Week 5 - Simulating a Dynamical System

1 Introduction

We are finally in the project phase! You will simulate the evolution of one of the following biologically-inspired dynamical systems: Hopfield networks [Hop82] or Turing pattern formation [Tur90].

During this first exercise session you will choose one of the two models and write code to simulate its evolution in time. Today you will focus on writing a simple script to *just make things work*. During the next exercise sessions you will be asked to improve the code by adding new features, restructuring the code to make it more readable, including tests and improving code efficiency.

The two projects pose similar challenges, so feel free to choose your favourite one based on your biological interests.

1.1 Hopfield network

The Hopfield network is a computational model for associative memory, proposed by John Hopfield in 1982. It models a simple mechanism to explain how a neural network could store representations (i.e., the neural activity corresponding to a certain concept) in the form of weighted connections between neurons. In this project you will implement the iterative process which allows to retrieve one of the stored (memorized) patterns starting from the representation of a new (unseen) pattern.

1.2 Turing pattern formation

In 1952, Alan Turing proposed a reaction-diffusion model to explain the formation of many natural patterns, such as the stripes of a zebra or the spots of a leopard. For this project we will not focus on the chemical and biological principles behind morphogenesis, but we will derive an iterative algorithm to solve the partial differential equations which have been proposed to model this phenomenon. As a result, we will observe the formation of beautiful patterns starting from a homogeneous initial state.

2 Logistics of projects

The projects are solved in teams! Find your teammates and work together to solve the project of your choice (2-3 students per team). Note that you should not change the topic over the course of the semester.

Irrespective of your project, put the code to answer the questions below in a single script named *main.py* and upload it to the GitHub repository for your group (called 'BIO-210-2022-groupX') which was created by us.

3 Project 1 - Hopfield network

3.1 Network definition

Consider a Hopfield network with N neurons. In the classical Hopfield model, each neuron can be in only one of two states at a given time, corresponding to firing (active) and non-firing (passive). In this implementation, *firing* corresponds to the value 1, while *non-firing* corresponds to the value

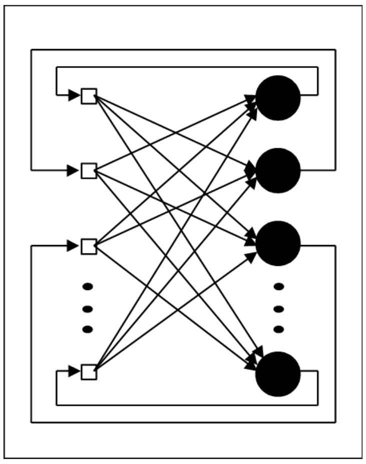


Figure 1: A recurrent network, like the Hopfield one. The output of the network is the input for the following step of its evolution. The weights of the connections between the input and the output are given by the synaptic weight matrix W.

-1. Given this convention, at any given time point the state of a Hopfield network with N neurons can be described by an N-dimensional binary vector in $\{-1, 1\}^N$.

The Hopfield network is *fully connected*, meaning that each neuron is connected to the other neurons in the network via synapses (i.e. directed connections). This connectivity structure can be represented by a dense $N \times N$ matrix W, whose elements $w_{ij} \in [-1, 1]$ represent the connection (or synaptic) weight from unit j to unit i. We consider the simplified case in which the connection weights are symmetric without self-connections. This translates into the following constraints: $w_{ij} = w_{ji} \quad \forall i, j \in \{1, ..., N\}$ and $w_{ii} = 0 \quad \forall i \in \{1, ..., N\}$.

The Hopfield network is *recurrent*. This means that the network state, corresponding to the vector of the states of each neuron (firing: +1, or non-fining: -1) at the current timestamp, is fed to the network to generate the new state. The simplified image 1 shows how the network output is then used as an input for it.

We will show that a Hopfield network with N neurons can store a certain number of network firing patterns $\mathbf{p}^\mu \in \{-1, 1\}^N, \mu \in \{1, ..., M\}$ in its synaptic weights. This means that, if the network is presented with a pattern $\tilde{\mathbf{p}}$ similar to one of the memorized patterns $\mathbf{p}^\mu$, the network will converge to the memorized pattern.

3.2 Network connectivity

The Hebbian learning rule [Heb05] is the simplest one to train a Hopfield network. It is based on the principle ”Neurons that fire together, wire together. Neurons that fire out of sync, fail to link”. This translates into the following network connectivity:

$$w_{ij} = \frac{1}{M} \sum_{\mu=1}^{M} p_i^\mu p_j^\mu \quad i \neq j \tag{1}$$

We can observe that the weight is the average of the contribution of each pattern to the synaptic weight. Each pattern contributes positively to a certain weight if the state of the two connected neurons is the same, and negatively otherwise.

