MSc Machine Learning

DD2437 Artificial Neural Networks and Deep Architectures

Lab3 Report: Hopfield Networks

Authors:

Alexandros Ferles
ferles@kth.se

Pantelis Myriokefalitakis
pmy@kth.se

George Zervakis
zervakis@kth.se

Professor:

Pawel Herman
Assistant Professor

Contents

3 Tasks and questions **2**
3.1 Convergence and attractors 2
3.2 Sequential Update 2
3.3 Energy 5
3.4 Distortion Resistance 7
3.5 Capacity 7
3.6 Sparse Patterns 11

3 Tasks and questions

3.1 Convergence and attractors

Apply the update rule repeatedly until you reach a stable fixed point. Did all the patterns converge towards stored patterns?

The patterns, managed to converge towards the stored patterns after 2 updates.

How many attractors are there in this network? Hint: automate the searching.

All three stored patterns are *attractors* in the network.

What happens when you make the starting pattern even more dissimilar to the stored ones (e.g. more than half is wrong)?

In comparison with the performance of the network in the former case, we get results of lower quality. In addition, the algorithm does not seem to behave in a consistent manner e.g. sometimes it slightly recovers some patterns while damaging the rest, or it only does extra damage to the patters e.t.c.

3.2 Sequential Update

Can the network complete a degraded pattern? Try the pattern p10, which is a degraded version of p1, or p11 which is a mixture of p2 and p3.

For *p1-p10*, the network manages to fully complete the distorted pattern after 1 update of the neurons.

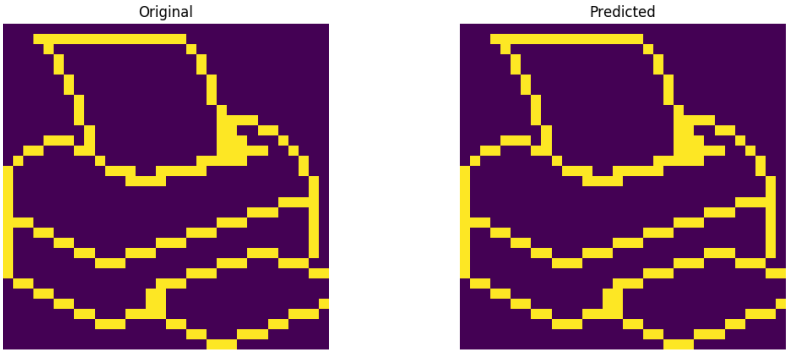

Figure 1: Original pattern *p1*(left) along with the predicted output of the network(right)

For *p2,p3-p11*, the network couldn't approximate at all neither *p2* nor *p3*. Instead, it looks like the network, is trying to "mimic" the inverse of pattern *p1*; predicted output has opposite pixel colors in place of the original.

Figure 2: Original pattern *p2*(left) along with the predicted output of the network(right)

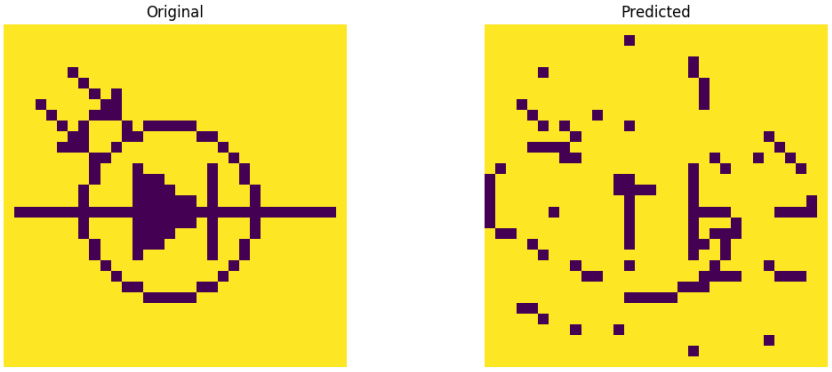

Figure 3: Original pattern *p3*(left) along with the predicted output of the network(right)

Clearly convergence is practically instantaneous. What happens if we select units randomly? Please calculate their new state and then repeat the process in the spirit of the original sequential Hopeld dynamics. Please emonstrate the image every hundredth iteration or so.

For *p1-p10* the network manages to fully complete the distorted pattern like before, after 1 update of the neurons, while for *p2,p3-p11*, it provides a good approximation of *p3* but it forgets about *p2*.

