Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 3

Hopfield Networks

QIU Danny, SAMAILLE Noe

September 30, 2019

1 Main objectives and scope of the assignment

- Study the Hopfield network **addressable memory**, its **attractors** and the circonstances leading to **spurious patterns**.
- Evaluate the storage **capacity** and stability depending on the distribution of patterns, and resistance to **distortion**.

2 Methods

The programming language used for this lab is **Python**, with **numpy** library for handling matrix computation.

3 Results and discussion - Hopfield Networks

3.1 Convergence and attractors

The learned patterns `x1`, `x2` and `x3` are perfectly recalled so they are indeed attractors in this network.

With *Little model* update, only `x1d` with one bit error converged towards stored pattern `x1`. The two other distorted patterns (with two bits error) alternate between two different states with the same energy.

With asynchronous update, all distorted patterns converge, but the recall is not always perfect.

- with one bit error, `xd1` always converges towards `x1`

Pattern	Perfect recall rate	Standard deviation
x1d	1.0	0.0
x2d	0.48	0.50
x3d	0.55	0.50

Table 1: Rate of perfect recall based on 100 recalls

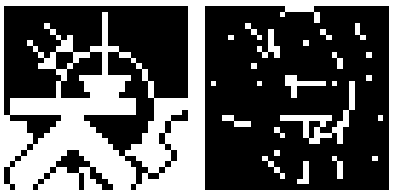
- because asynchronous updates has a part of randomness, with two errors, there is 50% chance xd2 and xd3 converge towards a different pattern.

There are 6 attractors (the stored patterns and their "opposit", where the -1 and 1 are reversed).

When the starting pattern is even more dissimilar, it often converges to a different attractor and eventually to the opposit attractor.

3.2 Sequential update

The first three patterns are stable. The network can complete the first distorted pattern (degraded version of p1) but not the second one, which is a mixture of p2 and p3. Figure 1 shows the output is a spurious pattern, combination of the three memorized ones.



(a) p11 (b) p11 recall

Figure 1: p11 input and output from the network

If we select the units randomly, each neuron fires or not sequentially based on the values of all the other neurons. It can takes one or two steps (each with 1024 activation updates) for the network to settle to a steady state.

3.3 Energy

The energy function is: $E = -\sum_i \sum_j w_{ij} x_i x_j$. *Table.2* shows the energy at the memorized attractors and distorted patterns.

When weights are initialized following a zero-mean normal distribution ($\forall i, j, w_{ij} \sim \mathcal{N}(\mu = 0, \sigma = 1)$), the network doesn't converge. The energy is unstable, does not decrease nor increase iteration after iteration.

Point	Energy
p1	-1436.39
p2	-1362.64
p3	-1459.25
p10	-412.98
p11	-170.5
p10 (after recall)	-1436.39
p11 (after recall)	-1593.01

Table 2: Energy measured at different points

However, when the matrix of random weights is symmetric, energy decreases iteration after iteration, but in a much slower way. On a fifty runs, it took about 4.8 (std 1.0) iterations to converge.

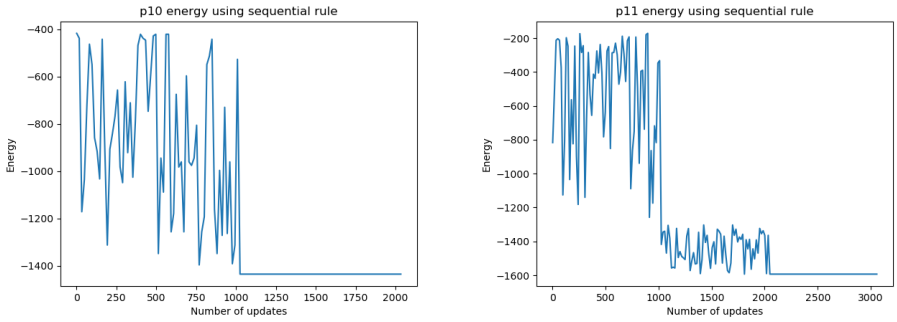
This can be explained by the formula of the energy. Let N be the size of the pattern, W the weight matrix, H be the energy at one iteration and H' the energy after the node s_l ($l \in \{1, ..., N\}$) updates. Then,

$$H - H' = -(s_l - s'_l) \sum_{i=1}^{N} w_{i,l} s_i - (s_l - s'_l) \sum_{i=1}^{N} w_{l,i} s_i$$

