Artificial Neural Networks and Deep Architectures, DD2437, Group 44

Short report on lab assignment 3

Hopfield networks

Patrik Ekman, Eric Hartmanis, Martin Koling and Daniel Wass

October 9, 2020

1 Main objectives and scope of the assignment

This report presents the work and results for assignment 3 in the course DD2437. The assignment concerned Hopfield networks and our main goals in the assignment were to:

- Gain a deeper understanding of Hopfield networks and autoassociative networks.
- Further our understanding of attractor dynamics and the energy function for Hopfield Networks.
- Obtain a better comprehension of the effect of noise and the ability for autoassociative networks to recall patterns.
- Gain deeper insights into the storage capacity of Hopfield networks.
- Better understand the storage capacity for biased, sparse patterns.

2 Methods

We have coded this assignment in Python3 using Microsoft's developer environment Visual Studio Code as it provides a convenient collaborative live sharing feature. The python library Matplotlib was used to visualize the results and numpy was used for computations.

3 Results and discussion

3.1 Convergence and attractors

When applying the update rule using *the little model* to the distorted patterns *dx1, dx2* and *dx3* until a stable fixed point was reached, only the distorted pattern *dx1* could be perfectly recalled from memory, and converged towards the stored pattern *x1*. At the point of reaching the stable fixed point, the recall of the pattern *dx2* had 6/8 features correct while the recall of the pattern *dx3* had 7/8 features correct.

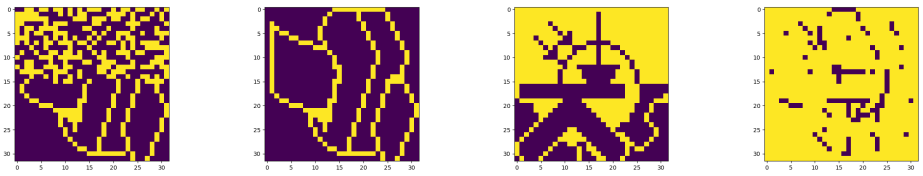
In order to find all attractors in the network, every possible 8 bit pattern containing -1 and 1 was put into the network too see whether or not the pattern could be perfectly recalled. Besides the given patterns *x1, x2* and *x3*, we also found three other attractors in the network making a total of six attractors for the network. These were the polar opposites of the original three patterns, meaning that the sign of every bit in the original three patterns were flipped.

When the starting patterns were even more distorted, with more than half of the bits being erroneous, the recalled patterns seemed to converge to the other attractors in the network, i.e. the opposite ones. This is quite intuitive as the distorted pattern is more alike the opposite one already from the start.

3.2 Sequential update

For this part, the dataset *pict.dat* containing 9 original patterns and two distorted ones was used. The patterns themselves were much larger, 1024 bits, meaning that the network was a 1024 neuron network. When learning the first three patterns, *p1, p2* and *p3* we found that none of them exhibited any change when continuously updating the network, meaning that all of these three patterns were stable.

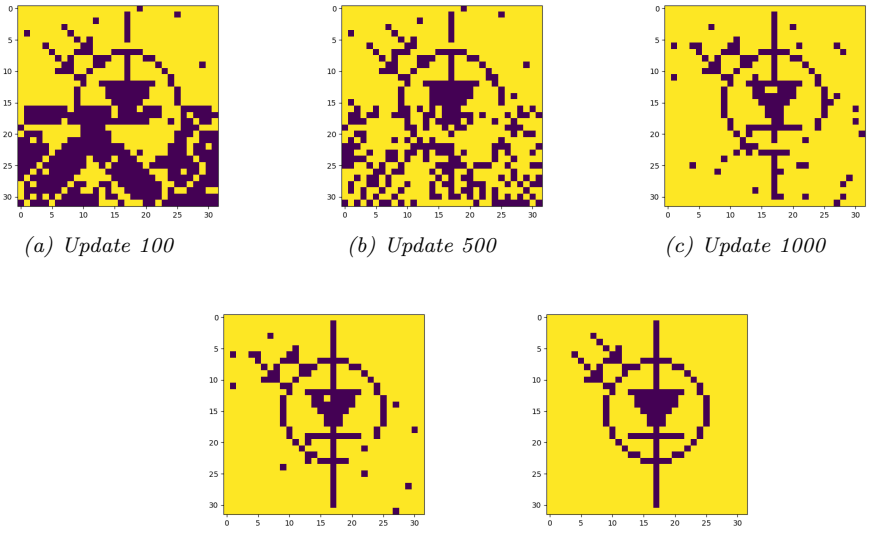
Using the first three patterns and the little model, the network could only complete the degraded pattern *p10* which seemed to be one part noise and one part pattern *p1*. It could not however complete pattern *p11*, the combination of pattern *p2* and *p3*. The results from the network trying to complete the mentioned patterns are shown in *Figure 1*.