Illustration of the image after 100 iterations

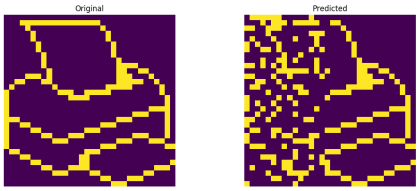

Illustration of the image after 500 iterations

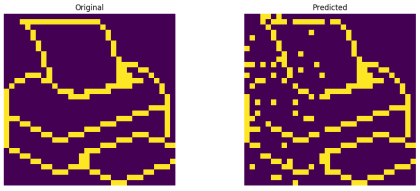

Illustration of the image after 800 iterations

Illustration of the final image (1024 iterations)

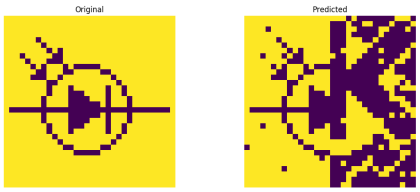

Illustration of the image after 100 iterations

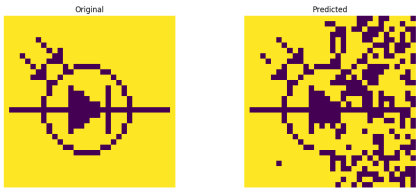

Illustration of the image after 500 iterations

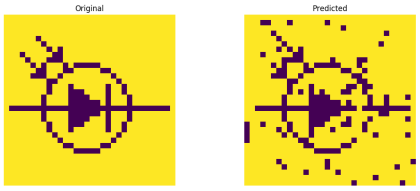

Illustration of the image after 1000 iterations

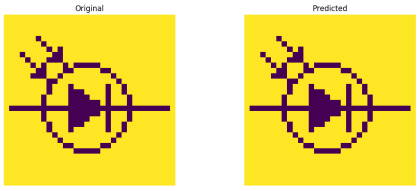

Illustration of the final image (2048 iterations)

3.3 Energy

What is the energy at the different attractors?

Attractors	#Energy
p1	-719.7
p2	-682.8
p3	-731.125

What is the energy at the points of the distorted patterns?

Distorted patterns	#Energy
p10	-719.7
p11	-731.125

Follow how the energy changes from iteration to iteration when you use the sequential update rule to approach an attractor.

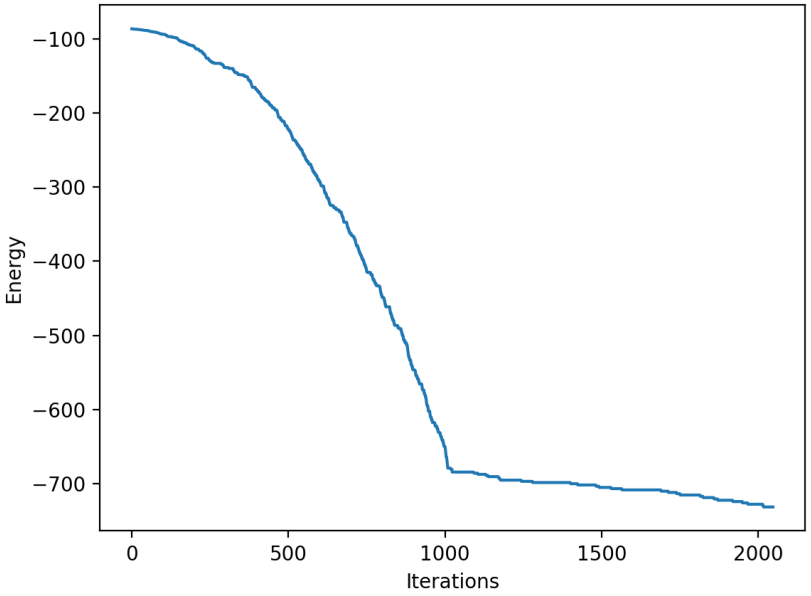

Figure 4: Energy evolution of p11 when trying to approach the attractor p3 using the sequential update rule.

Generate a weight matrix by setting the weights to normally distributed random numbers, and try iterating an arbitrary starting state. What happens?

In this case, the network is unable to restore any pattern.

Make the weight matrix symmetric (e.g. by setting $w = 0.5 * (w + w')$). What happens now? Why?