In symmetric matrices, $w_{i,l} = w_{l,i}$, thus,

$$H - H' = -2(s_l - s'_l) \sum_{i=1}^{N} w_{i,l} s_i$$

$s_l - s'_l$ is of s_l sign, s_l and $\sum_{i=1}^{N} w_{i,l} s_i$ are of opposite signs because s_l was updated. As a consequence, $H - H'$ is positive, which means $H' < H$. With non symmetric matrices, we cannot group the two sums and the sign of the difference is not guaranteed to be positive.



(a) `p10` energy changes

(b) `p11` energy changes

Figure 2: `p10` and `p11` energy depending on updates

Figure.2 shows it only takes one or two iterations to converge. Inside a iteration, the energy does not decrease regularly, instead it varies greatly.

3.4 Distortion resistance

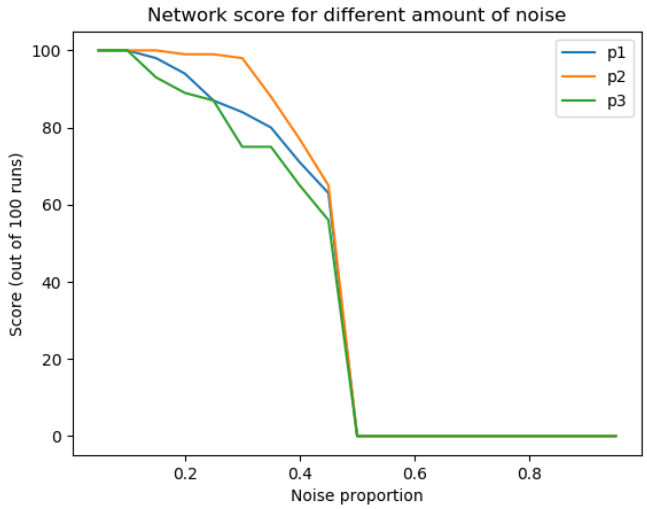


Figure 3: Score of the Network for different noise proportions

Figure.3 shows that the network can remove noise almost every time up to 15% noise for *p1* and *p3* and up to 30% for *p2* (so *p2* seems a better attractor in terms of noise tolerance). Beyond 60% noise the network starts to settle to the "opposit" attractor.

For a noise proportion of 50% the network always settles to a spurious pattern or a wrong attractor, which seems normal as the pattern does not look like the original one at all.

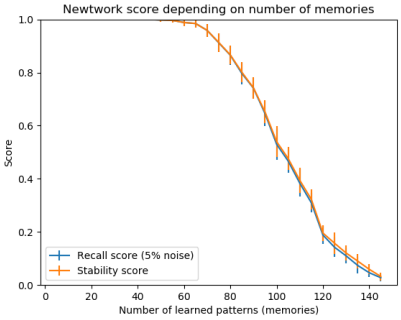
3.5 Capacity

When we added `p4`, none of the stored patterns were recognized: the outputs were spurious patterns. This can be explained by the closeness of `p3` and `p4`.

The drop in performance is abrupt from 3 images to 4. Adding `p5`, `p6` and `p7` separately to `p1`, `p2`, and `p3` gives us distorted output but we can still recognize the stored patterns.

With five images, only one or two of them could be safely stored but were not stable, the rest producing spurious patterns.

The best combination of images for memorization that was found (at the moment) was `p2`, `p5`, `p8`, `p9`. Distorted patterns (up to 40% of noise) are perfectly recognized.



