Neuro 120 Homework 4: Short and Long Term Memory
Due: Nov 15 in class

In this homework you will investigate models of short term and long term memory based on recurrent networks. Starter code and data for the assignment is provided on Github. Where details are not specified, you can make your own choices. We are interested in the correct conceptual conclusions. Work in groups of 2-4 students. You may use any resources or code you find online, but must properly attribute it. Submit all code you write, and the plots you produced to generate your answers.

1. **Short term memory.** How can a neural network store information about inputs that occurred in the recent past? One possibility is through recurrent connections: a network of neurons can be interconnected with themselves, such that they excite their own activity. Unlike a feedforward network, this can allow activity to change over time, imbuing the system with memory for recent inputs. As a model of this we consider a linear firing rate model of a neural network,

$$\tau \frac{d}{dt} r(t) = -r(t) + Wr(t) + VI(t)$$

where $r(t) \in R^N$ is a vector containing the activity of N neurons, $W \in R^{N \times N}$ is the recurrent matrix of connections between neurons, $I(t)$ is a scalar input over time, $V \in R^{N \times 1}$ is a vector of connections from the scalar input to each neuron in the network, and τ is a time constant setting the speed of the dynamics. This system can be simulated using Euler's method (which you may remember from the Hodgkin-Huxley homework), where the derivative is replaced by a finite difference. The starter code in `shortterm_mem.m` implements most pieces of this for you, but you will have to correctly fill in the commented line marked `your code here`. We will consider sending in a brief pulse, $I(t) = 1$ if $1 < t < 2$ and zero otherwise, and see how long the activity of the network persists for different kinds of recurrent weights W. This persistent activity is a memory trace that encodes the fact that an input occurred in the past (similarly to how ripples in a pond convey that a rock was thrown in recently).

   (a) First examine the dynamics without any recurrent connections ($W = 0$). How long after the end of the pulse does activity die out? Here you can say that activity has died out when the largest magnitude r is less than 0.1. You can do this by hand or by finding the first time for which all(abs(r) < 0.1)) in matlab (you may find the command `find` useful). Also note that because each neuron receives the same input and has the same dynamics, you'll only see one activity trace because all 50 neurons are doing the same thing.

   (b) Now add autapses ($W = c\mathbb{I}$) where $\mathbb{I}$ is an identity matrix and c is the weight scale. Autapses add connections from each neuron back to itself, but no connections between neurons in the network. Try weight scales of $c = 0.9, 1$, and 1.1, and find the time at which activity dies out. Does the system ever go unstable (activity diverges towards $\pm\infty$)? The third plot shows the magnitude of the eigenvalues of the recurrent matrix W. How are these related to the weight scale c (you do not need to prove this, you can guess based on the plots)? If we consider the dynamics in this case, it is $\tau \frac{d}{dt} r(t) = -r(t) + c\mathbb{I}r(t) + VI(t) = (c-1)r(t) + VI(t)$. In the memory period when the input is off, this is just $\tau \frac{d}{dt} r(t) = (c-1)r(t)$. Use this to qualitatively explain the results you've seen. Which of the three settings of c would seem like the best choice for stably storing a memory?

   (c) While the autapse network can store memories, all its neurons do the exact same thing, and they are not interconnected with each other. Neither of these facts is true in many brain systems. Consider a more realistic network with random orthonormal recurrent weights $W = cU$ where U is a random orthogonal matrix ($U^T U = \mathbb{I}$, this is computed for you by finding the SVD of a random Gaussian matrix). Again investigate the dynamics for weight scales $c = 0.9, 1$, and 1.1. When does activity die out? Is the memory behavior qualitatively similar to part (b)? For $c = 1$, do the activities in the network ever stop changing? How are the eigenvalues of the recurrent matrix related to c (you do not need to prove this, you can guess based on the plots)? Do the eigenvalue magnitudes continue to explain the memory properties of the network?

(d) Finally, in biological networks, synapses are noisy. Examine the robustness of the memory to small perturbations of the weights in the network by adding a small amount of Gaussian noise to the orthogonal weights. To do this you can adjust the `noise_scale` parameter. Set `noise_scale`= 0.2 and examine the behavior for `weight_scale`= 0.9 and 1. To get a better picture of what is happening, increase the max simulation time `T` to 100 for this section. Because randomness is involved, run your results a few ($\leq$ 5) times (we don't need plots from all runs, just one representative run). How does a small amount of noise in the recurrent synapses affect the stability of the memories? Explain this result with reference to the eigenvalue magnitudes of the perturbed matrix. Would a memory with $c = 0.9$ or 1 be better in practice, where noise is inevitable?

