Short report - Lab assignment 3

Hopfield Networks

Adrian Campoy Rodriguez, Nimara Doumitrou-Daniil and Gustavo T. D. Beck

February 17, 2020

1 Main objectives and scope of the assignment

Our major goals in the assignment were:(1) explain the principles underlying the operation and functionality of auto-associative networks; (2) train the Hopfield network; (3) explain the attractor dynamics of Hopfield networks and the concept of energy function; (4) demonstrate how auto-associative networks can do pattern completion and noise reduction; (5) investigate their storage capacity.

2 Methods

During this project the programming language Python 3 was used in three different IDEs such as Pycharm, VSCode and Spyder. Some particular toolboxes were used for specific tasks. For instance, matplotlib was used as an additional support to visualize the images and the generated patterns. Finally, the GitHub repository was used to store and manage different versions and pieces of code.

3 Results and discussion

3.1 Convergence and attractors

For the first task of this assignment, we were asked to train a Hopfield Network with the following binary sequences (patterns) composed by 8 units: $x_1 = [-1, -1, 1, -1, 1, -1, -1, 1]$; $x_2 = [-1, -1, -1, -1, -1, 1, -1, -1]$; $x_3 = [-1, 1, 1, -1, -1, 1, -1, 1]$. Afterward, in order to test the memory of the network, we were asked to apply the update rule repeatedly upon three binary distorted sequences of the original patterns (table 1) and check if the outputs corresponded to one of the patterns learned by the network. Otherwise, it would mean that the model fitting got stuck in a different local minima (different attractor than the original patterns).

	x_1	x_2	x_3
Original patterns	[-1, -1, 1, -1, 1, -1, -1, 1]	[-1, -1, -1, -1, -1, 1, -1, -1]	[-1, 1, 1, -1, -1, 1, -1, 1]
Input patterns	[1, -1, 1, -1, 1, -1, -1, 1]	[1, 1, -1, -1, -1, 1, -1, -1]	[1, 1, 1, -1, 1, 1, -1, 1]
Output patterns	[-1, -1, 1, -1, 1, -1, -1, 1]	[-1, 1, -1, -1, -1, 1, -1, -1]	[-1, 1, 1, -1, -1, 1, -1, 1]

Table 1: Comparison between converged patterns and original patterns.

Table 1 shows that not all of the input patterns (e.g. x_2 input with two bit errors) converged to one of the original patterns. This means that they got stuck in a local minima, i.e. another attractor (spurious patterns). To analyze the number of attractors we generated all the 256 binary sequences with 8 units and utilized the trained model to check in which local minima these 256 sequences would converge to. This analysis produced 14 attractors, one of which was the output of the distorted x_2 (table 1). For comparison, if we discard self connections ($w_{ii} = 0$),

as they promote the formation of spurious patterns (see 3.5), we noticed that the number of attractors get reduced to 6. In addition, by eliminating the self-connections, the distorted x_2 input converges to x_2, since the previous local minima (spurious patterns) no longer exists. Finally, if we increase the number of bit errors to 50% (e.g. $x_{distorted} = [1, 1, -1, 1, -1, -1, 1, 1]$) we noticed that it doesn't converge to any of the original patterns.

3.2 Sequential update

This section encouraged us to work with more complex networks. Unlike the previous task, where only 8 units were used, now 1024 units were required in order to work with 32x32 images. First, the three patterns shown in figures 1, 2 and 3 were learnt.

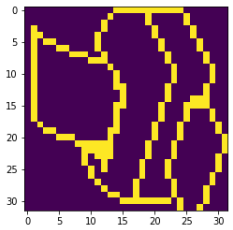

Figure 1: Pattern *p1* to be learnt

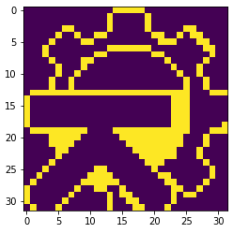

Figure 2: Pattern *p2* to be learnt

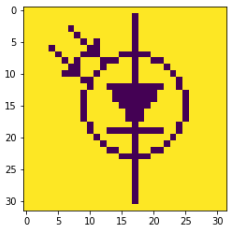

Figure 3: Pattern *p3* to be learnt

After fitting the network with these three patterns, we examined their stability as follows: *p1*, *p2* and *p3* were respectively used as an input to check that their output converged to themselves (i.e. the output pattern was equal to the input pattern). The three patterns converged as expected and were thus considered stable.
After ensuring that learnt patterns were stable, we used the network to recover two degraded patterns. In particular we tried to recover pattern *p1* from a degraded version of it (called *p10*), as well as recovering either pattern *p2* or *p3* from a mixture of both (called pattern *p11*). Finally, *p10* converged to an attractor different from the goal as it can be seen in figure, whereas *p11* converged to one of the desired patterns. The results are shown in figures 4, 5, 6 and 7.