Your task:

- Set $N = 50$ and generate $M = 3$ random binary patterns, i.e. N-dimensional vectors with elements in $\{-1, 1\}$.
- Create the weight matrix W with the Hebbian rule, as defined in Equation (1).

Once you have created the matrix, verify that:

- the elements on the diagonal are 0
- the matrix is symmetric
- all the weights are in $[-1, 1]$

3.3 Model dynamics: Retrieving a memorized pattern

We now want to show that the network remembers the patterns that it has memorized. When presented with a new, random pattern $\mathbf{p}^{(0)}$, the network starts an iterative process, governed by the following update rule:

$$\mathbf{p}^{(t+1)} = \sigma\left(\mathbf{W}\mathbf{p}^{(t)}\right) \tag{2}$$

where $\mathbf{W}$ is the weight matrix and the function $\sigma(x)$, applied to each element of $\mathbf{W}\mathbf{p}^{(t)}$, is defined as

$$\sigma(x) = \begin{cases} -1 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{3}$$

Your task:

- Generate a pattern $\mathbf{p}^{(0)}$ by changing 3 (random) values of one of the memorized patterns.
- Define a loop which applies the update rule defined in Equation (2) until convergence (or up to a maximum of $T = 20$ iterations).
- Verify that the iterative process converged to the original memorized pattern.
- What happens if you change more values of the initial pattern?

4 Project 2 - Turing pattern formation

The Turing pattern formation mechanism can be modeled with two time-dependent reaction-diffusion equations, describing the evolution and the interaction between an activator $u(x, y)$ and an inhibitor $v(x, y)$ morphogen. In the formation of the zebra's stripes, the activator and inhibitor morphogen then influences the concentration of white and black pigments in each point (x, y) of the zebra's skin, respectively. In the following we will omit the dependency on space (x, y). Together, the activator and the inhibitor determine the spatiotemporal evolution of the morphogen which, starting from a homogeneous initial state, can create coat patterns similar to the ones observed in nature. This process can be modeled with the following differential equations, that describe the evolution of the two morphogen over time:

$$\frac{\partial u}{\partial t} = \gamma f(u, v) + \nabla^2 u \qquad \frac{\partial v}{\partial t} = \gamma g(u, v) + d\nabla^2 v \tag{4}$$

The equations above each consist of two terms: one diffusion term ($\nabla^2 u$ in the equation for u and $d\nabla^2 v$ in the equation for v) and one reaction term ($\gamma f(u, v)$ and $\gamma g(u, v)$). The diffusion terms describe how the respective morphogen diffuses in space. The reaction term couples the two equations and describes how the two morphogens interact with each other while they diffuse. The equations for these interaction terms are

$$f(u,v) = a - u - h(u,v), \quad g(u,v) = \alpha(b-v) - h(u,v), \quad h(u,v) = \frac{\rho uv}{1+u+Ku^2}$$

with scalar parameters a, b, α, ρ and K.

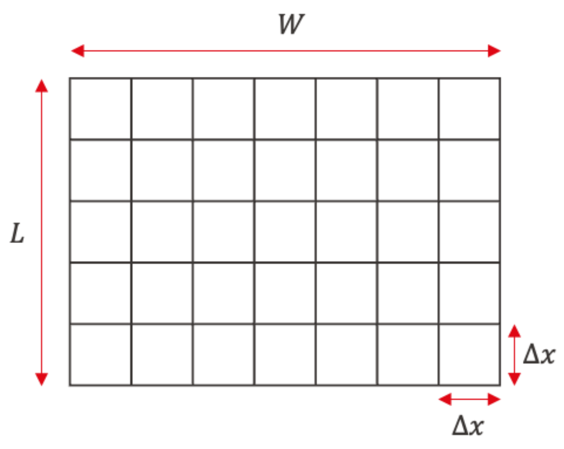
Crucially, the source of the instability is the parameter d of the reaction-diffusion equation for v, which, when larger than 1, makes the inhibitor diffuse faster than the activator. The parameter γ represents the size of the domain. This allows us to carry out experiments in the same domain, while obtaining patterns which are observed in animals of different sizes. The other parameters are specific to the characteristics of the activator and the inhibitor.

To find a unique solution to Equation (4), we need to specify an initial condition (i.e. the values of u and v at time 0) and a boundary condition (a constraint on the values of the solution at the border of the domain). For the latter, we choose a Neumann boundary condition (i.e., zero derivative at the boundary). Since the equations admit a stable state, depending on the values of the parameters, we can impose an initial condition equal to a random perturbation of the stable state.

One way to efficiently solve this equation in a numerical way is by discretization.

