Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 3

Hopfield Networks

Isabella Rositi, Gustav Thorén and Nicolas Guy Albert Wittmann

3 March, 2023

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- understand autoassociative networks and more specifically Hopfield networks
- demonstrate that autoassociative networks are good solutions for noise reduction and pattern completion

This project focuses on Hopfield networks and associative memory. We implemented the networks using the Hebbian learning principle and we constructed an autoassociative memory of the Hopfield type. We explored its behaviour, capacity and limitations.

2 Methods

The code was implemented in Python with the use of the *Numpy* and *Matplotlib* libraries.

3 Results and discussion

We implemented a Hopfield network with different variations on 9 different patterns which can be displayed as 32x32 images. Moreover, we also tried our network on 2 more patterns, one incomplete pattern and an image that combined two different patterns, to test the capability of the network.

3.1 Convergence and Attractors

For this first task we used three simple patterns made up by 8 bits each and then created a distorted version of them (1 or 2 bits changed) and a very distorted versions (more than 50% of the bits changed). We implemented a synchronous update of the neurons and we stopped once reached convergence. Due to the very small network, the iterations needed were only 2 for both sets of patterns, however for the distorted patterns, 2 out of 3 have been perfectly stored and retrieved, whereas the one not stored perfectly presents a wrong unit. The very distorted set, on the other hand, didn't allow the network to properly store any of the patterns.

The attractors present in this network were found to be 14, among which were the three initial patterns. We don't know, however, how big their basin of attraction is.

3.2 Sequential Update

We learnt the first three patterns in the data set (shown in Figure 1) and tested the ability of the network to complete a degraded pattern. We tried to restore two different images: the first is a degraded version of one of the learned patterns, and the other is a mixture of the other two patterns.

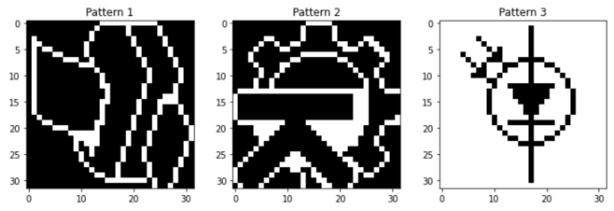

Figure 1: Three patterns learned by the network.

We tried to restore the images both with the synchronous and asynchronous update of the units, in order to retrieve the best images. The synchronous update (*Little Model*) produces excellent results for the first degraded pattern, but fails at restoring the second mixed pattern, as shown in Figure 2

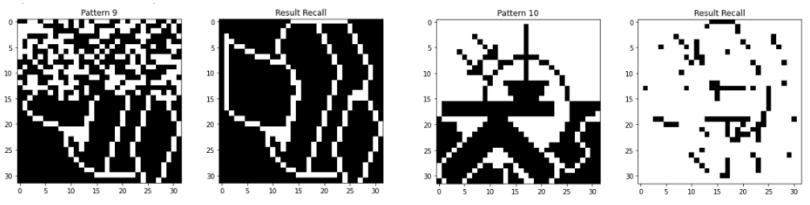

Figure 2: Restoring of two degraded patterns with synchronous update of units.

The *Little Model* reaches convergence basically immediately (after 2 and 3 iterations for the two patterns), and while it provides satisfactory results for the first pattern, it doesn't for the second. Then we implemented an asynchronous update algorithm which selects units randomly when updating. In Figure 3 we can see the update states throughout 5000 iterations. We observe that for the first image we obtain the restore pattern only after 5000 iterations, whereas the second image converges towards one of the two patterns composing it.

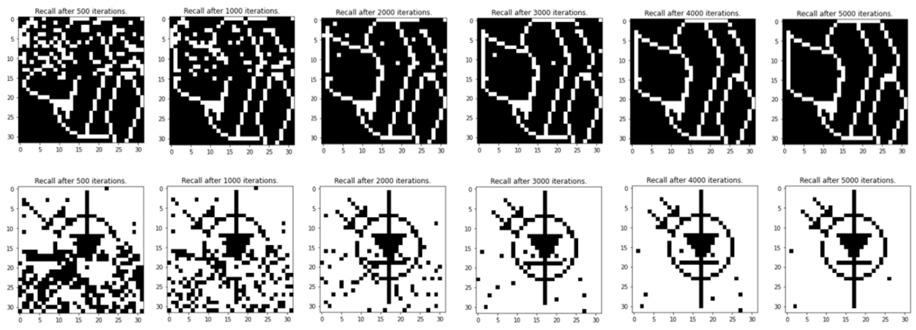

Figure 3: Restoring of two degraded patterns with asynchronous update of units throughout 5000 iterations.

3.3 Energy

We can observe that the energy at the attractors is very low, while it is higher for the distorted patterns.

