Biological Modeling of Neural Networks

Hopfield Model

Storage of Sequences of patterns in asymetric hopfield networks with delayed synapses

Miryam Chaabouni
Joseph Lemaitre

1 Exercise 1 : Standard Hopfield Network

1.1 Exercise 1.1 : Implementation

We created a class hopfieldNetwork which has the following attributes :

- N : number of neurons
- pattern : array of $P \times N$ states, where P is the number of patterns
- weight : array of $N \times N$ that represents the matrix of interaction between neurons
- x : state of the network at each time step, e.g., $x = \{1, -1, -1, \ldots\}$

and the following functions :

__init__ creates an instance of the class hofieldNetwork with the number of neurons N

makePattern creates a numpy array of N×P of ones, then randomly flips a given ratio of neurons in each pattern using the function numpy.random.choice.

makeWeight calculates the interaction weights using simple for loops according to the formula : $w_{ij} = \frac{1}{N} \sum_{m=1}^{P} \xi_i^\mu \xi_j^\mu$

dynamic updates the state of neuron i according to the formula : $S_i = \text{sign}(\sum_{j=1}^{N} w_{ij} S_j)$ using the weight matrix and the current state of the network

After creating an instance of the class hopfieldNework and creating the desired number of patterns, we run the simulation using the run function which does the following steps :

1. initialize the network by copying one of the patterns ξ^μ then flipping the state of randomly chosen neurons
2. Select a random neuron and update its state using the dynamic function, and do this for all the neurons of the network
3. repeat until convergence

For the convergence criterion, we store the value of the network x at each step, and after each iteration we compare the old value of x to the new one by a simple subtraction. If the difference is negligible, we stop. In addition, we set a time limit tmax = 100 steps, so that we exit even if the model doesn't converge.
The runAndPlot function performs the same steps as run, but plot at the same time the network and illustrates the convergence in a overlap w.r.t. time curve.
At each time step, we store the value of the overlap and the normalized pixel distance which will be useful later.

1.2 Exercise 1.2 : Pattern retrieval

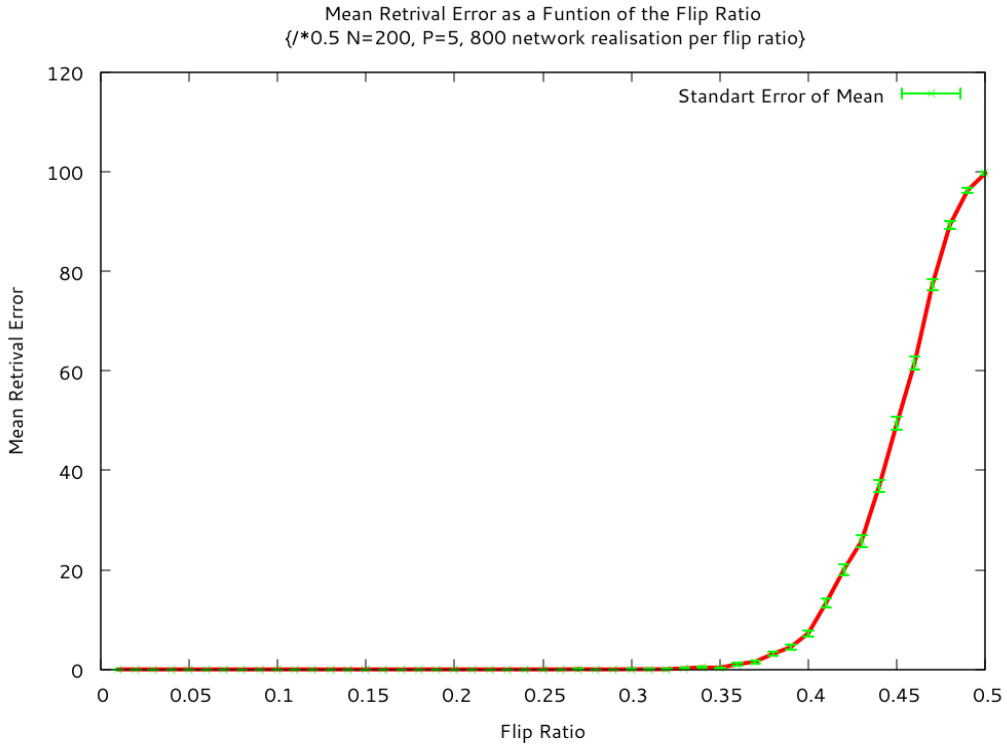
At this step we add two functions :