4.1 Discretization of the problem

The reaction-diffusion equations defined in (4) do not admit, in general, an analytical solution. However, we can compute a numerical solution of the problem, by deriving a discrete approximation of the continuous equations in (4) with the finite difference method. Finite difference methods are one of the simplest ways to approximate differential problems. The fundamental idea is to approximate the derivatives of a continuous function with the difference between its values in neighboring points. We consider a rectangular domain of size $L \times W$. In such a structured domain it is possible to discretize the partial differential equation with the finite differences method, both in space and time. This means that we are defining a grid of squares of size $\Delta x \times \Delta x$, as in the following figure:



We want to approximate the solution of the differential equation by estimating its value on each of the nodes of the grid. Therefore, u and v will be represented by two matrices, $\mathbf{u}^{(t)}$ and $\mathbf{v}^{(t)}$, which denote the estimated value of u and v on each node of the grid at time step t. Their size is therefore $M \times N$, with M and N the number of nodes along y and x, respectively. To compute the update rule for the values of $\mathbf{u}^{(t)}$ and $\mathbf{v}^{(t)}$, we need to transform the continuous equation (4) into a discrete one. The resulting update rule writes as follows:

$$u_{ij}^{(t+1)} = u_{ij}^{(t)} + \Delta t \left[\gamma f(u_{ij}^{(t)}, v_{ij}^{(t)}) + \frac{u_{i-1,j}^{(t)} + u_{i+1,j}^{(t)} + u_{i,j-1}^{(t)} + u_{i,j+1}^{(t)} - 4u_{ij}^{(t)}}{\Delta x^2} \right] \tag{5}$$

$$v_{ij}^{(t+1)} = v_{ij}^{(t)} + \Delta t \left[\gamma g(u_{ij}^{(t)}, v_{ij}^{(t)}) + d\frac{v_{i-1,j}^{(t)} + v_{i+1,j}^{(t)} + v_{i,j-1}^{(t)} + v_{i,j+1}^{(t)} - 4v_{ij}^{(t)}}{\Delta x^2} \right] \tag{6}$$

where Δt is the temporal step size and Δx, f, g, and d are defined as above.

We have mentioned that the Turing pattern formation mechanism is a reaction-diffusion equation. In fact, besides the diffusion operator ∇^2, both the equations for u and v present a forcing term, f and g, respectively. For this first week you will ignore the reaction part of the equation and simply solve a purely diffusion problem. This will allow you to familiarize with the solution of time-dependent dynamical systems and to verify that the diffusion operator has been correctly implemented. The solution of a time-dependent diffusion problem can be found with the following equation:

$$u_{ij}^{(t+1)} = u_{ij}^{(t)} + \Delta t \frac{u_{i-1,j}^{(t)} + u_{i+1,j}^{(t)} + u_{i,j-1}^{(t)} + u_{i,j+1}^{(t)} - 4u_{ij}^{(t)}}{\Delta x^2} \tag{7}$$

which means that the value of the matrix $\mathbf{u}^{(t+1)}$ in the position (i, j) depends on the values of $\mathbf{u}^{(t)}$ in the same position and in the neighboring ones.

Your task:

- Write the update rule of the diffusion problem. Given an input matrix $\mathbf{u}^{(t)}$ and the parameters Δt and Δx, your code must compute the updated matrix $\mathbf{u}^{(t+1)}$, preserving the size of the input matrix $\mathbf{u}^{(t)}$. Pay attention to the updates of the values at the borders of the matrix (first and last column, first and last row), as they depend on elements which are not defined (e.g., to compute the value of the element at the position $(0, 0)$, you would need the values at $(-1, 0)$ and $(0, -1)$). In this case, use the value of the closest available element of the matrix (e.g., if the update rule requires a value at the position (-1, 5), use the value at the position (0, 5) in order to model the boundary).
- Define a (5×5) matrix of your choice and compute the update rule by hand. Compare the result you have obtained by the one of the update rule at the previous point. Remember to deal with the borders of the matrix correctly.

4.2 Simulating the diffusion problem

In this second part you will simulate the evolution of the dynamical system defined in Eq. (7)

Your task:

- Use the previously defined update rule and set $\Delta x = 0.1$, $\Delta t = 0.0001$. Define the initial matrix $\mathbf{u}^{(0)}$ of size 9×9 as 0 everywhere apart from the central element, which is set to 1. Run 10 iterations of the diffusion update rule and store the intermediate solutions.
- Observe the list of the solutions that you have obtained, representing the time evolution of the system. Verify that the maximum value of the matrix decreases, while the neighboring values increase. This equation simulates, e.g., the diffusion of smoke in a room, starting from a high concentration of smoke in a single point.

References

[Hop82] J J Hopfield. “Neural networks and physical systems with emergent collective computational abilities”. In: *Proceedings of the National Academy of Sciences* 79.8 (1982), pp. 2554–2558. ISSN: 0027-8424. DOI: 10.1073/pnas.79.8.2554. URL: https://www.pnas.org/content/79/8/2554.

[Tur90] Alan Mathison Turing. “The chemical basis of morphogenesis”. In: *Bulletin of mathematical biology* 52.1 (1990), pp. 153–197.

[Heb05] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.