Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 3

Hopfield networks

Boyu Li, Tobias Bezner and Nik Joel Dorndorf

February 17, 2020

1 Main objectives and scope of the assignment

Our major goals in the assignment were to implement Hopfield Networks and explore its

- capabilities,
- capacity and
- limitations

In this assignment we tried to implement all functions on our own, in order to understand the functionality of the ANN. Due to the scope of the tasks, we stepped through them and expanded our knowledge.

2 Methods

We implemented everything in Python. We used PyCharm as scripting environment and Github to share our code. In this assignment we implemented all functions concerning the ANN ourselves and did not use libraries. For this reason, we only used standard libraries like numpy and matplotlib.

3 Results and discussion - Hopfield Networks

3.1 Convergence and attractors

x1d and x3d converge towards stored patterns, but x2d does not converge toward x2.

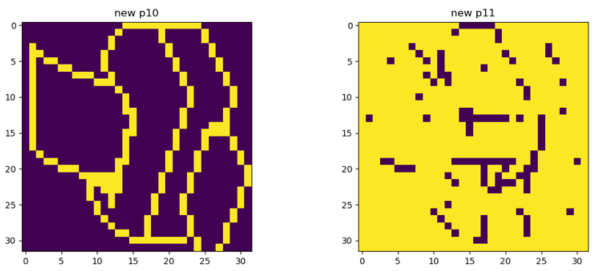


Figure 1: Recalled patterns p10 (left) and p11 (right) with synchronous update

There are 14 attractors in the network.

(1) [-1 -1 -1 -1 -1 1 -1 -1]	(2) [-1 -1 -1 -1 1 -1 -1 -1]
(3) [-1 -1 1 -1 -1 1 -1 1]	(4) [-1 -1 1 -1 1 -1 -1 1]
(5) [-1 -1 1 -1 1 1 -1 1]	(6) [-1 1 -1 -1 -1 1 -1 -1]
(7) [-1 1 1 -1 -1 1 -1 1]	(8) [-1 1 1 -1 1 -1 -1 1]
(9) [1 -1 -1 1 1 -1 1 -1]	(10) [1 1 -1 1 -1 1 1 -1]
(11) [1 1 -1 1 1 -1 1 -1]	(12) [1 1 -1 1 1 1 1 -1]
(13) [1 1 1 1 -1 1 1 1]	(14) [1 1 1 1 1 -1 1 1]

We created three input patterns which have 5 bit errors, compared with x1, x2 and x3. When the starting pattern is very dissimilar to the stored ones, the recalled patterns cannot converge towards stored patterns.

3.2 Sequential Update

The three patterns are stable. For p10, the network can complete a degraded pattern through the synchronous update, but not for p11 (see Figure 1).

For Random Sequential Update, p10 converged to p1 (Figure 2). For Original Sequential Update, p10 converged to a spurious pattern (Figure 3).

3.3 Energy

In this part of the assignment we investigated the convergence of the network and had a look into the structure of the weight matrix. For symmetric connection matrices it is possible to introduce an energy function. In Table 1 you can see the energy levels of the learned pattern P1, P2 and P3, which represent local minimas of the energy function. The distorted patterns are not located in local minimas and step closer to a minima each iteration. In Figure 4 you can see on the left side the energy trend over iterations of sequential updates for P11. The energy is always decreasing and end a stable minima.

It is important to point out that this garanteed convergence is only given for a sym-

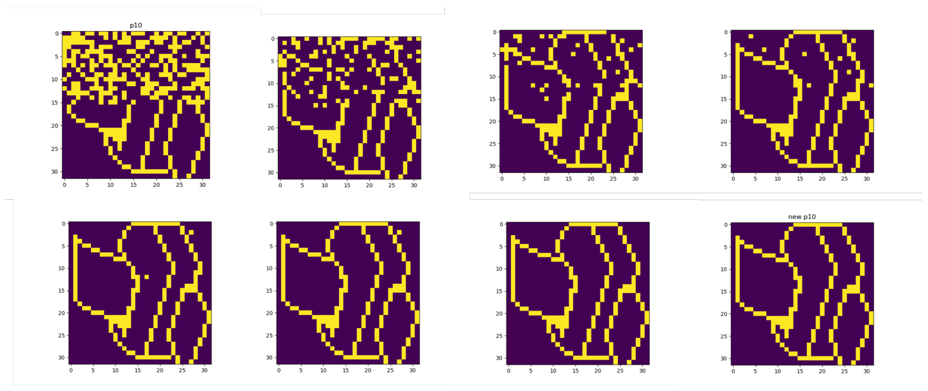


Figure 2: Random Sequential Update for p10 (Every 1024 Steps)