Setting the weight matrix to be symmetric, we observe no difference in the performance of the network, compared to the previous case. However, here the network managed to converge to a certain state after some iterations. (Hint: mis-match activation of neurons - weights that should be learnt)

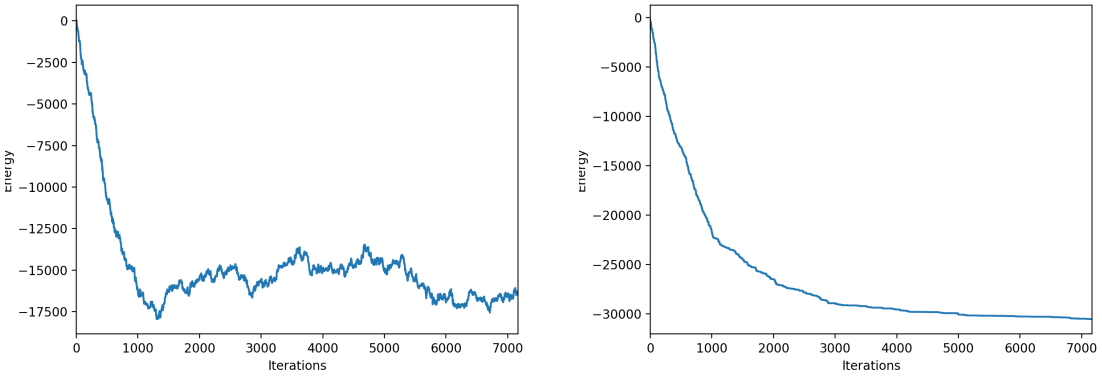

Figure 5: Evolution of the energy function for a randomly generated weight matrix (left) and for a symmetrix matrix (right)

3.4 Distortion Resistance

How much noise can be removed?

Up to 50% of noise can be removed. In fact, if half plus one of the samples are correct with regard to a stored pattern, the Hopfield network will manage to restore the noisy pattern**?**.

Is there any difference between the three attractors?

There is no difference between the three attractors, which follow the same behavior in relation with the amount of noise in the input data:

- For noisy data between 0% and 50%, the distorted pattern is restored to its original source.
- For noisy data between 50% and 75% the network cannot decide to which pattern it will covnverge, and as a result we get a random stored pattern in the output, not necessarily the correct one.
- For a noisy data percentage larger than 75%, the output pattern is the inverse pattern from which the distorted input was derived. This in fact, is also an attractor**?**

Does the network always converge to the right attractor? Do the extra iterations (beyond a single-step recall) help?

As we have already stated in the previous question, for a noisy data percentage greater than 50%, the network cannot converge to the right attractor.

3.5 Capacity

How many patterns could safely be stored? Was the drop in performance gradual or abrupt?

No more than 3 patterns could be safely stored. With the insertion of any 4th pattern from the available ones, we observed an abrupt drop in the performance

Try to repeat this with learning a few random patterns instead of the pictures and see if you can store more.

This time, by learning randomised patterns of -1's and 1's instead of the pictures, we observe a significantly better performance than the previous case. In fact, a number of approximately 130 patterns could be stored efficiently.

It has been shown that the capacity of a Hopfield network is around $0.138N$*. How do you explain the difference between random patterns and the pictures?*

With $N = 1024$, the capacity rule lets us know that the number of patterns that can be stored could reach the value of 141. As we have already noted, we managed to store approximately 130 patterns which is pretty close to this (theoretically) optimal value.

The reason why we managed to increase the performance in comparison with the available patterns from the *pic.dat* file, is because the randomised patterns have a much smaller deviation between themselves than the stored patterns. As a result, it i much easier to store and restore pretty similar patterns to each others, rather than the images that were made available to us.

What happens with the number of stable patterns as more are learned?

We can observe some deviations in the number of stable patterns learned as the number of stored ones is increasing:

- Until a number close to $0.138N$ is stored, the percentage of stable patterns is steadily dropped
- Afterwards, until a total minima close to 150 stored patterns is reached, the percentage is decreasing with small deviations creating local maxima
- The, until all patterns are stored, we observe an increase in the percentage with small deviations creating local minima.

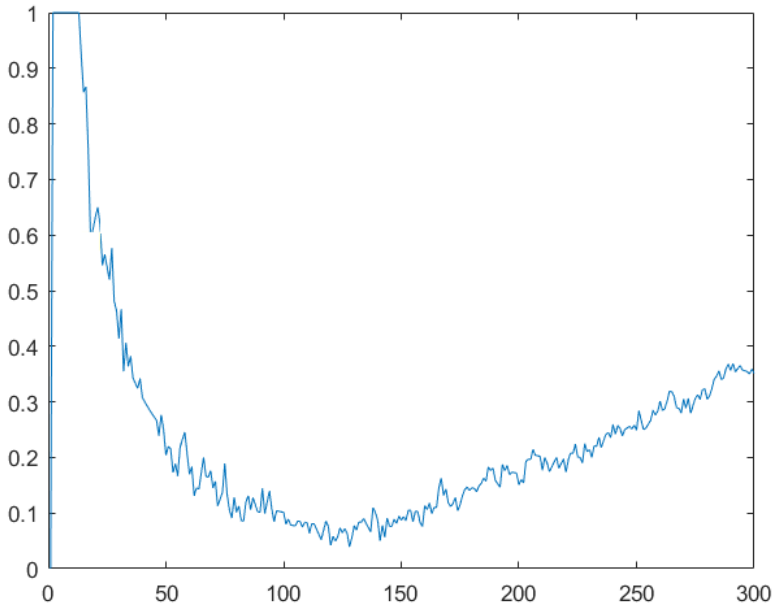

Figure 6: Evolution of the percentage of stable patterns learned until all 300 patterns are stored

What happens in convergence to the pattern from a noisy version (a few flipped units) is used? What does the different behavior for large number of patterns mean?

