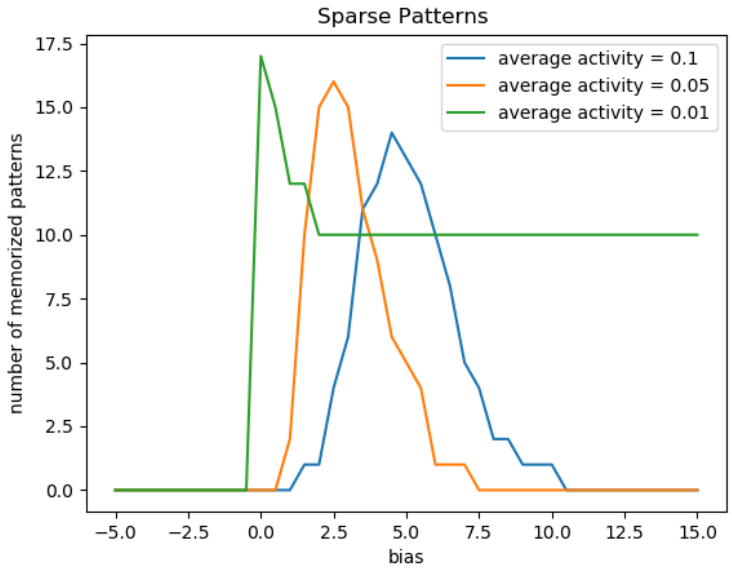
Figur 5: Stability of the network adding more patterns vs previously stored patterns.

4 Final remarks

Using the Hebbian learning was an interesting exercise, since it is a quite different method after perceptron. This assignment was a great way to learn the features and limitations of the Hopfield network.



Figur 6: Storage capacity of random patterns with and without bias



Figur 7: Sparse Patterns