Coursework 1

This coursework relates to the Hopfield network. This coursework may appear before we have covered Hopfield networks in the lectures, but you can still get started!

In a Hopfield network you have a network of neurons; these neurons can be in an up state:

$$x = 1 \tag{1}$$

or a down state

$$x = -1 \tag{2}$$

In this lab we will consider a Hopfield network with $d = 11$ "neurons". the state of the network can be represented as a $d \times 1$ column vector:

$$\boldsymbol{x} = \{x_1, x_2, \dots x_d\}^\top \tag{3}$$

Our network also has recurrent weights, which we can represent as the $d \times d$ weight matrix W.

In the synchronous version we will use here all the neurons are updated at the same time; I will put a hat on the new value, obviously in the next iteration the new value becomes the current value and loses the tilde.

The sign nonlinearity we will use for the Hopfield network maps positive values to 1 and zero and negative values to -1:

$$\mathrm{sgn}(r) = \begin{cases} -1 & \text{if } r > 0 \\ 1 & \text{otherwise.} \end{cases} \tag{4}$$

The update rule is:

$$\tilde{x}_i \leftarrow \mathrm{sgn}(\sum_j w_{ij} x_j). \tag{5}$$

In vector notation, Equation (5) is

$$\tilde{\boldsymbol{x}} \leftarrow \mathrm{sgn}(\mathrm{W}\boldsymbol{x}). \tag{6}$$

You can store patterns in a Hopfield network by setting the synapse strengths, that is the w_{ij}'s. The more brain-like version has a learning rule for changing w_{ij}.

Here, however, we will anticipate the end result of that learning and stipulate that, for $i \neq j$

$$w_{ij} = \frac{1}{N} \sum_a x_{i;a} x_{j;a} \tag{7}$$

where

- a indexes the patterns stored in the network,
- $x_{i;a}$ is the activation of the i^{th} node in the a^{th} pattern and
- N is the number of patterns.

If $i = j$, then this weight would represent a connection from the i^{th} neuron to itself. In this lab, we will assume there are no such connections and enforce that these diagonal weights are zero.

The idea is that after the synapses have been set, that is after the network has learned, if a partial patter is presented, so some but not all of the values of x_i^a for one of the a's, the other x_i^a will adjust to complete the pattern.

In the `coursework1` folder you will find programmes `seven_segment.jl` and `seven_segment.py`. These contain a function for converting an 11-component vector of ones and -1's into an old-fashioned seven-segment digit[1] and a number: the first seven digits correspond to the seven-segment display and the remaining four code for the number in a sort of binary where the zeros have been replaced with -1. It also contains three patterns: one, three and six.

The goal of this coursework is to extend one or other of these programmes to include a Hopfield network to store the patterns **one**, **three**, and **six** using the formula for w_{ij}

$$w_{ij} = \begin{cases} \frac{1}{N} \sum_a x_{i;a} x_{j;a} & \text{if } i \neq j \\ 0 & \text{if } i = j. \end{cases} \tag{8}$$

In the code there are also two test patterns; Your programme should update these synchronously until convergence and print out the patterns at each iteration.

This coursework is intended to check you understand Hopfield networks, you are not being asked to do anything over elaborate beyond that. In summary you should:

[1] `https://en.wikipedia.org/wiki/Seven-segment_display`

- Create a weight matrix of w_{ij}.
- Fix the weight values to store the three patterns **one**, **three**, and **six**.
- Write a function to evolve the network according to the McCulloch-Pitts formula; this should be done synchronously so all the nodes are updated at each timestep. Assume the threshold $\theta = 0$.
- For each of the two test patterns, evolve the patterns until they stop changing, printing the pattern at each step.

If you want you can add to the above a function to calculate the energy of a configuration

$$E = -\tfrac{1}{2} \sum_{ij} x_i w_{ij} x_j = -\tfrac{1}{2} \boldsymbol{x}^\top \mathrm{W} \boldsymbol{x} \tag{9}$$

and print out the energy of the three learned patterns, of the test patterns and any patterns formed as the patterns are updated.

This coursework is not marked, but the instructor(s) would be very much entertained and grateful if you could share your solutions, especially if

- You complete this project in an especially obscure programming language, or unusually clever way.
- You delve deeper in to the mechanics of Hopfield networks, perhaps exploring forms that use $\{0, 1\}$, biases as well as weights, alternative updates or learning rules, etc.
- You develop an especially beautiful routine for displaying 7-segment displays in the terminal using ASCII art or semigraphics.

Bonus material

If you've found the above too simple, consider this bonus material. This is beyond the scope of this unit and will not be on the exam, but you may find it enriching.

Seven-segment display drivers accept four bits of binary-coded decimal input, which can range from 0–15. They sometimes render the numbers 10–15 as the hexadecimal letters A–F. We've provided these patterns in the Python and Julia source files, but did not ask you to encode them, for reasons we will now explore.

1. Try storing more than just the patterns **one**, **three**, and **six** in your Hopfield network. After a certain point, you should exceed the *capacity* of the network.
2. Demonstrate (in code) that not all patterns can be recalled correctly by finding a pattern that is not a stable fixed-point under the update rule.
3. Demonstrate (in code) the existence of *spurious* local minima: Patterns that are absent from the training data, but still emerge as fixed-points of the update rule, and have a similar energy to patterns that were in the training data.
4. Discuss (or explore in code) strategies you could use to increase capacity or address the issue of spurious patterns.