Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 3

Hopfield Networks

Maximilian Auer, Lukas Frösslund and Valdemar Gezelius

October 5, 2020

1 Main objectives and scope of the assignment

The main objectives of this lab was to understand the fundamental principles of autoassociative memory, and how we can represent it by building and training a Hopfield Network. A main component withing the scope of the assignment was also to look at several aspects of the Hopfield Network, with regards to the energy function, noise reduction, storage capacity and sparse representations.

2 Methods

For all parts of the lab, Python 3 was used together with NumPy for mathematical operations and Matplotlib for plots.

3 Results and discussion

3.1 Convergence and attractors

We saw that when using synchronous update, the three patterns $x1$, $x2$ and $x3$ where perfectly recalled and converged with the learned patterns. When recalling the distorted pattern $x1d$, $x2d$ and $x3d$ they again converged with the correct attractors and recalled perfectly.

Doing an automated search by recalling all possible combinations of length 8, we saw that the network converged to 12 different attractors, where there were 6 pairs of pairwise inverted attractors. When increasing the distortion of the recalled patterns we saw that they had a larger tendency to move away from the three patterns that we trained the network on, but sometimes having a correct solution as well.

3.2 Sequential Update

Using the larger input patterns that where given in the *pict.dat* data set and having a asynchronous/sequential approach we saw that when training on the first three patterns we also get three stable recalls on those. See recalled patterns in images below.

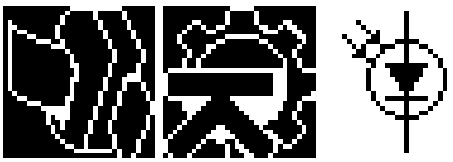

Figure 1: Patterns recalled perfectly.

We can also see that the network can recall a degraded pattern using synchronous learning. See images below.

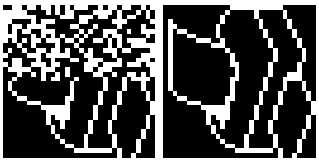

Figure 2: Perfect recall of a degraded pattern. **Left:** Degraded pattern. **Right:** Recalled pattern.

In the case of the pattern that is a mixture of two other patterns, our network could not recall correctly and recalled it to a non-trained attractor using synchronous learning. See images below.

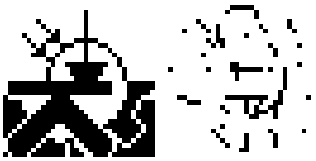

Figure 3: Wrong recall of a degraded pattern. **Left:** Mixed pattern. **Right:** Recalled pattern.

We could however recall one of the mixed patterns when using asynchronous learning. See images below.

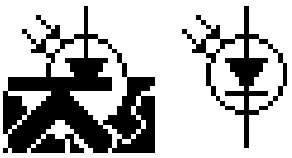

Figure 4: Almost correct recall of a degraded pattern. **Left:** Mixed pattern. **Right:** Recalled pattern.

3.3 Energy

The energy at the different attractors and at the distorted patterns can be seen in Table 1. What is clear is that the energy is much lower at the attractors, which is logical given that they correspond to the local minimum in the energy function. Note that we did not include the normalization term in the weight matrix initialization. If we had included it, the weights would have all been smaller negative numbers, but the proportion between them would have been the same.

Pattern	Energy
P1	-1470864
P2	-1395344
P3	-1494272
P10	-422892
P11	-174592

Table 1: Stored attractors and distorted patterns and their respective energies.

In Figure 5 below, we can see the evolution of the energy across iterations for the distorted patterns P10 and P11 respectively, as they approach an attractor. In this experiment, P10 converged to P1 and P11 converged to P3, using sequential updates with a random ordering.

