PROBLEM SET 3

Learning and Memory

Note: this problem set is more exploratory and less guided than the previous problems. You are encouraged to think of your own questions and model behavior to explore.

Contents

1 Sequence learning in birdsong

1.1 Introduction

This problem explores a model of neural sequence development in the birdsong motor system. A schematic of the anatomy of this system is shown in Figure 1. We will be focusing on the HVC nucleus, this nucleus has neurons that fire bursts at particular moments during the adult song.

First, you will need to read the paper by Okubo et. al. [2], we will be working with the model in that paper (see Figure 5 in the Okubo paper). Download the Matlab code used to generate this figure from the course website.

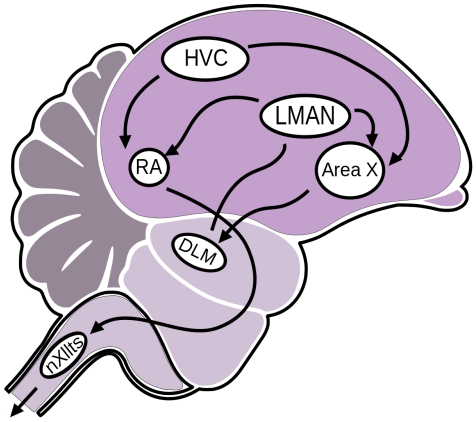

Figure 1: Anatomy of the songbird motor system. The HVC nucleus contains neurons that generate brief bursts of spikes at a specific time during the adult song [2]. *(Image from Wikipedia)*

1.2 Exploration

First, you will need to familiarize yourself with the Matlab code. A lot of it is for visualization, instead you mainly need to focus on the function `HVCIter.m` and main script `RUN_HVCModel.m`. Once you are comfortable with the code, work on the following exploration:

1. Run the script `RUN_HVCModel.m` and verify that you reproduce Figure 5 from Okubo et. al. [2] (during learning the code updates a plot of the fraction of neurons in and width of each chain, the paper figure is generated after the model has run for 1400 iterations). Describe (in your own words) what the different panels in Figure 5 from the paper are showing.

2. What would a good statistic (a single number) be to quantify the result of one run of the model for one set of the parameters? What statistic could represent the dynamics of splitting (how fast the chain splits)?

3. Choose two parameters from lines 29-38 of the main script `RUN_HVCModel.m`.

4. In your groups, manipulate these variables one at a time and build intuition for how it affects chain formation. First, modify the parameters by hand, and then loop over a range of interest (use the statistics from #2 to quantify the effect of your manipulation).

5. Describe your manipulation(s) and what you learned.

6. Is the effect manipulating each variable independent? What happens if you modify them together?

7. Which parameter(s) is the chain splitting behavior most sensitive to?

8. Which parameters can be changed (and over what range) without modifying the model behavior?

1.3 Robustness to noise

For this question, we will explore the robustness of the model to noise. Specifically, we will look at adding noise to either the neurons or to the synaptic weights.

1. The neurons in the simulation are binary (they are either on (1) or off (0)). Add code to `HVCIter.m` that randomly flips the output of each neuron with some small probability (e.g. 0.05). This models noise since with probability 0.05 the neuron has a random chance of flipping its output.

2. As you increase the probability of flipping, what happens to chain formation in the network?

3. Go back to the noiseless case (no flipping). Instead, add some small random Gaussian noise (using the `randn` command) to the weight matrix in the `Update weights` section of `HVCIter.m`. Make sure to add noise *before* the weights are clipped to be between 0 and `wmax`.

4. Come up with an interesting range of noise strengths to add. What happens as you increase the noise strength?

5. (Optional) This simulation relies on analog weights (continuous values). What happens if the weights are discrete? Add a constraint that the weights must be either 0 or 1 (You can do this by thresholding the weights). What happens to chain formation?

2 Hopfield model

2.1 Introduction

The goal of this problem set is to explore the behavior of the Hopfield Model [1]. Specifically, we are interested in addressing several questions regarding the time it takes to recall a memory and the sizes of basins of attraction in the Hopfield model. Make sure to download the Matlab example code `HopfieldExample.m` from the zip archive on the course website. You will modify the code to answer the questions below.

One note about the implementation–you do not have to scan parameters (like number of patterns P, and basin of attraction size K) at a level of resolution of every P and K, as this would take too long. Feel free to scan them at lower resolution–and also you don't have to scan them over the entire range–only over a

range in which interesting quantities like recall probability, overlap, and recall time vary in interesting ways. Basically, imagine that you are doing research on the Hopfield model, and you have several hypotheses that as the number of stored patterns increase, basins of attraction get smaller and recall probability goes down, and recall time (conditional on successful recall) goes up. Lets say you want to present numerical evidence for these conjectures in the form some figures for a paper. Then part of your job would be not only figuring out what to plot (which is outlined in the different parts of the problem) but also over what ranges and what resolution of parameters to plot at (for which we gave initial suggestions that are most likely suboptimal). So feel free to make your own decisions about what exactly to plot.