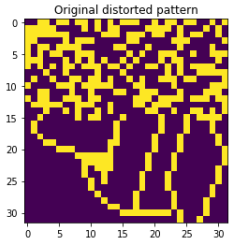

Figure 4: Original distorted pattern *p10*

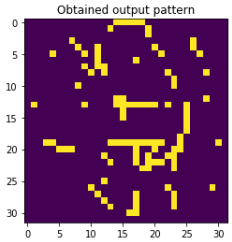

Figure 5: Converged pattern with *p10* as input

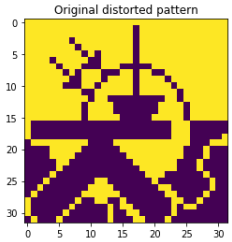

Figure 6: Original distorted pattern *p11*

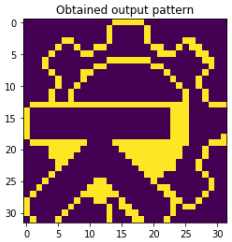

Figure 7: Converged pattern with *p11* as input

So far, units were updated in order. However, it is also interesting to observe the behaviour of the networks and their convergence when the units were randomly updated. Working with the same distorted patterns as before, the results were those showed in figures 8, 9, 10 and 11.

As it can be seen in figures 8 and 9 by randomly updating units it was possible to reach the desired attractor, in this case, pattern *p1*. Moreover, pattern *p11* needed one update of all the units (1024 updates) when ordered update was applied however in this case 1424 updates were required in order to converge to a desired attractor. It is worth noticing that in this case *p3* was recovered from *p11* unlike in the ordered update (when *p2* was recovered). Several experiments were performed with different random combinations of units updates. The obtained attractor depended on the random order of updates, sometimes obtained one of the expected attractors and sometimes other undesired attractors. Therefore, it can be said that random update of units can help on the convergence of certain patterns to the expected attractors and not any local minimum, as in the case of pattern *p10*. From our observations, we consider that randomly

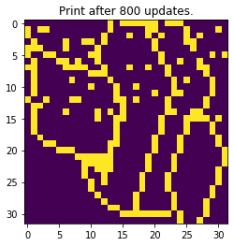

Figure 8: Convergence of pattern *p10* after 800 updates of a random unit

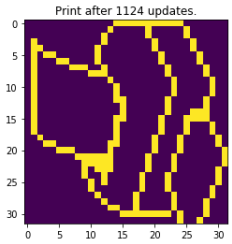

Figure 9: Convergence of pattern *p10* after 1124 updates of a random unit

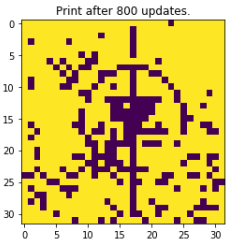

Figure 10: Convergence of pattern *p11* after 800 updates of a random unit

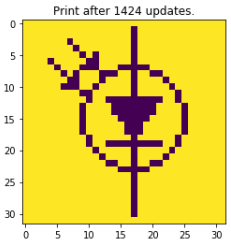

Figure 11: Convergence of pattern *p11* after 1424 updates of a random unit

picking inputs to update help escaping local minima. Nevertheless, it also makes convergence slower, as in pattern *p11*.

3.3 Energy

For networks with a symmetric connection matrix, it is possible to define an energy function such as the Lyapnov function. Using this fucntion, we observed the decrease in energy as a pattern is modified towards an attractor. The proposed function in this task has been $E = -\sum_i \sum_j w_{ij} x_i x_j$. First, the energy of the different attractors (*p1*, *p2* and *p3*) and distorted patterns (*p10* and *p11*) of the previous task was calculated. The results are:

	p1	*p2*	*p3*	*p10*	*p11*
Energy	-1439.39	-1365.64	-1462.25	-415.98	-173.5

Table 2: Energy of attractors (*p1*, *p2* and *p3*) and distorted patterns (*p10*, *p11*)

From table 2 it can be clearly observed how the energy is significantly lower in the attractors compared to the distorted patterns. This energy decrease was followed during the update of the distorted patterns, as it can be observed in figures 12 and 13.