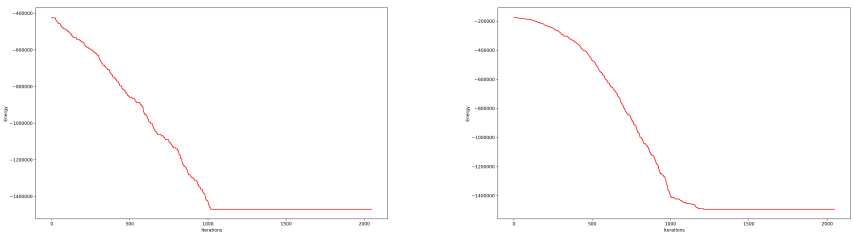

Figure 5: Energy across iterations for distorted patterns when approaching an attractor. **Left:** P10 converging to P1. **Right:** P11 converging to P3.

When generating a completely random weight matrix using normally distributed zero-mean numbers, and iterating from an arbitrary starting state, the network does not converge. As can be seen in Figure 6 below (blue line), the energy

goes up and down in a slightly irregular fashion and by examining the energy on the y-axis we can conclude that it is not even close to converging to a stored attractor. When we instead make the weight matrix symmetric, there is a clear change, which we can see in Figure 6 (orange line). The learning process here is very slow, but stable. Now, the energy always decreases across iterations, even if it is by a very small amount.

To understand why the energy always decreases for the symmetric weight matrix only, we need to examine the energy function for a given state x, and the change in energy when moving from one state x_j to a new state x_j*, given a symmetric weight matrix.

$$E(x) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j}x_i x_j \tag{1}$$

$$\Delta E_{x_j \to x_{j*}} = -\frac{1}{2}(\sum_{i}^{n} w_{i,j}x_i x_j * - \sum_{i}^{n} w_{i,j}x_i x_j) = -\frac{1}{2}(x_j * - x_j)\sum_{i}^{n} w_{i,j}x_i \tag{2}$$

Because of symmetry, we can remove the double sum and replace it with two single sums. Then we can further reorder and simplify the expression into one that contains our actual update rule. This is of course very beneficial because if we know the values of x_j and x_j*, we also know if this sum is positive or negative. Let's assume the update results in a flipped unit. In the first case, $x_j = -1$ and $x_j* = 1$. This means that $\sum_i^n w_{i,j}x_i \geq 0$, and thus the complete expression is negative. For the second case, where $x_j = 1$ and $x_j* = -1$, all three terms in the expression will be negative, and thus the complete expression will once again be negative. So, for a symmetric weight matrix, the energy will always decrease when a unit is flipped.

The roughly 5000 iterations in this experiment was not enough for it to converge to an attractor, but given enough iterations it will eventually converge. Despite this, we should stay away from random initialization of the weight matrix, because the difference in speed is very notable. As can be seen in Figure 6, the distorted patterns required circa 1000 iterations to converge to a stored attractor when initializing the weights with the stored patterns, much faster than a random but symmetric weight initialization.

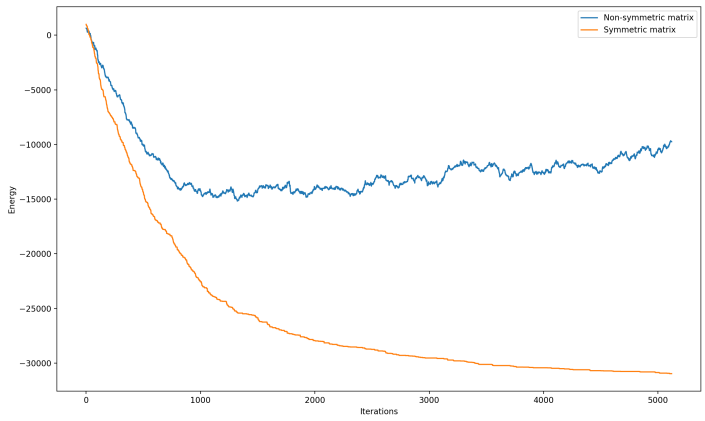

Figure 6: Energy across iterations from an arbitrary starting state for a random gaussian weight matrix, non-symmetric and symmetric.

3.4 Distortion Resistance

Because of the stochastic nature of both the asynchronous updates in the model as well as the random choice of nodes to add noise on, we ran a multitude of experiments to calculate how much noise that could be added, and then averaged by taking the ratio of correctly converged experiments at each given noise level, for each of the three stored attractors. As can be seen in Figure 7, all three patterns behaved similarly when adding noise, with small differences in ratios for noise levels between 10% and 50%.