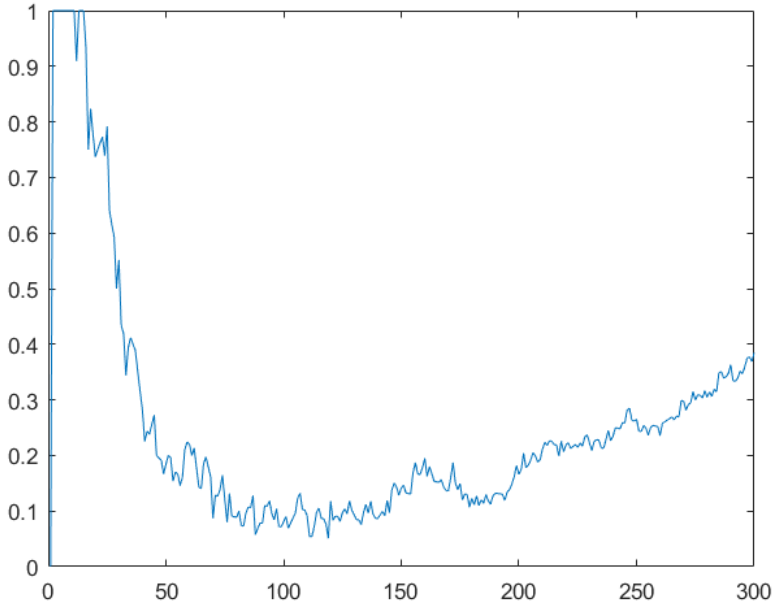

Figure 7: Evolution of the percentage of stable patterns learned using noisy data

What is the maximum number of retrievable patterns for this network?

The maximum number of retrievable patterns has a value of 24.

What happens if you bias the patterns, e.g. use sign(0.5+randn(300,100)) or something similar to make them contain more +1? How does this relate to the capacity results of the picture patterns?

The number of retrievable patterns decreased to 8. We present the corresponding plot:

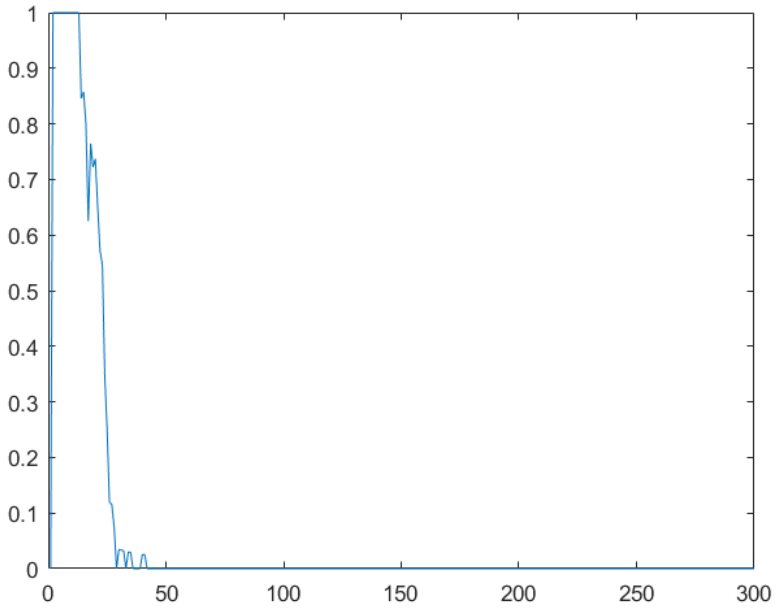

Figure 8: Biased patterns evolution

3.6 Sparse Patterns

Try generating sparse patterns with just 10% activity and see how many can be stored for different values of θ (use a script to check different values of the bias).

With 10% activity:

Theta	**Num of stored patterns**
0.02	4
0.04	7
0.05	8
0.07	26
0.1	10
0.17	8
0.25	22

What about even sparser patterns (ρ = 0.05 or 0.01)?

With 5% activity:

Theta	**Num of stored patterns**
0.01	7
0.02	20
0.059	19

With 1% activity:

Theta	**Num of stored patterns**
0.01	32
0.029	26