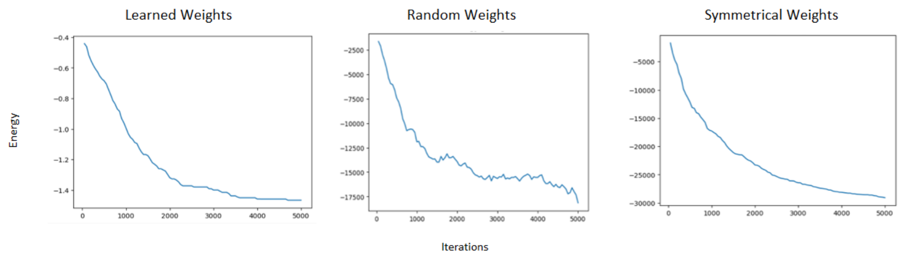

Figure 4: Evolution of the energy during the recall with the learned weight (left), a random weight matrix (center) and a random symmetric weight matrix (right).

We can see from the left plot in Figure 4, that when we use the sequential rule to approach the pattern the energy either declines or it is constant at every step of the process. It converges toward the energy of the pattern we try to recover. On the contrary, when the weight matrix is randomly generate, the energy decreases, but it's not guaranteed that at some steps it could not increase again (center plot). When we force the matrix to be symmetric the energy can only decrease throughout the recall process, since this is one of the property of the process when implementing a symmetric weight matrix (right plot).

3.4 Distortion Resistance

In this section we add random noise by randomly flipping a certain percentage of the pixel to test the resistance of the model.

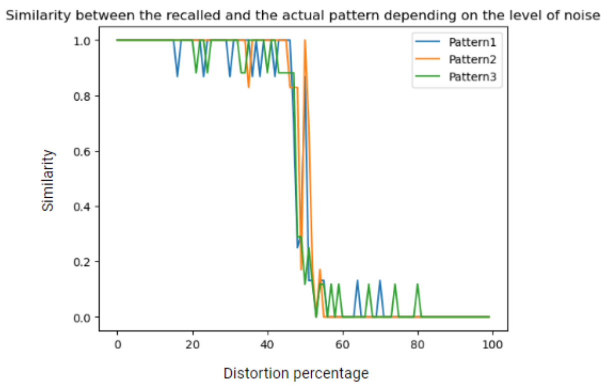

Figure 5: Measure of the similarity of the recalled image and the real pattern depending on the quantity of noise.

In Figure 5 we can observe that the model successfully restores the pattern for an image with up to 40% of noise. The trend is similar for all 3 patterns. Depending on the noise level we reach attractors which are not the actual pattern. When the level of noise is low we might still end on an attractor somewhat similar to the actual pattern, but when there is close to 50% of noise, the image is close to random and the network can reach any attractor, which can be a completely different learned pattern, as shown in Figure 6. When there is above 50% of noise, every pixel is more likely to be flipped than not, which makes the recall process end in an attractor which is the mirror image of the actual pattern (every black pixel is white and every white pixel is black).

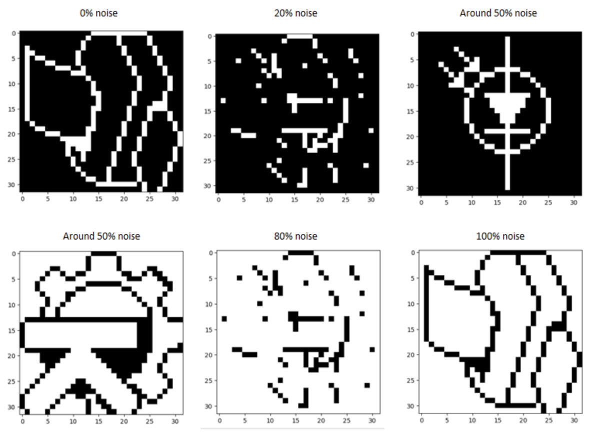

Figure 6: Different attractors the recall process reaches for the same pattern depending on the level of noise.

3.5 Capacity

The capacity of the network is heavily dependent on the distribution of the data. When using the pictures from part 3.2 the network could only remember a maximum of 3 pictures. Training with more pictures resulted in the trained pictures no longer being attractors and therefore not being stable or possible to recall. On the other hand, when the network was trained with a randomly generated data set it performed far better. The same network could hold roughly 100 patterns before failing to recall them.

Fig. 7 shows how much of the trained data was stable as the network was trained with more patterns for different amount of noise in the input. The figure shows how the network performed better when there were self connections in the weights, especially when the data had no noise. However, Hopfield networks are typically desired for their noise suppressing capabilities. Removing the self connections worsens the performance on data with low noise but gives a more consistent performance with different noise levels, as can be seen in the right figure. Since we typically use Hopfield networks with noise data we prefer to remove self connections for a more consistent performance.