At exactly 50% noise, no pattern converged to the correct attractor for any of the 50 experiments. This was a drastic difference from 40% noise, where all three patterns had a ratio above 0.5, i.e. more than half of the experiments resulted in correct convergence at that noise level.

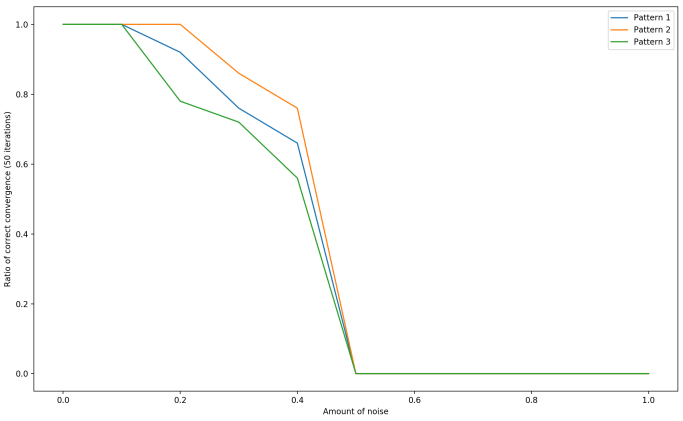

Figure 7: Ratio of correct convergence for all stored attractors at different levels of noise, across a total of 50 experiments.

The extra iterations beyond a single-step recall definitely helps, because it takes a multitude of iterations for the patterns to converge to an attractor, especially for higher levels of noise. When visualizing the recalled pattern, other attractors could be noticed, most notably the opposites of all three stored patterns. Several spurious patterns could also be seen, which exists because there exists correlation between the stored attractors P1, P2 and P3.

3.5 Capacity

When we add P4 to P1, P2, P3, we see that the network can not recall any of the four moderately distorted patterns. An explanation to this could be that when we study the correlation between the patterns, we see that P3 and P4 has a similarity of ≈80%. This means that they are far from orthogonal. Same result for P5-P7. When we add P8 and P9 to P1-P3 separately, we see that the network is able to recall pattern P8/P9 but non of patterns P1-P3.

Figure 8 shows the ratio of correctly recalled patterns of combinations of the patterns P1-P9, with 10% noise added. What we mean by correctly recalled in this regard is that the network recalls all the stored patterns in a combination correct. We see that the drop is almost abrupt after 3 stored patterns and 5-9 patterns can not be recalled. We interpreted the definition of a stable pattern as a pattern that doesn't change for 1 iteration of all units, with this definition all patterns was stable, but for >5 patterns stored, the recalled stable patterns did not match the stored ones.

When we increased the noise to 40% only ≈1% of the combinations of 4 patterns could be recalled, e.g the combination [P2, P5, P7, P8] which makes sense if we look at the relative dissimilarity between the patterns.

When we created our own 9 random "pictures" (size (1, 1024)) we saw that the randomness and even distribution of values makes it possible to store and

recall all combinations of the patterns. The patterns are are less correlated than P1-P9, a requirement for the network to reach the theoretical storage capacity of $0.138N$, N being the units in the pattern.

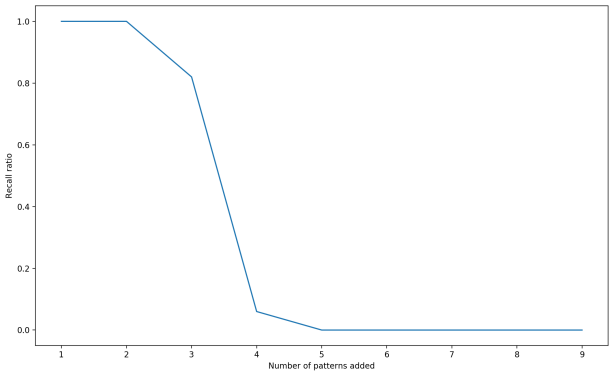

Figure 8: Recall ratio for correctly recovered pattern combinations of P1-P9, given stored patterns. 10% noise added.