(a) Pattern 10 *(b) Pattern 10 recalled* *(c) Pattern 11* *(d) Pattern 11 recalled*

Figure 1: The pictures of the distorted patterns and the network's attempt to complete them

When using asynchronous updates, in the spirit of the original Hopfield dynamics, and iteratively selecting the units to update randomly, the degraded pattern *p11* actually completely converged towards one of the patterns (pattern 3) that it was comprised of. The results from using this method on the degraded pattern *p11* is visualized in *figure 2*. As an experiment, we also tried removing pattern 3 from the network and instead only constructed it from pattern 1 and 2. Trying to complete the degraded pattern *p11* using this network, it instead converged towards pattern 2, which is the other half of the degraded pattern.



(a) Update 100 *(b) Update 500* *(c) Update 1000*

(d) Update 1500 *(e) Update 2000*

Figure 2: The completion of pattern p11 when using asynchronous updates on random neurons.

3.3 Energy

The energy of a network with symmetric connections can be calculated through the Lyapunov function: $E = -\sum_i \sum_j w_{ij} x_i x_j$. The energy at the attractors *p1, p2* and *p3* as well

Table 1: The energy at the different attractors and distorted points.

Attractor	Energy
p1	-1436.39
p2	-1362.64
p3	-1459.25
p10	-412.98
p11	-170.50

as the distorted points *p10* and *p11* is presented in table 1.

The energy change from iteration to iteration when using the sequential update rule for both the pattern *p10* and *p11* is displayed in *figure 3* below. As can be seen, pattern *p10* reaches its lowest point of energy after just one epoch, whereas the energy for pattern *p11* continues to decrease during the second epoch. It is evident that the energy strictly decreases for every update.

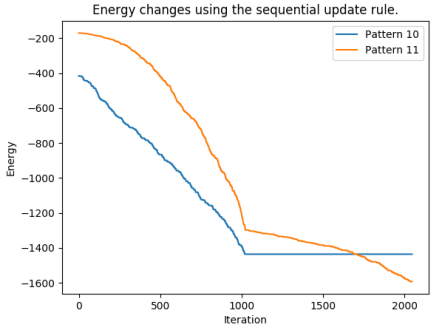


Figure 3: Energy decreases during the between iterations using sequential update for both of the distorted patterns.

If the weight matrix is drawn from a zero mean gaussian distribution, rather than taking the outer product $W = \bar{x}^T\bar{x}$, the decrease in energy exhibits other properties. If the weight matrix is not made to be symmetric, the energy does not converge and although a downwards trend for the energy can be discerned, it fluctuates up and down quite a bit for both pattern *p10* and *p11*, see *figure 4*, rather than strictly decreases, as in *figure 3*.

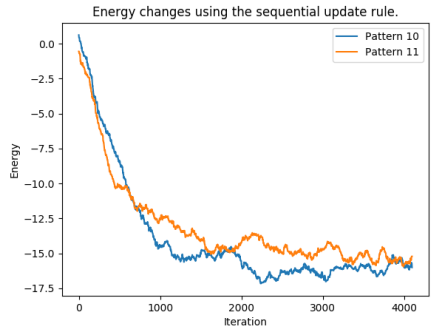


Figure 4: Energy decrease with a random, non-symmetric, weight matrix drawn from a zero mean gaussian distribution.

If the random weight matrix instead is made to be symmetric, by setting $W = 0.5 * (W + W^T)$, the decrease in energy for the both patterns have a much smoother trajectory, once again strictly decreasing. Furthermore, at the end of the 2048 iterations, the difference in

terms of absolute energy for both patterns was almost twice as low when compared to the instance of not making the random weight matrix symmetric. This is displayed in *figure 5* below.

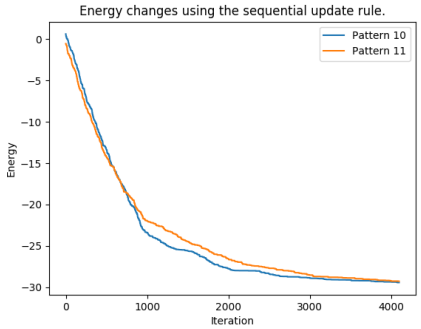


Figure 5: Energy decrease with a random, symmetric, weight matrix drawn from a zero mean gaussian distribution.

3.4 Distortion Resistance

In order to gain a better understanding of the distortion resistance of the 1024 neuron network with weights created from *p1, p2* and *p3*, noise was added to the three patterns and then put into the network again. The noise was implemented by iterating from 0 to 100% in increments of 2%. For all of the three patterns, there seemed to be a cutoff point at just under 50% noise at which the network could recall the original patterns. The results are visualized in *figure 6*.