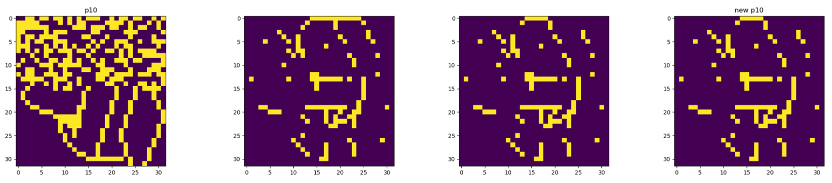


Figure 3: Original Sequential Update for p10 (Every 1024 steps)

metric weight function. With a random initialisation of the weight matrix the energy does not always decrease and oscillates between states (Figure 4). In the case of a random symmetric weight matrix the energy level always decreases with every iteration and will end in a local minimum, which represents an attractor.

3.4 Distortion Resistance

In order to study the network's robustness we added noise to the learned patterns and tried to recover the original clean data. We added noise from 0 to 100% to the learned patterns P1, P2 and P3 and calculated the network outputs. In Figure 5 the results for P1 are shown. For up to 40% of noise converges the network in the right attractor and the original data is recovered. For 50% it detects a wrong pattern and for 60% it ends in a not learned attractor. For 70% noise or higher the network convergences in the inverse pattern of P1 because the inverse patterns are also attractors. The results

Pattern	Energy
p1	-1439.4
p2	-1365.6
p3	-1462.3
p10	-416
p11	-173.5

Table 1: Energy of attractors (p1,p2,p3) and distorted patterns (p10,p11)

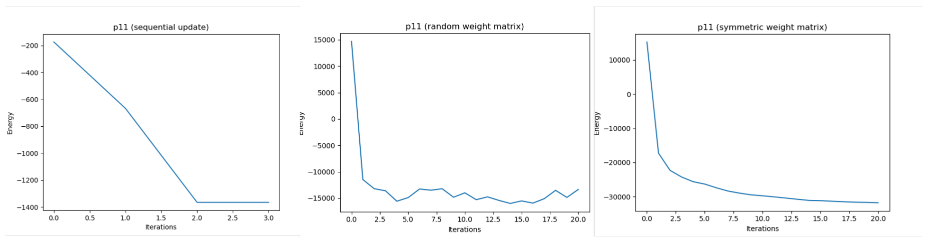
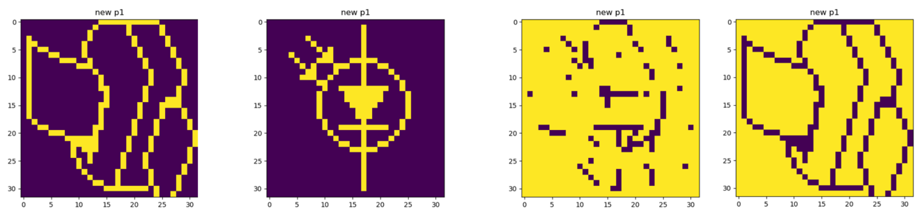


Figure 4: Energy over iterations: Sequential update rule (left), random weight matrix (center) and symmetric weight matrix (right)



Up to 40% of noise

For 50% of noise

For 60% of noise

Over 70% of noise

Figure 5: Distortion Resistance P1

for the other patterns, P2 and P3, behaved in the same way, so we could not find differences depending on the patterns. A higher number of iterations does not avoid detecting the wrong pattern when there is high noise.