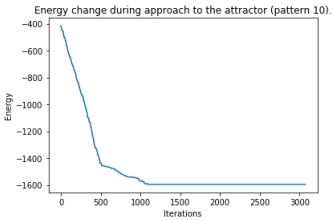

Figure 12: Decrease in energy as iterations increase approaching to attractors) for pattern *p10*

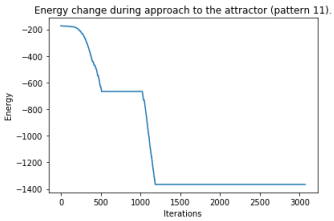

Figure 13: Decrease in energy as iterations increase approaching to attractors) for pattern *p11*

One can also notice how the patterns, in some cases, remain unaltered (unchanged bits), yielding constant energy, inducing a step shape energy figure, as can be clearly seen in 13.
Then, randomly initialized weights were created in order to test the behaviour of the network. The results indicated that the network would cycle between different states with similar energy levels as it can be observed in figures 14 and 15.

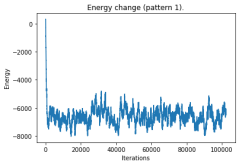

Figure 14: Energy during iterations for pattern *p1* (with non-symmetric random weights)

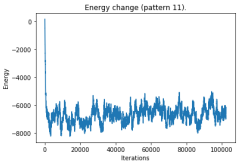

Figure 15: Energy during iterations for pattern *p11* (with non-symmetric random weights)

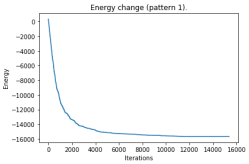

Figure 16: Energy during iterations for pattern *p1* (with symmetric random weights)

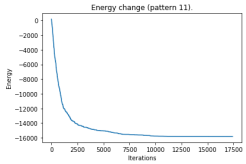

Figure 17: Energy during iterations for pattern *p11* (with symmetric random weights)

Finally, the same weight matrix was made symmetric in order to observe the behaviour of the network. We did not recover the expected attractors but we obtained distorted images as an output. This behaviour was however expected since the weights do not correspond to those of the trained network. Furthermore, as seen in figures 16 and 17, the network no longer oscillates between states, due to the symmetry of the weight matrix. Lastly, the network exhibits a low convergence rate, as it takes too long to converge to a specific pattern and keeps slightly changing over iterations.

3.4 Distortion Resistance

Possibly the most crucial application of a Hopfield Network is as a denoiser. That is, given a distorted input of one of its memorized patterns, it should be able to filter through the noise and retrieve the original pattern-attractor. It is therefore, of great interest, to examine this property, and analyze their noise robustness.
For this reason, we trained our network by memorizing patterns p1, p2 and p3 (as in 3.2) and iteratively incremented the noise from 1% to 100%, observing whether or not the distorted images of p1, p2 and p3 would converge to their pure counterparts. Our experiments showed that the noise resistance of our inputs were $[49\%, 40\%, 42\%]$, as can be seen in the following graph:

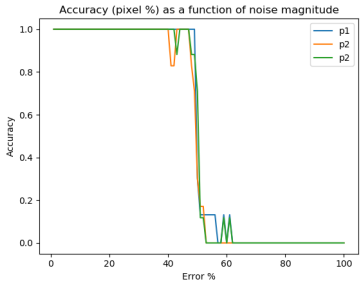

Figure 18: Accuracy (% of pixels) as a function of the error magnitude. Incrementing the percentage eventually leads the network to instability, as it converges to spurious memories.

Interestingly, altering the pattern too drastically, made it converge to a flipped spurious pattern:

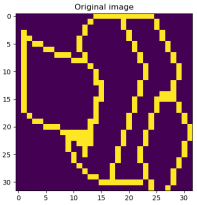

Figure 19: Original Image-pattern p_1

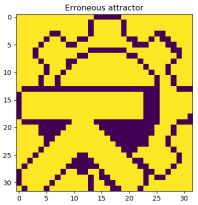

Figure 20: Erroneous pattern retrieved for 50% noise (flipped p2).