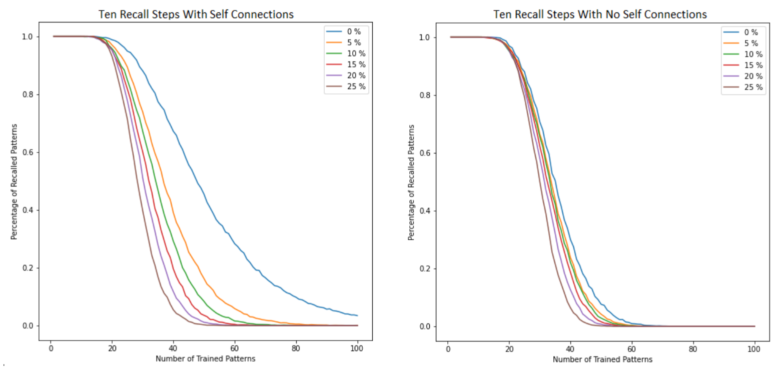

Figure 7: The percentage of pattern that could correctly be recalled after 10 synchronous recall steps with different levels of noise (flipped bits).

We also tested the performance when there was a bias towards one color. The results can be seen in Fig. 8 and shows how there is a large loss in capacity when the data is biased. With as little as 70% of one color barely a fifth of the patterns could be stored compared to 50%.

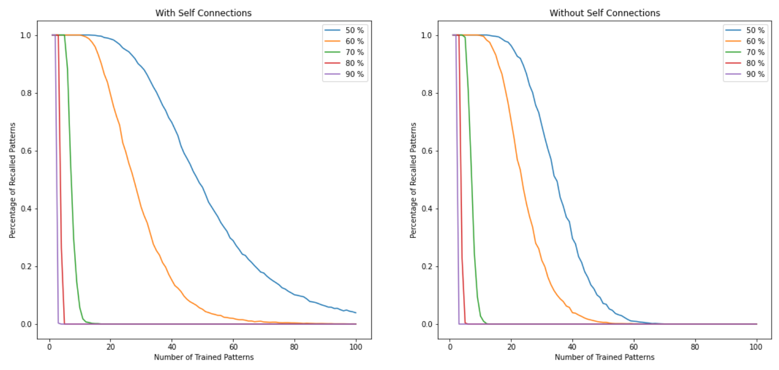

Figure 8: The percentage of pattern that could correctly be recalled after 10 synchronous recall steps with different levels of noise (flipped bits) and bias.

3.6 Sparse Patterns

We then carried out an experiment testing how many patterns could be correctly memorized by the network as the value for the bias changes (going from 0 to 15), as well as the activity of the units (10%, 5% and 1%).

Fig. 9 shows the performance of the network with different activity levels and recall bias. All of the data is noiseless. We can see that there is a peak for each of the bias value where the network memorizes the best. When the bias is at the same level as the activity the peak seems to be at the same position but with a larger breadth for lower biases. We do not know why this is observed. Intuitively the performance should worsen when storing more pattern but instead it seems to improve around the bias.

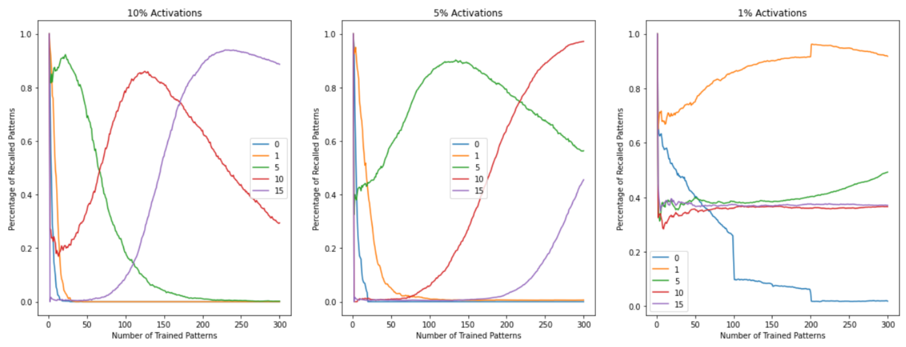

Figure 9: Number of correctly memorised patterns recalled with synchronous update. Shows the result with different bias in the recall update and activity level in the memorized pattern.

4 Final remarks

From this project resulted that implementing the asynchronous update of the units is slower than the using the *Little Model* (synchronous update), however, the sequential update could eventually lead to recall patterns that the synchronous is not able to.

Moreover, the network is able to overall restore images with a fair amount of distortion. With a low level of noise the model restores the actual image, with around 50% of noise it reaches random images among the attractors, and with a high percentage of flipped pixels the image converges towards the mirror image of the actual pattern.