When we created random, evenly distributed patterns with 100 units we saw that the number of recalled patterns rose after 50 added patterns with no noise and self-connections, as shown in Figure 9, left. This artifact is caused by the self-connections will to remain at the current state. When we removed self-connections this artifact disappeared, as shown in Figure 9, right. We see that the network can store 0.138*100≈14 patterns.

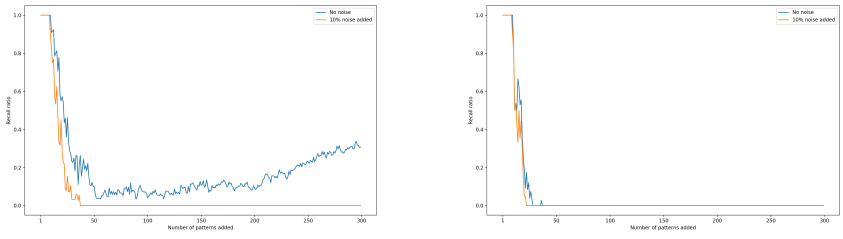

Figure 9: Recall ratio for patterns added to network. Clean- and noisy patterns recalled. **Left:** With self-connections. **Right:** Without self-connections.

When we skewed the distribution and made the patterns more bias towards 1 than -1, we saw that the network could store fewer patterns, as shown in Figure 10. This distribution or a distribution bias towards -1, is similar to the picture patterns P1-P9.

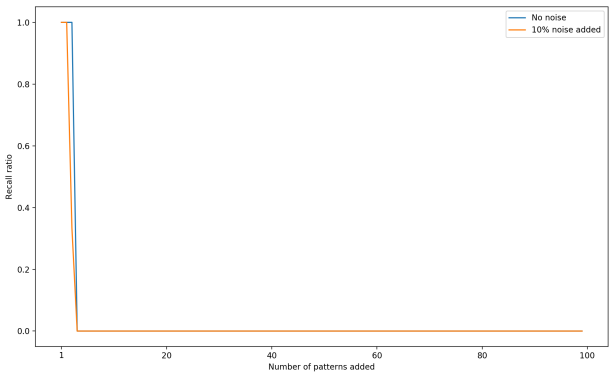

Figure 10: Recall ratio for correctly recovered pattern with bias patterns, given stored patterns. Clean- and noisy patterns recalled.

3.6 Sparse Patterns

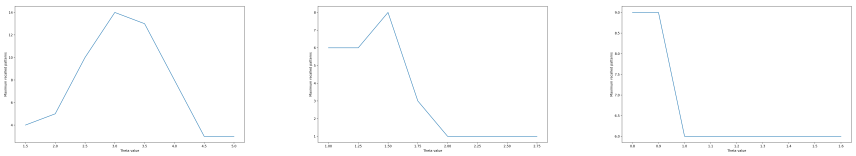

Figure 11: Maximum recalled patterns for different $\theta - values$. **Left:** $\rho = 0.1$ **Center:** $\rho = 0.05$ **Right:** $\rho = 0.01$

To deal with a more real life scenario we adjusted the learning rule to deal with unevenly balanced data, since we in this case knows the average activity of the dataset, ρ. Figure 11 shows the maximum of correctly recalled patterns, i.e the network can recall all stored patterns checked, for different values of θ. We see that a higher rho-value, i.e less sparse data, requires a higher theta value, and vice versa. We see that, with θ=3 and ρ=0.1 the network can store 14 patterns, with θ=1.5 and ρ=0.05 the network can store 8 patterns and with θ=0.9 and ρ=0.01 the network can store 9 patterns. This is a considerably improvement to the picture patterns P1-P9 from section *3.5*.

4 Final remarks

Overall, the lab yielded good insights into most aspects of the Hopfield Network and auto-associative memory, and granted us with an understanding of the limitations of the network. Some of the terminology could have been more clearly described in the assignment description.