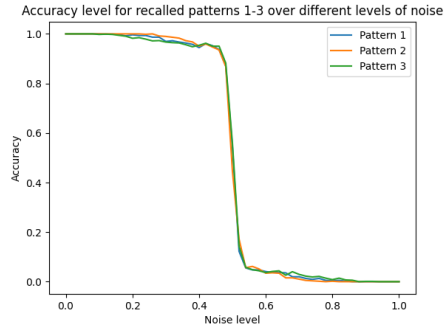


Figure 6: Accuracy for the three different patterns with noise ranging from 0 to 100%.

There seems to be a very minor difference between the distortion resistance for the three patterns. From figure 6, a slightly better performance from pattern *p2* can be seen.

Furthermore, the network always seem to converge to the right attractor if the noise level is below 40-45%. When the noise is above 50-55%, the patterns seem to converge towards the opposite attractor, as can be seen in figure 6, where the accuracy with high levels of noise is zero, hence meaning that all elements are flipped. However, between noise levels of ca 40-45% to 50-55% the network converges to other attractors than either the correct or the opposite attractor. To investigate this we created a test focused on noise-levels between 40% and 60% and plotted the recalled patterns. Which attractor that the network converged to seemed somewhat arbitrary and changed from time to time. Sometimes, noisy pattern 1 converged to pattern 2 or pattern 3, and vice versa, all of which are obvious attractors. We also noticed that pattern 2 and pattern 3 could converge towards pattern 11 and its

opposite which as known is a combination of pattern 2 and pattern 3. These thus seemed to be even more attractors.

Extra iterations beyond a single-step recall did not help the network to perform any better, i.e. when the network converged it did not make any further changes when continuing to update. For this task however, the little model was used. Perhaps the result would be different if sequential updates would have been used instead.

3.5 Capacity

A noise level of 5% and a 100-unit network was used for this task, and both the little model and the asynchronous update scheme was tested. Both performed at an equal level.

When using three patterns, all could be safely stored. However, when adding another fourth pattern, we could see that the recalled patterns looked somewhat like their original but with errors, only spurious recalled patterns were attained. When adding even more patterns in some random order, the network converged towards some even more unrecognized patterns. Adding patterns randomly and looking for the best combinations gave marginally better results, especially when for example pattern 3 and 4 were separated as they look somewhat similar. The drop in performance was abrupt. When having 3 patterns, all were safely stored. However, when we used 4 patterns, none were perfectly recalled, but still pretty much recognizable. Figure 7 showcases the element-wise accuracy when adding more and more patterns to the weight matrix and testing if they can recall the distorted versions of themselves.

	Patterns included in network								
Pattern	1	2	3	4	5	6	7	8	9
1	1,0000	1,0000	1,0000	0,8984	0,8311	0,8047	0,8271	0,8271	0,8213
2	-	1,0000	1,0000	0,8291	0,7842	0,7617	0,7725	0,7725	0,7568
3	-	-	1,0000	0,9512	0,8936	0,9258	0,8936	0,8936	0,8994
4	-	-	-	0,8887	0,8594	0,8662	0,8496	0,8496	0,8555
5	-	-	-	-	0,8252	0,8047	0,8115	0,8115	0,8076
6	-	-	-	-	-	0,8477	0,8408	0,8408	0,8467
7	-	-	-	-	-	-	0,8555	0,8555	0,8496
8	-	-	-	-	-	-	-	0,6221	0,6404
9	-	-	-	-	-	-	-	-	0,7246
Average	1,0000	1,0000	1,0000	0,8918	0,8387	0,8351	0,8358	0,8091	0,8002

Figure 7: Element-wise accuracy when incrementally adding more patterns to the network weight matrix

When learning random patterns instead of the pictures, a lot more moderately distorted patterns could be safely restored, see figure 8. In fact, around 50-60 degraded patterns could be perfectly completed. The big difference in storage capacity for random patterns versus the pictures stems from the fact that there is much more variance and much less correlation in the random patterns. Some of the pictures look fairly similar (for example picture 3 and 4), indicating that their 1024 bit patterns also would display similarities, making it harder for the network.

Intuitively, as more patterns are learned, the number of stable patterns decrease. This however is only the case up until a certain level, where the number of stable patterns start to increase slightly again, see figure 9a. This probably has to do with the self-connections becoming larger and larger as more patterns are added, restricting the network from changing the states of the patterns.

When convergence to the patterns from noisy versions (5% flipped units) was used, the network behaved very differently for large number of patterns, see figure 9b. Suddenly, the network only started to learn spurious patterns instead of getting some of them right and when increasing the noise level even more, the network became even worse. This implies that self-connections along with noisy data promote the formation of spurious patterns and rather have negative effects on the noise removal capabilities of the network, just as explained in the task instructions.