3.5 Capacity

If we take p1, p2 and p3, like we have shown before, all of them can be safely stored. By only adding p4 this behaviour changes completely and we cannot store the images anymore. This is probably the case because the images are too similar. We, therefore, decided to not take the images in any order, but to take random images of the first 10 and look at how many of them could be stored for different numbers of patterns and different runs. The results can be seen in Table 2. We can see that even for 3 random patterns only 95% for synchronous and 93% for asynchronous can be safely stored on average. This number slowly decreases with increasing number of patterns. For a Hopfield network trained on 6 patterns less than 5% of them can be safely stored. For random patterns (same size as before) the results are way different and a lot more patterns can be stored. This is the case because they are not as correlated as the images. In Figure 6 one can see, that the maximum number of stable patterns in this case is around 80. At some point the number of stable patterns is decreasing again, this is probably the case because the network is building spurious patterns as attractors because there are too many stored patterns. We can also observe that noise is hurting the performance. This effect decreases when we get rid of the self-connections by removing the diagonal of the weight matrix. Then the impact of the noise is not as strong anymore but the overall performance is decreasing. When we unbalance the random dataset the maximum number of stable pattern goes down again.

Number of Patterns	Synchronous	Asynchronous
3	0.95 (0.047)	0.93 (0.039)
4	0.34 (0.120)	0.34 (0.107)
5	0.09 (0.009)	0.09 (0.009)
6	0.03 (0.004)	0.01 (0.001)

Table 2: Proportion of stable patterns for different amounts (patterns randomly selected, mean over 20 runs, variance in brackets)

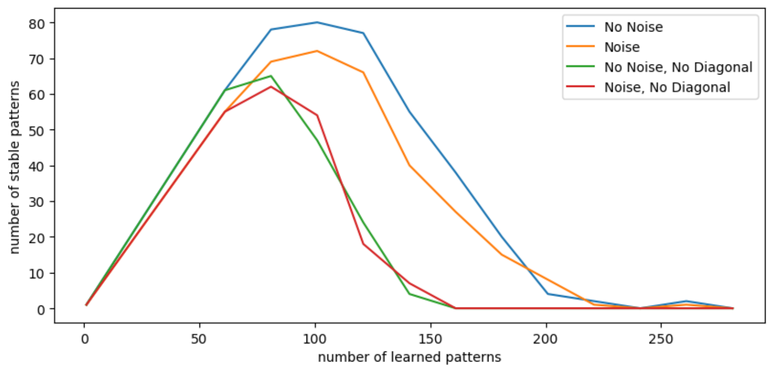


Figure 6: Number of stable patterns over number of learned patterns for different setups

3.6 Sparse Patterns

The results for different sparsities can be seen in Figure 7. We can see that the best bias is decreasing when the patterns get sparser. This method for sparse patterns with the bias seems to work quite well since nearly all of the 300 patterns can be safely stored at the best bias value. Anyways, one has to find a good value for the bias because otherwise none of the patterns can be stored. For the sparsest patterns one can see a strange behaviour because the number of stored patterns is never going down to zero. This is probably the case because some patterns are only consisting of zeros.

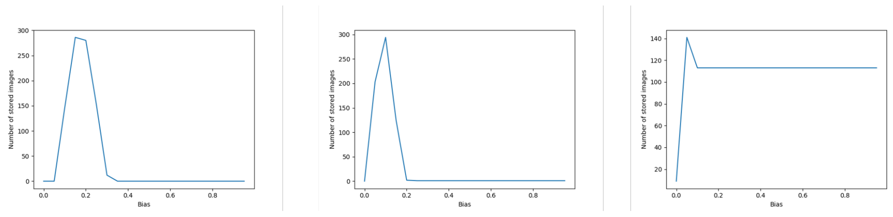


Figure 7: Maximum number of stored patterns for different activations over different bias values

4 Final remarks

What is most noticeable is probably how quickly the updates have converged. In most cases this was already the case after one step and you couldn't really see a continuous change. In general, Hopfield networks seem to be very unstable models where you have to pay a lot of attention to the correct parameterization. What is also interesting to see is that often the patterns were inverse attractors in addition to the learned patterns, which makes sense since they are orthogonal.