PROBLEM SET 3

Learning and Memory

Contents

1 Hopfield model

Introduction

The goal of this problem set is to explore the behavior of the Hopfield model [**?**, **?**]. 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.

1.1 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 (Figure 1). 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).

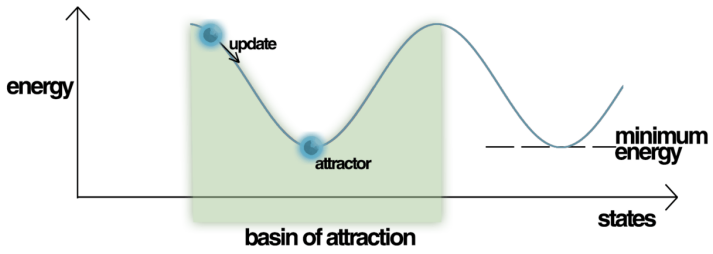

Figure 1: Energy Landscape of a Hopfield Network. The memories in the network are attractor states (local minima) of the energy landscape. Given an initial condition (e.g. a partial memory) in a particular basin of attraction, the dynamics of the network are such that the network settles at the attractor. *(Image from Wikipedia)*

A. The code (see lines 15-17) fixes P random patterns stored in a Hopfield network of size $N = 1000$ by defining the network connectivity (J) as the outer product of the patterns. If you run the code, it will build the Hopfield network, initialize the network at either a random location (if `InitCondType` is 1) or at a location that is K bit flips away from one of the original patterns (if `InitCondType` is 2), and finally run the dynamics of the network. There is code at the bottom that visualizes the overlap of the current network state with each of the learned patterns. Successful recall occurs when, after running the dynamics, when the network is attracted to one of the input patterns (one of the overlap bars should shoot up close to 1, as in Figure 2). With `InitCondType` set to 1, verify that the network successfully recalls a stored pattern (we will define successful recall if the final maximum overlap is at least 0.9). If you increase the number of patterns, P, to 200, does the network recall the pattern?

B. For P ranging from 20 to 600, start the network from an initial condition consisting of K bit flips away from a randomly chosen stored pattern (`InitCondType` = 2) for K ranging from 1 to 500 (you don't need to test every value in this range), 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 (the variable `successful_recall` computes this). You will have to write your own `for` loops to accomplish this. You do not need to test every value of P and K in the range–for example, in Matlab, you can use `for P=20:5:600` to scan P from 20 to 600 in steps of 5, for example. For $K = 1$, around what value of P/N does the system transition from successful to failed recall? As K gets bigger, does this critical P/N value increase, decrease, or stay the same?

C. 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. Plot this probability as a heat map (using the `imagesc` or `pcolor` commands) 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.

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

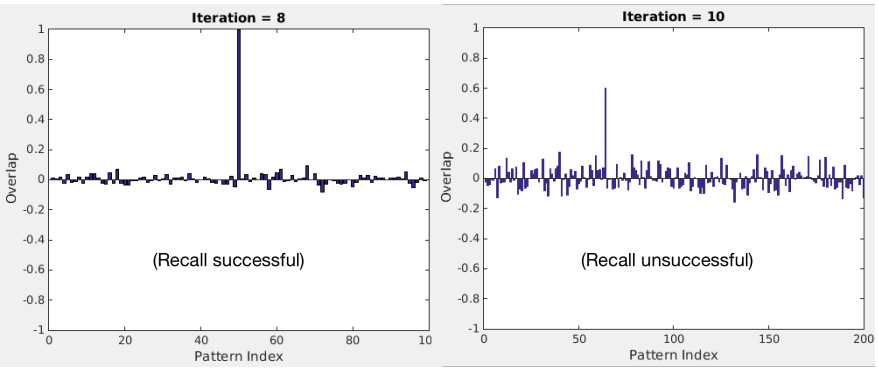

Figure 2: Successful (left) and unsuccessful (right) memory recall in a Hopfield model.

1.2 Recall probability and recall time (extra credit)

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). 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. Note that this problem is extra credit.

A. Again, we will start with P random patterns stored in a Hopfield network of size $N = 1000$. For P ranging from 20 to 600, run the network multiple times (say 100 times) from a completely random initial condition (`InitCondType` = 1). Record the fraction of times that the network achieves a recall state (just like before, if the final state has a large overlap (> 0.9 in absolute value)) with one of the P patterns. Plot this fraction as a function of P/N.

B. Also, plot the mean and standard deviation (across trials in which a recall state was found) of the *time* (number of iterations of the dynamics) it takes to find a recall state, as a function of P. 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.

C. What is the relationship between the recall time and recall probability?

2 Perceptron Learning

A. Implement Perceptron learning for binary patterns. An input pattern ξ^μ generates an output y^μ that we would like by learning to be equal to σ^μ. Recall that the perceptron learning rules is:

$$\Delta w_j = \begin{cases} 0 & \text{if } \sigma^\mu = y^\mu \\ 2\eta\xi_j^\mu\sigma^\mu & \text{if } \sigma^\mu \neq y^\mu \end{cases} \tag{1}$$

   (a) Implement inputs and outputs for the Boolean AND function. Show that it can be learned.

   (b) Implement inputs and outputs for the Boolean XOR function. Show that it can *not* be learned.

   (c) **Bonus:** implement perceptron learning for multi-layer perceptrons to show that the XOR function can be classified once a hidden layer is added.

B. How many patterns can a perceptron learn? Implement a 10 input neuron Perceptron. Generate datasets with different number of randomly generated binary patterns. Train the Perceptron. See how large a dataset can still successfully train.

C. **Bonus:** the Hopfield model weights are chosen so as to make the memory patterns stable. Can you phrase generating a set of weights for a recurrent network that also perform memory completion like the Hopfield model. Hint (shockingly): try to think how you can use Perceptron learning to do this.