The Cyclic Hopfield Network

NX-465 Mini-project **MP2**

Spring semester 2025

* Read the general instructions carefully before starting the mini-project. *

Introduction

The Hopfield network is a classical model that helps us understand how memories are stored and retrieved in neural systems. It operates by associating stored patterns with stable fixed points; if a small error occurs, the system naturally corrects itself by converging back to the correct pattern.

In this project, we will take the Hopfield model a step further. Instead of focusing on the stationary states, we will explore stable limit cycles where the neuronal states evolve over time following a sequence of patterns. You will be implementing this extension with minimal modifications to the classical model and investigate its ability to store and retrieve sequences of patterns. Further, you will also implement the cyclic Hopfield model as a continuous dynamical system, which will be compared to the discrete system and analysed using simple dimensionality reduction techniques.

Note: the project is intended to be solved using Python without the need for any specific library (other than the usual `numpy` and `matplotlib`). You are free to use other libraries if you want.
At the bottom of the project you will find a list of resources and references for further reading.

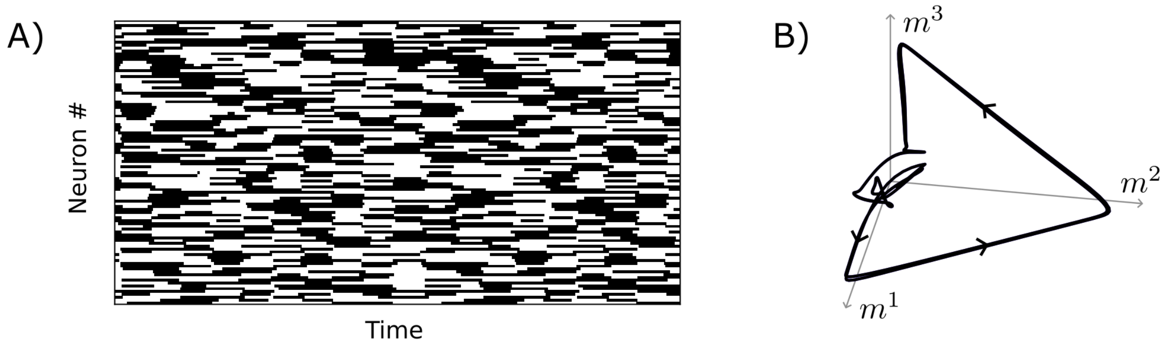

Figure 1: In the cyclic Hopfield network the neuronal states follow a sequence of patterns, which can be seen both in **A)** the neuronal states over time (exaggerated here for clarity) and **B)** their overlap with the patterns (Eq. (3)).

Ex 0. Getting Started: Cyclic Hopfield model

We start with a classical Hopfield-like setup, with a network consisting of N neurons with connectivity matrix w_{ij}. Each neuron i has a continuously-valued state $S_i \in [-1, 1]$, which updates according to

$$S_i^{(n)} = \tanh\left(\beta \sum_{j=1}^{N} w_{ij} S_j^{(n-1)}\right) \tag{1}$$

where (n) denotes the iteration number and β is the shape parameter of the transfer function.
The connectivity matrix will be defined as

$$w_{ij} = \frac{1}{N}\left[p_i^1 p_j^P + \sum_{\mu=2}^{P} p_i^\mu p_j^{\mu-1}\right] \tag{2}$$

where $P \leq N$ is the number of patterns that make up the connectivity. Unlike in the standard, static Hopfield model, here the patterns are *not* fixed points of the system, but rather points along a limit cycle. As such this system will be called a *cyclic* Hopfield model. We will be taking random patterns where each entry per pattern takes on a value $p_i^\mu \in \{-1, 1\}$ both with equal probability.

0.1. Write a function that generates network patterns as defined above, as a function of P and N.

As in the standard Hopfield model, we can define overlap variables both to better understand the network dynamics as well as to reduce computational complexity. For each pattern p^μ the corresponding overlap variable is defined as

$$m^{\mu,(n)} = \frac{1}{N}\sum_{i=1}^{N} p_i^\mu S_i^{(n)} \tag{3}$$

0.2. Rewrite the right-hand side of equation (1) in terms of the patterns and the overlap variables, making use of Eqs. (2) and (3).