overlap calculates the overlap between the pattern ξ^μ and the current state of the network according to the formula : $m^\mu = \frac{1}{N}\sum_{i=1}^{N} \xi_i^\mu S_i(t)$

pixeldistance calculates the percentage of neurons in the network that differ from the pattern ξ^μ using the formula : $(1 - m^\mu) \times 100$, where m^μ is the overlap returned by the aforementioned function.

To get the retrieval error as a function of the ratio c of the flipped bits at the initialization of the network, we define the function patternRetrieval. We fix N=200 and P = 5. We take 50 values of c in the interval [0.01, 0.51]. For each value of c, we get run the simulation 50 times, and get the pixel distance at the end of each simulation. We show the average of the retrieval error in figure 1.2. The error bar represents the standard error of the mean, and is given by SciPy function stats.sem.
As we could expect it, the error is negligible ($< 1\%$) if we flip a reasonable ratio of the neurons at the initialization step. According to the figure, this threshold value is $c = 0.35$. As the number of flipped neurons increases, the error grows exponentially. For $c = 0.5$, meaning that we start with a network where half of the neurons are different from the retrieved pattern, the error reaches 100%.
(What happens when m=-1???)
We conclude that the Hopfield model works well only if the initial state of the Network is close to the target pattern.

Figure 1: Mean retrieval error as a function of the flip ration $c \in [0.01, 0.51]$.



1.3 Exercise 1.3 : Capacity estimation

The number of patterns that can be stored and retrieved correctly by a neurons network depends on the number of neurons. In Hofield model, the storage capacity of a network is defined by :

$$C_{stor} = \frac{P^{max}}{N} \tag{1}$$

where P^{max} is the maximum number of patterns that the network can retrieve correctly.
To study this capacity, we define the function maxLoad. To obtain the capacity of the Hopfield Network, we create a network of N neurons and a growing number of patterns $P \leq 51$. We fix $c = 0.1$. For each P we try to retrieve a random pattern 10 times and we get the mean error. When the mean error reaches 2% we stop and save the number of patterns created P_{max}. To speed up the simulation, we start from P=???. Finally, we calculate the average value of P_{max} for a given N, and we calculate the maximal load $\alpha_{max} = P_{max}/N$. We will do this for several values of N. The results are shown in table 1.3. The error on the value of α_{max} is calculated by ...

N	100	250	500
P_{max}	17	39	
α_{max}	0.17	0.156	–
α_{th}	–	–	–

Table 1: Capacity storage of a network of N neurons

As we could expect, we observe that the capacity of the network increases with the number of neurons. Moreover, we see that the maximal load that we obtain corresponds extremely well to the theoretical value.
Note that we only retrieve one pattern at a time. If we want to retrieve a sequence of patterns, the literature says that C_{stor} should not exceed 0.138[1], otherwise, the error will propagate at each pattern of the sequence. From our table, we see that this threshold is exceeded for N=250 and N=500. Therefore, we should to modify the model if we want to retrieve a sequence of patterns.

2 Exercise 2

2.1 Exercise 2.2

We used the heaviside filter function. We had a bit of freedom to define *sequencial behaviour*. We decreased lambda to get the minimum value for which all pattern are visited. We found :

$$\lambda_{min} \approx 0.9$$

For values below this one, the network get stuck in a particular pattern. Due to the random generation of some variables, the value of λ_{min} is approximative.

For the calculus of λ_{max}, we saw that for high lambda the network jump over some frame, not getting an overlap of one. Therefore we defined the sequencial behaviour as all state are visited with a overlap of 1, one after each other. We found :

$$\lambda_{max} = 3.6$$

2.2 Exercice 2.3

References

[1] W. Gerstner, W. Kistler, R. Naud, L. Paninski, *Neuronal Dynamics : from single neurons to networks and models of cognition*, Cambridge University Press, 2014.