9.49 / 9.490 Neural Circuits for Cognition
Homework 3
Due: Friday November 1, by midnight.

In this assignment, we will explore random recurrent networks and neural networks for feedback control and function learning.

1) **Random recurrent networks and chaos.** Consider the case of a simple rate-based network with dynamics given by

$$\tau \frac{d\mathbf{s}}{dt} = -\mathbf{s} + \mathbf{w} f(\mathbf{s}) + \mathbf{b} \tag{1}$$

where W is a set of random weights (chosen independently and identically from a Gaussian distribution with mean 0 and variance g^2/n; set your seed to a fixed value when generating these random weights so that on different runs you get exactly the same random matrix). Let the neural transfer function f be the hyperbolic tangent, $f(x) = \tanh(x)$, and let $\tau = 10$ ms. Using $n = 500$ neurons, $\mathbf{b} = 0$ and the Euler step-size dt ($dt = 0.2$ ms), we are going to explore the dynamics of this network.

a. Visualization of the state-space dynamics of the system. Set $g = 1.1$, and set your initial condition $\mathbf{s}(0)$ to be independently sampled from a uniform random distribution over the $[0, 1]$ interval. Set a different fixed seed for this random initial condition so you have the same initial condition every time (important for the parts below). Run the simulation for a duration of 8s. Setup your code to generate a 2D projection to visualize state-space activity of the network, in the three following ways: i) Generate one plot of $s_i(t)$ versus $s_j(t)$ where i, j are two cells chosen at random. ii) Generate two random vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ of dimension n, and plot $\mathbf{v}_1^T \mathbf{s}(t)$ versus $\mathbf{v}_2^T \mathbf{s}(t)$; this is based on the Johnson-Lindenstrauss lemma as covered briefly in class; iii) Plot $s_i(t)$ versus $s_i(t + T)$, where T is at least a few times larger than τ (experiment with a few values of T for good visualization of the response of the network); this is based on Takens' embedding theorem as covered briefly in class.

b. The periodic regime and transition to non-periodic irregular responses with increasing coupling strength. Step g (in steps of 0.1, from 0.9 to 1.6), and observe the changes in the behavior of the state-space trajectories (show your results). In these runs, also plot the summed autocorrelation $\sum_i C_i(T)$ where $C_i(T) = \sum_t s_i(t) s_i(t + T)$ of the responses of all neurons; for computational speed, do this for lags T from -2s to 2s. Observe that even in the irregular response regime, there is a characteristic time-scale that looks a little like periodic

behavior. In the next part, we will plot a metric more diagnostic of chaotic behavior.

c. The Lyapunov exponent quantifies how quickly two infinitesimally close trajectories diverge from each other. The exponent λ is defined by:

$$|\delta \mathbf{s}(t)| = |\delta \mathbf{s}(0)| e^{\lambda t}.$$

If the states remain within a bounded region of state space while the Lyapunov exponent is positive, it is a signature of chaotic behavior. To practically estimate the Lyapunov exponent of your random network: for a few coupling strengths g ($g = \{0.9, 1.1, 1.2, 1.4, 1.6\}$), run your code and save $\mathbf{s}(t)$; then run it a second time, adding a small perturbation pulse $\mathbf{b} = 0.005\mathbf{1}$, where $\mathbf{1}$ is the vector of n ones, for a duration 1ms starting at time $t = 5s$. Call the resulting states $\mathbf{s}_{pert}(t)$, and define $|\delta \mathbf{s}(t)| = \sum_i |s_i(t) - s_{i,pert}(t)|$.

Plot $\delta \mathbf{s}(t)$ for each g, and see how the state difference diverges exponentially for large g. To estimate λ for each g, plot $|\delta \mathbf{s}(t)|$ in a semilog plot (log axes in $|\delta \mathbf{s}|$ and linear in t) and obtain the best least-mean-square linear fit over the interval $t = [5.05, 6.05]$s. Plot these values of λ as a function of g, highlighting when λ is positive. For these cases, verify that the states remain in a bounded region of state space by plotting $\frac{1}{n} \sum_i |s_i(t)|$. Thus, the network dynamics are consistent with chaotic behavior.

2) **Feedback control-based learning with a chaotic reservoir.**

a. Download code from the class website (either `forceexternalfeedbackloop.m` or `forceexternalfeedbackloop.py`), and run it. This code a. sets up a random recurrent network exactly as in Problem 1 above, b. specifies a target function ft (in this case, it's a sum of four sine waves over a fixed time-interval), c. constructs a linear readout z of the recurrent network with plastic readout weights wo; the readout is also fed back into the network as an input, d. uses recursive least squares (RLS) acting on the error e between z and ft to train the plastic readout weights. The weight changes are large and rapid: RLS computes the desired weights to produce zero error at that moment; it is not an incrementally learning gradient rule. Thus, from the moment learning is turned on, the instantaneous weight changes make the output z equal the target function ft. The chaotic dynamics perturb the output in each time-step so the learning rule has to keep making weight adjustments. After some amount of learning, the network can reproduce the target with its learned weights held fixed (the test phase). Submit the output plots, together with a few-sentence explanation in your own words of

how the network works. Which parts of the learning rules for P and wo depend on the error?

b. The role of chaos: Modify the recurrent network parameter g so that the dynamics are periodic rather than chaotic (it's the same parameterization as in Problem 1, so you know what this regime is). Verify that this is true by plotting the network activities, since this network has $n = 1000$ neurons. With non-chaotic dynamics, how well does the network learn compared to when the dynamics are chaotic? Try changing the learning rate and the learning time to see if this improves learning. Try to interpret your findings.

c. Train the network simultaneously on two target output functions, $f1t = ft$ and a new target $f2t$ using two different sets of readout weights $w1$ and $w2$ that are trained in parallel. Let the target functions have the same frequency content (the same four sine waves summed with different amplitude mixture), and modify both weights simultaneously using two different error functions $e1$ and $e2$. Plot the results during training for both target functions, and then during testing.

d. Do the same as in c. above, but let $f2t$ have distinct frequency content from $f1t$. Specifically, $f1t$ contains frequencies $\omega \times \{1, 2, 3, 4\}$ where ω is some underlying frequency; let $f2t$ contain frequencies $0.8\omega \times \{1, 2, 3, 4\}$. Does it still work? Why do you think you see what you do?

e. Bonus questions: Is it possible to train the network as desired in d. above, if the readout weights $w1$ and $w2$ each only see and are each trained on non-overlapping halves of the outputs of the recurrent network? In other words, the recurrent network has n neurons, but the update rules for $w1$ and $w2$ can each see only half of the recurrent units, and can only construct linear combinations of these? On the one hand, we are trying to reduce interference by asking different subsets of the recurrent neurons to produce different frequencies; on the other hand, the the recurrent network is not modular. You can experiment with another piece of code, supplied as a matlab file, in which the recurrent weights of the network are also trained. Does the network do better with learning two distinct functions in this case?