0.3. Explain (without simulation) why this system can be considered as *cyclic*, by explaining how the network would evolve if the state of all neurons is initialised along pattern 1: $S_i^{(0)} = p_i^1$.
Hints:

- For this you may consider an ideal scenario where β is sufficiently large such that $\tanh(\beta x) \approx \text{sign}(x)$ and where the different patterns have no overlap between each other: $\sum_{i=1}^{N} p_i^\mu p_i^\nu = 0$ if $\mu \neq \nu$.
- To manually compute the state of the system at iteration 1, first compute the overlap variables $m^{\mu,(0)}$, and combine it with the equation derived in the previous question.
- From this, Eq. (3) can be used to find $m^{\mu,(1)}$. Extrapolate your result to many iterations to answer the question.

0.4. Verify your prediction from the previous question with a simulation of the more general, non-ideal scenario.

- Generate $P = 10$ patterns using your implementation from Ex. 0.1.
- Write code that simulates the evolution of the system for $N = 100$ neurons, $\beta = 4$, $n_{\max} = 20$ iterations. It will be useful to keep track of both the neural state S and the overlap variables m^μ for every iteration.
- **Important:** Implement the update step according to the update equation derived in Ex 0.2, instead of using Eq. (1) directly. This will strongly reduce the computational complexity and make your simulations run many times faster.
- Create a single plot showing the evolution of the P overlap variables m^μ over iterations, and comment on your findings.

Ex 1. Sequence storage and retrieval

Now that we have a working implementation of the cyclic Hopfield model we can evaluate some of its properties. As in the classical Hopfield network, one of the main properties of interest is the network capacity. We would like to find out how many patterns can be stored in the cycle and still be retrieved. We start by looking at what happens when we increase the number of patterns P in the network.

1.1. Redo the simulation of Ex. 0.4 but now with $P = 50$ patterns. You should observe a clear difference in the dynamics of the overlap variables over iterations, with no clear cyclic behaviour. Why does this happen?

To further investigate where this pattern retrieval breaks down, we need a formal definition of correct behaviour. For this, we say that the cycle is 'retrieved' if the network cycles between all patterns in the cycle, *in the correct order*, when initialised along one of the patterns.

1.2. Implement a way to determine if a cycle has been correctly retrieved in a simulation.
Hints:

- Compute for each iteration the index of the pattern with the largest overlap. What should the sequence of indices look like for a correctly retrieved cycle?
- Checking the retrieval can be done in two ways: Either through a function that can be evaluated at the end of the simulation, or through a check that is run every iteration. The latter can be useful for implementing early stopping of the simulation.

The *cycle capacity* of the network is defined as the maximum load $\alpha = P/N$ for which the cycle can still be retrieved. Note that due to the randomness of the patterns there is generally no hard boundary between retrieval and non-retrieval. Next we will be computing and comparing the cycle capacity of two networks with different sizes.

1.3. For $N = 100$ and for $N = 1000$, run a simulation for $\alpha \in \{0.05, 0.1, 0.15, \ldots, 0.4\}$, $n_{\max} = 2 \cdot P$, with 10 repetitions per α to get a more accurate estimate of the retrieval probability. Produce a single plot showing for both networks the fraction of retrieved cycles as a function of the network load α. Roughly what is the cycle capacity for each network?

1.4. If all went well, you should see in Ex. 1.3 that the more neurons there are in the network, the sharper the transition becomes between retrieval and non-retrieval. Explain where this difference comes from.

1.5. *(Bonus)* Above we have considered a network with only a single stored cycle. However, there is no reason we can't implement multiple cycles at the same time. Construct a network that stores two cycles simultaneously, each with length $P/2$ (where P is even), each of which can be retrieved separately based on the initialisation of the network. Find a P, N for which this works, and produce a set of plots to show that initialising into a pattern in cycle 1 retrieves cycle 1 without activating cycle 2, and vice versa. For this, show for each conditions the overlap variables for the patterns in the two cycles separately.

Ex 2. Continuous dynamical system

In this section we will be extending the cyclic Hopfield model from Ex. 0 and 1 as a continuous dynamical system. That is, we will replace the update rule of Eq. (1) defined on discrete time points by a differential equation defined for continuous time. The state of neuron i will be replaced from the variable $S_i^{(n)}$ to the variable $x_i(t) \in [-1, 1]$, where t denotes time, to differentiate it from the discrete system. The dynamics of neuron i are given by an ordinary differential equation on the state

$$\tau \frac{dx_i}{dt}(t) = -x_i(t) + \tanh\left(\beta \sum_{j=1}^{N} w_{ij} x_j(t - \tau_{\text{delay}})\right) \tag{4}$$

where τ is the time constant of the dynamics and τ_{delay} is the interaction delay between neurons, and all other parameters are as before. Unless mentioned otherwise, we will use $\beta = 4$, $\tau = 5$ ms and $\Delta t = 0.5$ ms, which is the spacing of time points at which the network will be evaluated. Convince yourself that this equation corresponds to a continuous version of the update rule in Eq. (1).