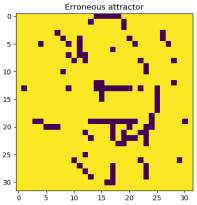

Figure 21: Erroneous pattern retrieved for $\sim 51 + \%$ noise.

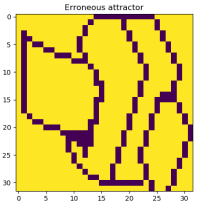

Figure 22: Erroneous pattern retrieved for almost 100% noise.

Overall, after a certain error threshold, the network converges to spurious attractors. Interestingly, the three patterns do not share exactly the same threshold (even though they are

quite similar). This is likely due to their distance from the nearest spurious attractor, implying that p_2 and p_3 are more unstable, as they lie closer to a spurious one. Finally, because we utilized the synchronous update, we are unable to escape such local minima and thus, extra iterations do not yield any improvements in this setting (update is fully deterministic: if $x_{t+1} = x_t \Rightarrow x_{t+2} = \text{sign}(Wx_{t+1}) = \text{sign}(Wx_t) = x_{t+1}$).

3.5 Capacity

The capacity of a Hopfield Network is equally important and is often the main point of criticism for such networks. Its main capacity limitation stems from one of its core assumptions: the input-patterns are **orthogonal**. As one continuously introduces new patterns in his network, such an assumption becomes increasingly unlikely, as these correlated (overlapping) inputs give rise to spurious patterns. In fact, our database was such that when introducing the fourth picture $p4$ during training, our network was incapable of retrieving a moderately noised variation of p_1, p_2, p_3 (15%). More importantly, it failed to retrieve them even in the absence of noise. As we continued adding more and more pictures from our database, its accuracy **gradually decreased**. On the contrary, when adding random images, our network's capacity was more robust, as our newly added samples would be on average close to orthogonal with our initial pictures p_1, p_2, p_3, since these random images have $|+1| \simeq |-1|$. In fact, if we only memorize random images, then we expect to be able to effectively memorize roughly $0.138N \simeq 141$ images (as random images are initially, on average, mutually orthogonal).

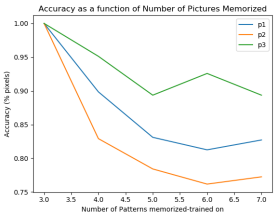

Figure 23: Accuracy when incrementally adding pictures from our database

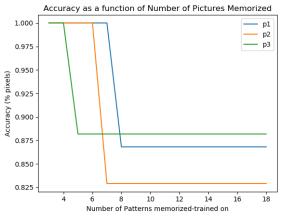

Figure 24: Accuracy when incrementally adding random images.

To better assess the network's capacity, we constructed the following toy example, where we utilized 300 random 100-dimensional training data, and examined its accuracy as a function of the training data size. We noticed that the network's performance drastically plummeted at around $0.138N$.

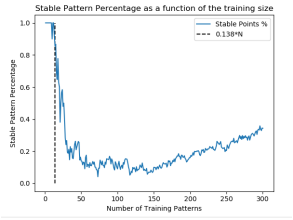

Figure 25: Accuracy on non-noisy data.

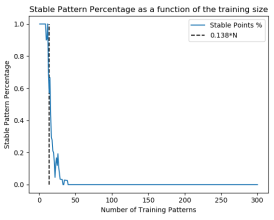

Figure 26: Accuracy on noisy data

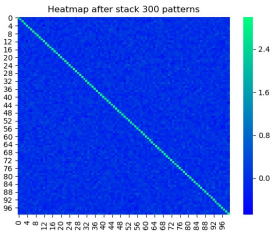

Figure 27: Weight Heat-Map. Notice how $w_{i,i} >> w_{i,j}$.

Interestingly, the drop in performance was more abrupt for the noisy data. This is mostly because of the existence of self connections ($w_{i,i} > 0$), which gradually increase, as more and more data are memorized. Furthermore, recall that $x_i = \text{sign}(\sum_j w_{i.j} x_j) \simeq \text{sign}(x_i)$ for large positive $w_{i,i}$. Our network is thus, rigid and overfits on our data, as each bit x_i has the tendency to remain unaltered. This explains the difference between the two graphs for large training data size. If one forces $w_{i.i} = 0$, then this difference fades away. In fact, after imposing this regularization technique, our network is capable of efficiently memorizing 10 patterns, which is marginally below the expected 13.8 patterns.
Lastly, all of these toy experiments were conducted while sampling 100 dimensional vectors with

roughly $|+1| \simeq |-1|$. This is, in general, unrealistic, as real world data do not exhibit such symmetries. To test our network's robustness to such symmetries, we sampled random vectors with a bias, such that $\frac{|+1|}{|-1|} \simeq \frac{9}{1}$. Our network's performance vastly decreased, as it was only capable of learning 2 patterns.