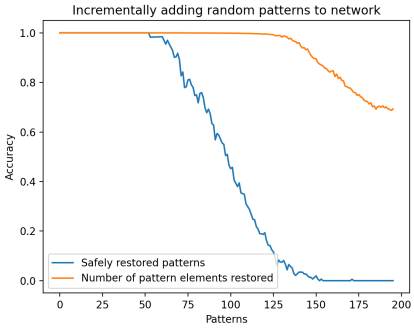
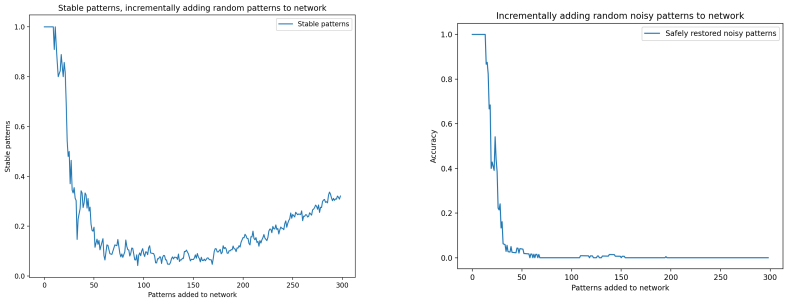


Figure 8: Element-wise and pattern-wise accuracy when incrementally adding more random patterns to the network weight matrix



(a) Stable patterns.

(b) Recall accuracy.

Figure 9: Change of stability and convergence when incrementally adding more random patterns to the network weight matrix.

When removing the self-connections and comparing curves from pure and noisy patterns, the difference went away, and the network performed almost as good on noisy as on noise-free data. Moreover, the network performance did not decrease as much when increasing the amount of noise, clearly showcasing how such a network generalizes better than one with self-connections present. With the self-connections removed, the maximum number of perfectly retrieved patterns were about 13-15 for a 100-unit network, which goes hand in hand with the proven capacity of a Hopfield network being 0.138N.

If the patterns were biased, e.g. contained much more +1 than -1, the network capacity became even worse, and only about 7-8 patterns could be stored. This relates to the capacity results of the picture patterns, since most of the pictures are also unbalanced and biased towards either +1 or -1.

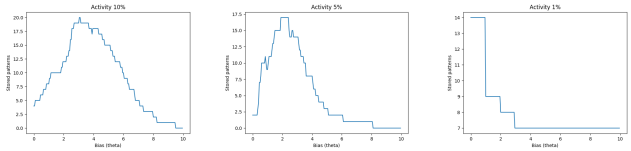
3.6 Sparse Patterns

In order to gain a better knowledge on the storage capcity for networks with sparse patterns, 0 and +1 were used for the patterns as per the instructions. Furthermore, the updated calculations for deriving the weight matrix; $w_{ij} = \sum_{\mu=1}^{P} \left(x_i^\mu - \rho\right)\left(x_j^\mu - \rho\right)$ and the adjusted update formula;

$$x_i \leftarrow 0.5 + 0.5 * \text{sign}\left(\sum_j w_{ij} x_j - \theta\right).$$

was used. In order to observe how many patterns could be stored for different values of θ, the bias, we iterated over θ ranging from 0 to 10 in increments of 0.05. Furthermore, the

network contained 100 units and 300 sparse patterns were created. This rigorous testing of different θ was slow upon which we tested the capacity to perfectly recall 25 of these patterns. The results for $\rho = 10\%, 5\%, 1\%$ is presented in *figure 10*.



(a) $\rho = 10\%$ *(b)* $\rho = 5\%$ *(c)* $\rho = 1\%$

Figure 10: Number of patterns recalled for different $\rho = 10\%$*,* $\rho = 5\%$ *and* $\rho = 1\%$ *using a 100 unit network.*

As can be seen, for an activity level of $\rho = 10\%$, the network could perfectly recall 20 of the patterns when $\theta = 3$. For $\rho = 5\%$, the network could perfectly recall 17 of the 25 patterns for θ ranging between 2 and 2.4. For $\rho = 1\%$, the network storage capacity was even lower and could only perfectly recall 14 out of the 25 patterns at the best θ, which in this case ranged from 0 to 1. From this a general trend can be observed. The higher the activity level, the more patterns can be recalled. Furthermore, for higher activity levels, the more bias is needed to recall as many patterns as possible. For lower activity level, this optimal bias threshold is shifted towards a lower level of bias.

4 Final remarks

This assignment has been really good and helped us gain both a much better overall understanding as a well as a deeper understanding of Hopfield networks. We thoroughly enjoyed the broad nature of the assignment as every as it gave us insights into all aspects of Hopfield networks.