In this continuous setup we can again define our overlap variables

$$m^{\mu}(t) = \frac{1}{N} \sum_{i=1}^{N} p_i^{\mu} x_i(t) \tag{5}$$

to help us intuitively understand the dynamics and strongly reduce the computational complexity in the update step.

2.1. Simulate the dynamics of $N = 100$ neurons with minimal synaptic delay ($\tau_{\text{delay}} = \Delta t = 0.5$ ms), connected through $P = 10$ patterns for $T = 100$ ms, with initial condition $x_i(0) = p_i^1$ (the first pattern). What do you observe?

Hints:

- Write a method that runs the evolution of the neuronal states, according to Eq. (4) and making use of the overlap variables to reduce the computational cost. It should take as an argument the initial state $x_i(t = 0)$, and return the state of all neurons for all time steps. You may assume that the state of the neurons before the start of the simulation is constant ($x_i(t) = x_i(0)$ for $t < 0$).
- Eq. (4) can be integrated directly using the forward Euler method, with discrete time steps $t = t_0, t_1, \ldots$ ($t_k = k \cdot \Delta t$). It gives the following update rule:

$$x_i(t_{k+1}) = x_i(t_k) + \Delta t \cdot \frac{dx_i}{dt}(t_k) \tag{6}$$

- Plot the time evolution of the P overlap variables $m^{\mu}(t)$ as a function of time in one plot.

2.2. Now add synaptic delay to the network by setting $\tau_{\text{delay}} = 2 \cdot \tau$ and run the network again. You should now observe cyclic dynamics on the level of overlap variables. Produce a plot similar to that of the previous exercise, and explain intuitively why the synaptic delay is necessary for this cyclic behaviour. Roughly how much time does it take to complete a full cycle in terms of network parameters?

Now that we have a working setup we can again evaluate its properties, such as the cycle capacity as defined in Exercise 1.

2.3. Repeat Exercise 1.3 for the continuous model to find its cycle capacity, using $T = 2 \cdot P\tau_{\text{delay}}$. Produce a new plot and comment on how your findings compare to those of Exercise 1.

Hint: Here too you can implement an early stopping rule similar to Ex. 1.2 to strongly reduce simulation time.

2.4. *(Bonus)* In the classical Hopfield network as covered in the class, there are also fixed points at states *in between* the patterns (e.g. $x_i^* = \frac{1}{2}\left(p_i^1 + \ldots + p_i^M\right)$), with the stability of the fixed point depending on if the number of patterns used for the state (M) is odd or even. Test if this also holds in the cyclic dynamic Hopfield model from the exercise. (Either three activations in a single cycle, or preferably three cycles of different length simultaneously.)

Ex 3. Dimensionality analysis

An experimentalist often only has access to a recording of the activity of the neurons, without knowing the underlying connectivity structure or dimensionality of the dynamics. Simply looking at the activity itself can make it difficult to understand what the network is doing, especially in complex, real-world scenarios. As such, it is a major challenge in neuroscience to find ways to understand or explain the computations or processing in large networks of neurons. In this section, we will be applying a simple dimensionality reduction technique to see if we can correctly retrieve the dimensionality of the underlying dynamics.

Principal Component Analysis

The main goal behind PCA is to find a set of components (like patterns) along which the neuronal state most varies over time, with the idea being that axes along which the state varies are relevant for the functioning of the network. Suppose that we want to represent our N-dimensional states as a combination of a small number of components $K \leq N$. That is, we want to find vectors $\left\{\vec{v}^1, \ldots, \vec{v}^K \mid \vec{v}^k \in \mathbb{R}^N\right\}$ and weights (also called loadings) $\{l_1(t), \ldots, l_K(t)\}$ such that

$$x_i(t) \approx \sum_{k=1}^{K} v_i^k l_k(t) \tag{7}$$

This formulation should look very familiar to you. It is the same formulation as how the input to a neuron i can be written as a sum over patterns and overlap variables as in Ex. 0.2, except in this case we have an approximation. Equality can generally only be obtained when $K = N$.