2. **Long term memory in the Hopfield network.** The memory system in Problem 1 relies on persistent neural activity. Under that scheme, to maintain a memory for a long time would require neurons to fire in a sustained way the whole time, which could be energetically costly. Additionally, as you saw, the system would have to be finely tuned to maintain stable memories on long time scales. Moreover, even a brief disruption of the activity would cause the memory to be lost forever (e.g., you might actively rehearse a phone number to maintain it in memory, but if you then spill coffee on yourself and momentarily stop rehearsing the number, it will fly out of your mind for good). For these reasons, activity-based memory is usually considered to be a model of short term memory.

   One way to build a longer-term memory is to embed memories in the synapses or weights of a recurrent network. This strategy maps most directly to the hippocampus, where neurons in the CA3 subfield have conspicuous recurrent connections between them. Bilateral lesions to the hippocampus profoundly impair the formation of episodic memories (memories for specific events like where you parked your car or what you ate for lunch). Here we will investigate an abstract model of how this memory process might work called the Hopfield network. A Hopfield network is a simple recurrent network where network activities are binary (0 or 1). The activities $h \in R^N$ in the network evolve in discrete time as

$$h[t+1] = f(Wh[t] - 1/2)$$

   where here $W \in R^{N \times N}$ is a recurrent weight matrix and $f(u)$ is a step function which is 0 if $u < 0$ and 1 if $u \geq 0$. Notice that again, these dynamics are recurrent and activity will evolve over time.

   In the Hopfield network, we are given a collection of binary activity patterns to memorize $\{\xi^\mu\}, \mu = 1, \cdots, P$, where each pattern $\xi^\mu \in R^N$. Here we will take these patterns to encode questions and answers from the midterm you recently took, encoded as binary images of dimension 32×400 pixels. Starter code is provided in the file `longterm_mem.m`. The first two sections of `longterm_mem.m` will generate and plot this dataset.

   Memories are stored by carefully choosing the weight matrix W so that each of the patterns ξ^μ is a *fixed point* of the dynamics. At a fixed point, the neural activity at the next time step is the same as the neural activity at the current time step, and the dynamics stop. That is, we will choose W such that $\xi^\mu = f(W\xi^\mu - 1/2)$ for all $\mu = 1, \cdots, P$. This means that if $h[t] = \xi^\mu$, then $h[t+1]$ will also equal ξ^μ, and so on. The third section will generate the recurrent weight matrix W that satisfies this property, using a method called the generalized perceptron learning rule. The perceptron learning rule is similar to gradient descent learning, but adapted to work with binary activities (which have discontinuous derivative so that standard gradient descent can't be applied). The hope is that, if we start the neural activity nearby ξ^μ, it will also be attracted to the fixed point, thereby recalling the memory.

   The Hopfield network is an example of a *content addressable* memory: the way you query the memory is by giving a partial input $h[0] = \tilde{\xi}^\mu$, where $\tilde{\xi}^\mu$ is some partial or corrupted version of ξ^μ. The network will then recurrently excite itself, starting from this partial input. Eventually, the activity will stop changing and if the recurrent weights are carefully chosen, the network will hopefully "fill-in" the correct pattern $h[\infty] = \xi^\mu$. Content addressable memory stands in contrast to the memory in a computer, which is *location addressable*. In a computer memory, there are labeled slots which you can

fill with different patterns (corresponding to something like "store the string '-65mV' in slot 5"). To query the memory, you need to supply the location of the memory you're interested in ("what is in slot 5?"), which has nothing to do with the content.

(a) Implement the Hopfield network update and try recalling midterm answers by supplying the question as the input $\tilde{\xi}^{\mu}$. To implement the update, complete the anonymous function in the commented line marked `your code here`. Out of the 7 questions stored, how many does the network get correct?

(b) Look in more detail at the dynamics for questions 3, 5, and 7. Qualitatively, what happens in these cases?

(c) Try giving a hint by supplying part of the answer for questions that were wrong. How many characters are required before the correct answer is recalled?

(d) As you saw in the preceding parts, some memories are more successfully embedded in the network than others. To investigate this further, we can try randomly flipping some fraction of bits of the original input pattern, and see when the memory recall fails. For questions 6 and 7, adjust the `corruption_probability` parameter to find the approximate percentage of corrupted bits at which memory recall begins to fail. Based on your results, is question 6 or 7 more strongly memorized by the network? Can you make out what the correct answer is from looking at the corrupted inputs?

(e) Briefly discuss some advantages and weaknesses of content addressable and location addressable memories. When might one be better than the other?