2.2 Basins of Attraction

First, we will investigate the basins of attraction. Recall that a *basin of attraction* in the Hopfield model is set of neural activity patterns, that when set as the initial condition, all converge to the same attractor state (see Figure 2 for a schematic). We would like to understand properties of the basins of attraction, because they directly correspond to how corrupt a recoverable memory can be. Here, we will explore how big the basins of attraction are as a function of the number of patterns in the network (P) and the amount of corruption (K).

1. The code (see lines 15-17) fixes P random patterns stored in a Hopfield network of size $N = 1000$. For P ranging from 20 to 600, start the network from an initial condition consisting of K spin flips away from a randomly chosen stored pattern (for K ranging from 1 to 500), and check to see if the network goes back to the stored pattern–i.e. if the fixed point that the network arrives at has overlap > 0.9 with the stored pattern. (You will have to write your own `for` loops to accomplish this).

2. For each K and P, repeat this procedure multiple times (say for ≥ 100 trials) to estimate the probability that a network will be able to recall a stored pattern from a corrupted version.

3. Plot this probability as a heat map as a function of K/N and P/N. This will give you an indication of how the size of basins of attraction shrink as the number of stored patterns increases. Also, by plotting this in terms of the ratios, you will obtain a plot that will remain invariant as N increases.

4. Comment on how this plot relates to the Hopfield capacity of $P_{\max} = 0.14N$.

5. Instead of plotting the probability that the overlap with the chosen stored pattern is > 0.9, plot also the mean overlap with the chosen stored pattern across trials as a heat map as a function of K/N and P/N. This provides a view of the location of nearby valleys in the energy landscape (valleys near the one containing the uncorrupted chosen pattern).

2.3 Recall probability and recall time

Local minima of the energy function are memories of the Hopfield network. Although our learned (desired) patterns are local minima, there are also spurious local minima. Next, we will look at the probability that we recall one of the original P patterns given a random initial condition (as opposed to a spurious local minima).

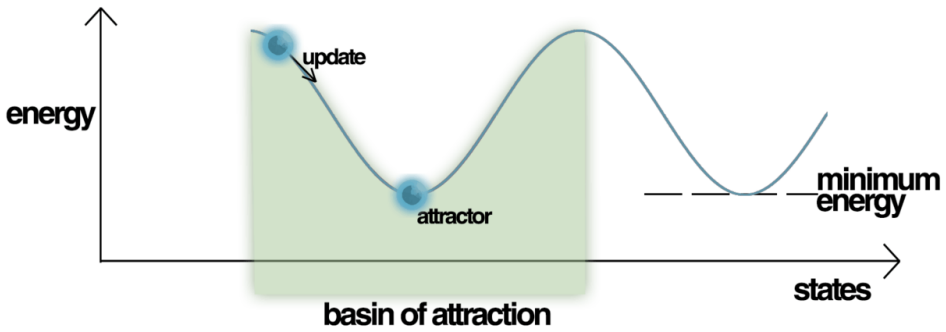

Figure 2: Energy Landscape of a Hopfield Network. The memories in the network are attractor states (local minima) of the energy landscape. Given an initial condition (e.g. a partial memory) in a particular basin of attraction, the dynamics of the network are such that the network settles at the attractor. *(Image from Wikipedia)*

This is the *recall probability*. We will also look at the *recall time*, the number of update steps required to reach one of the existing patterns.

1. Again, we will start with P random patterns stored in a Hopfield network of size $N = 1000$.
2. Now for P ranging from 20 to 600, run the network multiple times (say 100 times) from a completely random initial condition.
3. Record the fraction of times that the network achieves a recall state (final fixed point has a large overlap (¿ 0.9 in absolute value)) with one of the P patterns.
4. Plot this fraction as a function of P/N.
5. Also, plot the mean and standard deviation (across trials in which a recall state was found) of the time it takes to find a recall state, as a function of P.
6. This plot will terminate as some early value of P because at larger values, the network will never be able to find a recall state from a random initial condition.

2.4 Exploration

This part is completely optional, but feel free to play around with the system and report anything interesting that you find.

References

[1] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[2] Tatsuo S Okubo, Emily L Mackevicius, Hannah L Payne, Galen F Lynch, and Michale S Fee. Growth and splitting of neural sequences in songbird vocal development. *Nature*, 528(7582):352–357, 2015.