In PCA, the directions $\vec{v}^k$ are exactly the directions in the N-dimensional space in which the neuronal state $\vec{x}(t)$ most varies over time, with higher k components having smaller contributions to the total sum. The vectors $\vec{v}^k$ are obtained exactly as the eigenvectors of the cross-covariance matrix $C \in \mathbb{R}^{N\times N}$ of the neuronal states $\vec{x}(t)$, ordered by the magnitude of the corresponding eigenvalues in decreasing order.

$$C_{ij} = \mathrm{Cov}\left(x_i, x_j\right) \tag{8}$$

$$C\vec{v}^k = \lambda_k \vec{v}^k \tag{9}$$

3.1. Implement a function that calculates the cross-covariance matrix C and returns the eigenvalues and eigenvectors of C in decreasing order of importance.

Hint: You may use the functions `numpy.cov` and `numpy.linalg.eigh`. Have a look at the documention for how to correctly use these functions.

To get a low-dimensional approximation of our data we need to find the loadings $l_k(t)$. As the vectors $\vec{v}^k$ are orthonormal, the loading for component k is exactly the overlap between the neuronal state at time t and the corresponding vector:

$$l_k(t) = \sum_{i=1}^{N} v_i^k x_i(t) \tag{10}$$

3.2. Run a simulation using your model from Ex. 2, with $N = 500$ neurons, $P = 10$ patterns, $T = 500$ ms, initial state $x_i(0) = p_i^1$. Calculate the loadings $l^k(t)$ for the first P components ($k = 1, \ldots, P$) using the definitions and your functions from above, and produce a single plot showing how the loadings vary over time, similar to the plot of Ex. 2.2. Comment on the observed behaviour and on how it compares to the plot of overlap variables.

Despite the seeming similarity between Eqs. (3) and (10), the trajectories of $l^k(t)$ may look very different from those of $m^\mu(t)$. However, this does not say anything about the quality of the approximation of x as defined in Eq. (7).

3.3. Make two reconstructions of the neuronal state $\hat{x}_i(t)$ using Eq. (7) from the first K components $\vec{v}^k$ and loadings $l_k(t)$, for $K = P/2$ and $K = P$. Produce a single plot showing together the original and two reconstructions of the time evolution of the neuronal state for one example neuron. How do the two reconstructions compare?

If we want to know how many components we need to accurately describe the data, we could of course make comparisons as done in the above exercise. But there is a much simpler way. PCA is constructed to exactly capture as much variance of the data as possible through several components. The variance that is captured by each component k is actually exactly the square of the corresponding eigenvalue λ_k as obtained in Ex. 3.1 above. The *fraction of explained variance* by the k-th component can be calculated as

$$\%EV(k) = \frac{(\lambda_k)^2}{\sum_{k=1}^{N} (\lambda_k)^2} \tag{11}$$

3.4. Produce a bar plot of the $\%EV$ as a function of component number (up to $K = 20$). What do you observe? And can you correctly estimate from this the number of patterns P used in the network?

3.5. In general, explain a procedure for how an experimentalist can use PCA to estimate the dimensionality of recorded data.

3.6. *(Bonus)* If the neuronal dynamics can be understood through the dynamics of the P overlap variables $m^{\mu}(t)$, why might we expect the neuronal state itself to be mostly constrained to a P-dimensional subspace? And why is the subspace spanned by the patterns the only relevant part for the dynamics?
Hint: Consider again your answer to Ex. 0.2 for the first part. For the second, consider what happens to any neuronal state that is (partly) orthogonal to all patterns.

Resources

The classic Hopfield network was covered in weeks 5 and 6 of the course, corresponding to Chapter 17 *of the* Neuronal Dynamics *course book.*
You can learn more on cyclic Hopfield networks (also called association chains) in the classic paper Temporal Association in Asymmetric Neural Networks *by H. Sompolinsky and I. Kanter (1986).*

Here is an excellent review in Annual Reviews on the challenges and successes in extracting low-dimensional dynamics as a way to understand network functioning, titled Computation Through Neural Population Dynamics*, by S. Vyas, M.D. Golub, D. Sussilo and K.V. Shenoy (2020).*
For further reading on attractor networks such as the one in this project have a look at this great review in Nature Neuroscience: Attractor and integrator networks in the brain *by M. Khona and I.R. Fiete (2022).*