(a) W without diagonal coefficients

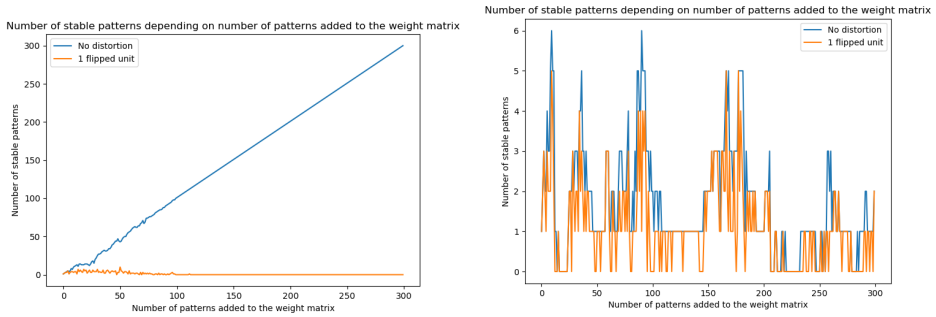
Figure 4: Capacity of the network depending on the number of memories (stored patterns)

Figure.4 shows the capacity of the network to recall reasonably noisy patterns (5% noise) depending on the number of stored patterns, which are randomly distributed. It clearly shows that the network can store a lot more random patterns than images (100% accuracy up to 60 random patterns in memory).

Figure.4 also shows that beyond 60 random memories all of them are not stable anymore.

The huge difference between random patterns and the pictures is due to the fact that images are very close to each other in the input space (for example *p3* and *p4* which are both really similar images), whereas we don't have these similarities between random uniformly distributed patterns.

With a network of 100 units and 300 patterns, we compared the number of stable patterns memorized by the weight matrix with and without its diagonal.



(a) W with diagonal coefficients

(b) W without diagonal coefficients

Figure 5: Number of stable patterns depending on the weight matrix and stored patterns.

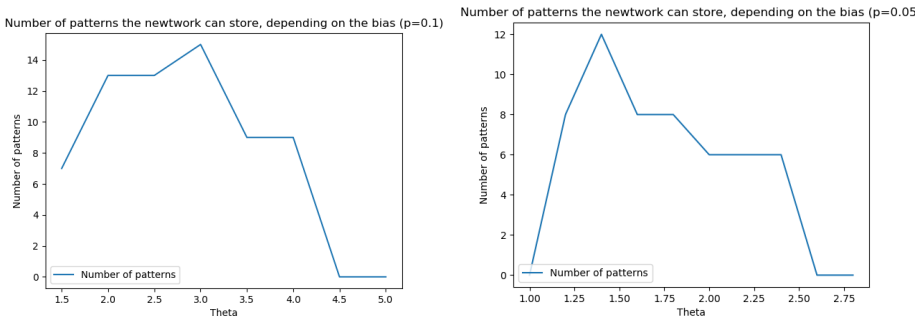
1. With non-zero values on diagonals, all the patterns are stable, especially

when the number of learned patterns is high. Taking a look at the matrix, we see the diagonal is more than 5 times higher than other coefficients and causes inputs to remain the same state. As a result, learning is very sensitive to distortions.

2. Without a diagonal, the network has fewer stable patterns and the number of learned patterns does not influence the memory. However distorted inputs of stable patterns are recognised.

When patterns are biased towards 1, only the pattern full of ones is stable. This relates to the picture patterns which are also biased (either dominated by 1 or by -1). When 5 or more pictures with same dominant color are stored, the output is always the same spurious pattern which is almost completely of that dominant color.

3.6 Sparse patterns



(a) 10% activity ($\rho = 0.1$)

(b) 5% activity ($\rho = 0.05$)

Figure 6: Number of patterns the Network can store depending on bias θ)

Figure.6 shows that bias θ has a great influence on the network capacity. For 10% activity ($\rho = 0.1$), $\theta = 3$ allows the network to store up to 15 patterns. For even sparser patterns ($\rho = 0.05$, right figure), $\theta = 1.4$ allows the network to store up to 12 patterns. The smaller the activation, the smaller the bias needed.

4 Final remarks

Hopfield networks are useful at storing patterns and denoising them (up to 50% distortions). However, patterns of interest (pictures, letters...) are often biased and limit the storage capacity, easily producing spurious patterns. This capacity reduction can be solved by adding a bias to the network.