3.6 Sparse Patterns

As we saw in the previous section, the Hopfield network struggles in processing patterns with a disproportionate amount of $|+1|$ and $|-1|$. Similarly, as the numbers 1 and -1 are mostly utilized for convention and numerical conveniance, one can imagine that this network also struggles with patterns in $\{0,1\}^N$, such that $|0| >> |1|$ (very few 1). We call such patterns **sparse**, as their active features (those that showcase 1) are disproportionally few in comparison with the inactive ones. We can thus quantify the average activity ρ of our training data $X^{train} \in \{0,1\}^{P \times N}$, as $\rho = \frac{1}{|X^{train}|} \sum_{p,n} X^{train}_{p,n}$.
Thus, before training our network, we first transform our data by subtracting ρ from each feature (normalization), generating weights close to zero (as in previous section). However, to account for the imbalance in the data, we also need to add a **bias** θ, which should incentivize our features towards 0, pushing them towards sparsity. We alter our update rule such that $x_i = 0.5 + 0.5 \cdot \text{sign}(\sum_j w_{i.j} \cdot x_j - \theta)$. Thus, to incentivize sparsity, we wish to push x_i towards 0, hence $\text{sign}(\sum_j w_{i.j} \cdot x_j - \theta) \to -1 \iff \theta$ large positive number. Note that, if we make θ too large, then we will push every feature towards zero, which is undesirable. Thus, a fine balance must be found ($\theta = \theta(\rho)$). In fact, we expect that $\sum_j w_{i,j} \cdot x_j$ decreases as the data becomes sparser, since fewer bits x_i are non zero. Thus, for sparser data, the optimum θ_{opt} is smaller, because it becomes easier to push every bit to zero. Our experiments reinforce our previous claims:

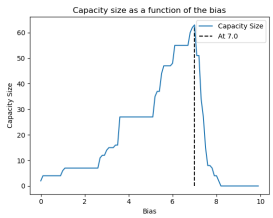

Figure 28: Activity 0.1.

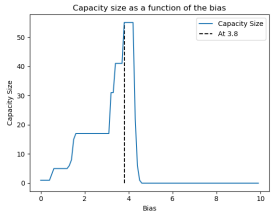

Figure 29: Activity 0.05

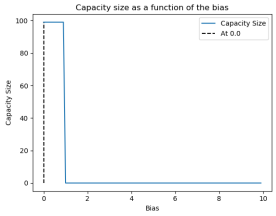

Figure 30: Activity 0.01

Note that in Figure 30, we restricted our sample size to 100, as only 100 unique samples can be generated with sparsity $\rho = 0.01$ (only one bit is 1). Finally, the network exhibits impressive capacity, since for $\rho = 0.01$ we have $w_{i,j} \simeq \sum_p x^p_i x^p_j = \delta_{i,j}$ and thus $x_i \simeq 0.5 + 0.5 \cdot \text{sign}(\sum_j \delta_{i.j} \cdot x_j - 0) = 0.5 + 0.5 \cdot \text{sign}(x_i) = x_i$. Notice how for $\theta \geq 1 \geq x_i \Rightarrow x_i \simeq 0.5 + 0.5 \cdot \text{sign}(x_i - \theta) = 0$, resulting in zero capacity.

4 Final remarks

The present assignment promoted the comprehension of auto-associative networks and pattern recognition. In retrospect, we would had appreciated if we had scratched pre-processing techniques of images before the learning and fitting steps, since most of real-life applications require some kind of pre-processing. On the other hand, one positive aspect of this assignment relies on the fact that most of the analyses required visualization to fully comprehend the outcomes, which can be a powerful skill for data scientists. Once more, it served as a good exercise to become familiar with the scientific and concise format of reports that is usually requested in the workplace.

In terms of the application of Hopfield Networks, the assignment showcased the relevancy of the orthogonality factor between training patterns, as it drastically affects the performance of the method and the number of spurious patterns. This factor greatly hinders its memory capacity, especially on low dimensional data or real life data exhibiting multiple overlapping features.